THÈSE DE DOCTORAT DE L'UNIVERSITÉ PARIS VI

SPÉCIALITÉ INFORMATIQUE

Présentée par Ana Belén ABRIL GARCÍA

Pour obtenir le titre de

DOCTEUR DE L'UNIVERSITÉ PARIS VI

ESTIMATION ET OPTIMISATION DE LA CONSOMMATION DANS LES DESCRIPTIONS ARCHITECTURALES DES SYSTÈMES INTÉGRÉS COMPLEXES

Soutenue le 21 juin 2005, devant le jury composé de

Mme.	Nathalie JULIEN	Rapporteur
M.	Christian PIGUET	Rapporteur
Mme.	Nathalie DRACH-TEMAM	Examinateur
M.	Jean GOBERT	Examinateur
M.	Frédéric PÉTROT	Examinateur
M.	Habib MEHREZ	Directeur de thèse

Résumé

Cette thèse propose une méthode d'estimation et d'optimisation de la consommation d'énergie des systèmes embarqués matériel-logiciel. Le but de notre travail est de fournir un moyen d'estimation de la consommation dans les descriptions de haut niveau (modèles comportementaux en C) des systèmes incluant tous les composants, y compris les nouveaux. De telles analyses sont nécessaires pour sélectionner la meilleure architecture de système et l'organisation logicielle d'une application particulière en termes de consommation ainsi que pour tester l'application des techniques de basse consommation au niveau architecture de système.

Le point de départ est la description architecturale du système utilisée en simulation. Nous nous concentrons sur le niveau cycle-précis et l'outil de simulation que nous utilisons est un simulateur de systèmes qui modélise les composants matériels en langage C/C++. Les modèles sont très abstraits et utilisent des automates d'états pour représenter le comportement par cycle du matériel.

Le simulateur de systèmes cycle-précis est interfacé avec un simulateur de jeux d'instructions du processeur et tous les deux sont enrichis avec des modèles d'énergie qui prennent en compte les opérations exécutées par transition dans l'automate d'état des composants. Pendant la simulation, l'énergie correspondante à chaque opération est détectée et accumulée en donnant la dissipation instantanée et totale par cycle de chaque module et de l'ensemble du système. Nous obtenons l'évolution de l'énergie, ce qui permet d'identifier les points critiques (quels sont les modules qui consomment le plus, quand et pourquoi) et de localiser les périodes actives et inactives de chacun. Cela permet l'analyse de l'implantation d'une gestion de la consommation au niveau architecture de système.

Cette méthode a été testée sur un exemple de système, un décodeur MPEG4 développé par Philips. Nous avons décodé des images INTRA et évalué l'énergie dissipée par image. Les valeurs ont été calibrées à partir d'estimations de bas niveau obtenues avec un outil industriel. La précision est donc très bonne : l'erreur totale du système est estimée inférieure à 6 % par rapport aux mesures physiques. Des techniques de réduction de la consommation ont été appliquées et analysées comme l'utilisation d'une autre mémoire, les horloges inhibées et la réduction de la tension/fréquence. Cela a permis une réduction de la consommation du système de 93 %.

Mots clés

Estimation de la puissance/énergie - basse consommation - co-estimation matériel-logiciel - système embarqué - système sur puce - modèles d'énergie - simulation cycle-précis - TSS - SystemC - horloge inhibée - réduction tension/fréquence - gestion dynamique de la puissance.

Abstract

This thesis proposes a method for energy consumption estimation and optimisation in hardware-software embedded systems on a chip. The aim of our work is to provide a simulation framework enabling power estimations for high level descriptions (behavioural C models) of the system that include all the hardware components, also the new ones. Such analysis are needed to select the best hardware architecture of the system and the software organization for a particular application in terms of power consumption, and to apply low power techniques at system architectural level.

The starting point is an architectural description of the system used for simulation. We focus on the cycle-accurate level and the simulation tool we use is a system simulator that models the hardware components in C/C++-language. These models are very abstract and use states machines to model the hardware behaviour per cycle.

This system-level cycle-accurate simulator is linked to a processor instruction set simulator, and they are extended with energy models that take into account the operations executed per transition into the state machine of the components. During a simulation, the energy corresponding to each operation is detected giving the instantaneous and total energy consumption per cycle for each component and for the whole system. We obtain the evolution of the energy, and detect peaks of consumption (which components consume more, when, why) and the active and idle periods of each one. This allows the analysis of a power management implementation at system architectural level.

The method has been tested with a system example, a MPEG4 decoder developed at Philips. We have decoded INTRA images and evaluated the energy consumption per image. Values have been calibrated with low level power estimations obtained from an industrial estimation tool. We find a very good accuracy : the total error of the system has been estimated lower than 6 % from physical measurements. Low power techniques have been applied and analyzed like the use of another memory, clock gating and voltage/frequency scaling. That permits to reduce the consumption cost of the system on 93 %.

Keywords

Power/energy estimation - low power consumption - hardware-software co-estimation - embedded system - System on Chip - cycle-accurate simulation - energy model - TSS - SystemC - clock gating - voltage/frequency scaling - dynamic power management.

Remerciements

Je souhaite exprimer toute ma reconnaissance à Monsieur Jean Gobert, Senior Consultant Engineer à Philips France, qui a co-dirigé mes travaux de recherche. Il a toujours été disponible et a constamment suivi mes travaux. Je tiens à le remercier pour ses conseils avisés, ainsi que sa relecture attentive de ce manuscrit. Son soutien m'a été d'une aide plus que précieuse.

Je remercie également Monsieur Habib Mehrez, Professeur à l'Université Paris VI, pour ses conseils, son soutien ainsi que ses corrections pendant ces années où il a été mon directeur de thèse. Je tiens également à lui exprimer ma gratitude pour la confiance et la gentillesse qu'il m'a témoignées.

J'adresse mes remerciements à Madame Nathalie Julien, Professeur au Laboratoire d'Électronique des systèmes TEmps Réel (LESTER) de l'Université de Bretagne Sud et Monsieur Christian Piguet, Professeur à l'Université de Neuchâtel et chef du secteur ultra low power au CSEM, qui m'ont fait l'honneur d'être les rapporteurs de ma thèse. De même je tiens à remercier Madame Nathalie Drach-Temam, Professeur à l'Université Paris VI, d'avoir accepté d'être examinateur de cette thèse.

Je tiens à remercier Monsieur Frédéric Pétrot, Professeur au laboratoire TIMA de l'Institut National Polytechnique de Grenoble, pour son aide, ses multiples compétences et l'intérêt qu'il a porté au déroulement de mes travaux. J'ai beaucoup appris à ses côtés et je lui adresse toute ma gratitude.

Je remercie spécialement Carolina Miro et Thomas Dombek pour leurs compétences, leurs précieuses questions, conseils et amitié, ainsi qu'à l'ensemble des membres de l'équipe Philips Applied Technologies-Paris et les thésards et étudiants d'ASIM qui ont contribué de près ou de loin à mes travaux de thèse.

Je remercie Monsieur Thierry Brouste, chef de groupe de Philips Applied Technologies-Paris et Monsieur Alain Greiner, directeur de l'équipe ASIM du LIP6, de m'avoir accueillie dans leurs laboratoires respectifs dans les meilleurs conditions.

J'adresse mes plus chaleureux remerciements à tous mes amis pour m'avoir toujours soutenu et encouragé pendant ces longues années, tout spécialement à Annick, Beatriz, Belen, Carlos, Chaden, Elisabete, Eduardo, Francisco, Gianmarco, Hugo, Ines, Ivan, Jaume, Juan Carlos, Kathrin, Kayoko, Odile, Robin, Samuel, Sara, Takashi, Teresa et Valerie. Mille mercis pour votre temps et joie de vivre.

Y finalmente, un agradecimiento muy especial a mi padre, a mi madre, a mi hermano Francisco y a toda mi familia, por su apoyo incondicional, su paciencia, su confianza y por estar siempre cerca de mi a pesar de la distancia.

Quidquid nitet, notandum

(Tout ce qui brille doit être noté)

Proverbe latin

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction

Le marché des appareils numériques portables se développe rapidement grâce à la demande croissante de produits tels que téléphones mobiles, ordinateurs portables ou agendas électroniques, comme nous l'illustrons dans la figure 1.1. En même temps, le développement des technologies a permis d'augmenter énormément la densité des circuits permettant l'intégration de tout un système sur une même puce. Cela permet l'utilisation de composants très complexes et fortement spécialisés qui sont des systèmes embarqués alliant hautes performances et taille réduite, où de nouvelles fonctionnalités comme la vidéo peuvent être introduites.

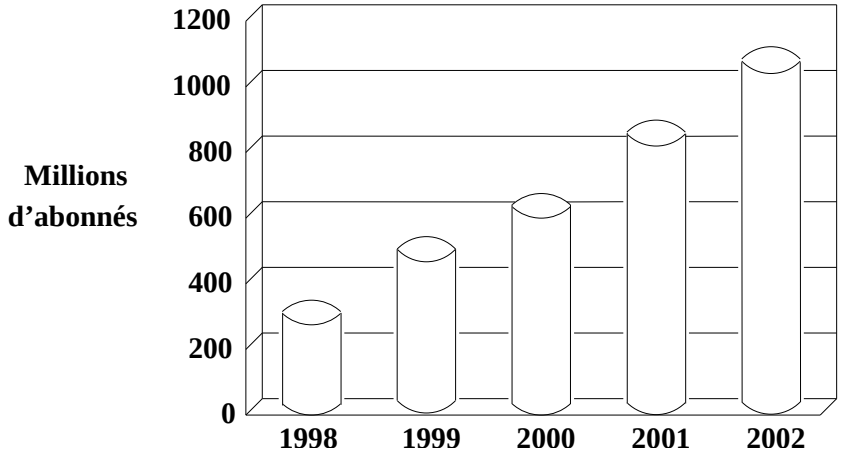

FIG. 1.1 Les abonnés aux opérateurs de téléphonie mobile dans le monde, source [PSZ04]

Un tel système embarqué est conçu pour exécuter une application particulière connue à l'avance. Il doit respecter le cahier de charges de l'application, en termes de coût, de performance et de consommation d'énergie. Une architecture hybride combinant matériel spécialisé (pour la performance et la faible consommation) et des processeurs à usage général (pour la souplesse) s'est imposée dans ce type de systèmes.

La consommation est importante seulement depuis quelques années. Tout d'abord, les problèmes principaux des concepteurs étaient la réduction de la surface en silicium du circuit et l'augmentation de la vitesse de traitement des processeurs. La surface est importante parce que le prix de la puce aug-

mente considérablement avec sa taille. L'augmentation de la vitesse de traitement permet d'atteindre les contraintes en vitesse demandées dans les spécifications et d'implanter de nouvelles fonctionnalités qui demandent de plus en plus de puissance de calcul. L'efficacité en consommation d'énergie est devenue une caractéristique très importante, principalement dans l'optimisation de l'autonomie des terminaux portables.

La contribution en consommation la plus importante dans les dispositifs portables est celle due aux dispositifs d'affichage comme l'écran et aux circuits de réception et d'envoi des signaux de radiofréquence [vB00]. Celle due aux circuits intégrés numériques devient aussi d'un intérêt considérable grâce à la croissance de la complexité de ces circuits. Le nombre de transistors augmente très rapidement, tout comme le nombre de points mémorisants, très gourmands en énergie et les nouvelles applications multimédia accroîssent rapidement l'activité du circuit, provoquant une augmentation très forte en consommation d'énergie.

Pour être autonomes ces dispositifs portables fonctionnent avec des batteries. Cependant, les batteries ont une capacité limitée qui rend critique la consommation électrique du système. Les progrès sur les batteries sont beaucoup plus lents que l'évolution de la complexité de traitement et la puissance de calcul, comme le montre la figure 1.2. Cela crée un problème de manque de ressources énergétiques dans le circuit dû aux durées de vie des batteries de plus en plus réduites par rapport à l'énergie demandée. En conséquence, l'étude et la réduction de la consommation devient un enjeu majeur pour ce type de circuits.

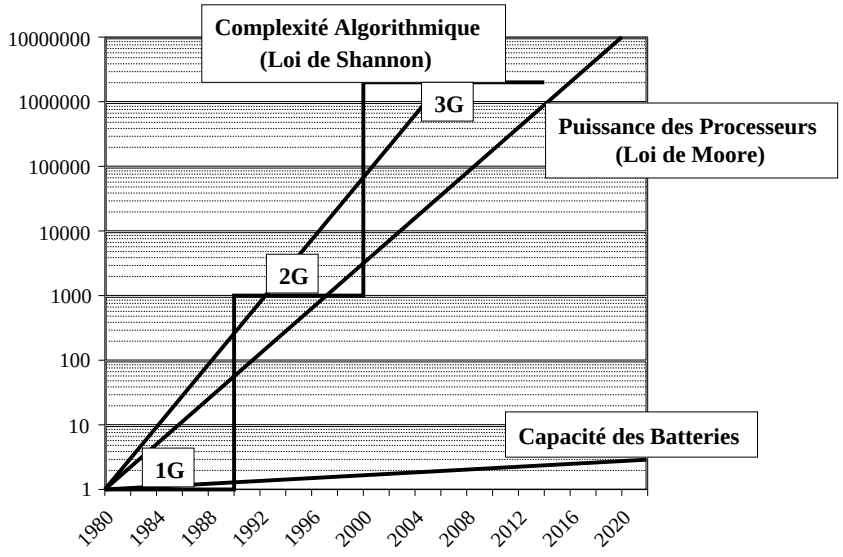

FIG. 1.2 L'évolution de la technologie, source R. Subramanian (Morphics Tech Inc.)

L'arrivée des technologies fortement submicroniques a provoqué aussi un fort intérêt pour la consommation, puisque la densité de chaleur générée par mm^2 et le risque d'endommager le circuit dans les périodes de plus forte activité deviennent non négligeables. Des systèmes de refroidissement doivent être ajoutés augmentant considérablement le prix du dispositif. Pour éviter cela, des mesures de réduction de la consommation adéquates doivent être prises. La figure 1.3 nous montre la densité de chaleur générée dans les processeurs actuels en fonction de leur technologie et la tendance prévisible si l'on continue à ce rythme de miniaturisation de la technologie.

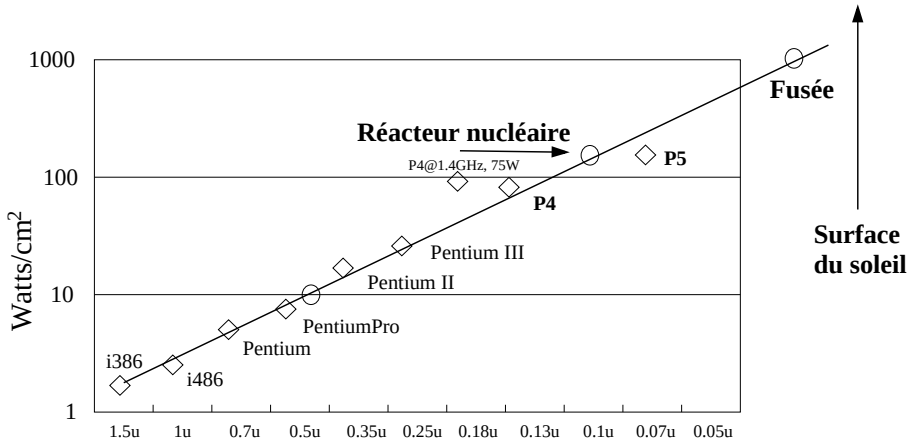

FIG. 1.3 L'évolution de la densité de chaleur dans les processeurs, source F. Pollack (Intel Inc.)

Une dernière raison est d'ordre écologique. La consommation d'énergie dans le monde due aux appareils électroniques devient de plus en plus importante. 20% de la consommation d'énergie à Amsterdam est utilisée par les systèmes de télécommunications. Aux Etats Unis, 9% de la consommation électrique nationale provient de l'utilisation d'internet, 13% si toutes les applications d'ordinateur sont comptées [PSZ04]. Le transfert de 4 MBytes de données sur le web consomme l'énergie générée par un kilogramme de charbon. En conséquence, la réduction de la consommation électrique des appareils électroniques aidera à réduire la consommation globale d'énergie sur la planète et les risques associés à sa production. Pour toutes ces raisons, un fort intérêt est né chez les industriels et les universitaires pour l'étude des méthodes et des techniques d'estimation et de réduction de la consommation dans les circuits intégrés.

Le but de ce travail est donc de proposer une méthode d'estimation de la consommation des systèmes embarqués, permettant l'évaluation des techniques de réduction. Pour cela notre travail a commencé par l'étude de la problématique associée à ce type d'analyse, cherchant à situer dans le flot de conception le niveau dans lequel ces études ont le plus grand impact sur le circuit. Le chapitre 2 fixe donc le contexte de notre travail.

Tout d'abord, nous réalisons une description du type de système intégré visé et du flot de conception typique utilisé dans ces systèmes. Puis, nous parlons de l'origine de la consommation électrique des circuits CMOS et des techniques de réduction existantes. Nous énumérons les outils d'optimisation et d'estimation dans le flot de conception jusqu'au niveau architectural, niveau dans lequel nous situons notre travail. Ensuite, nous décrivons les caractéristiques demandées à un outil d'estimation à ce niveau et les difficultés soulevées. Pour les résoudre, nous proposons la simulation architecturale au niveau cycle-précis et nous énumérons les problèmes posés lors de la modélisation en se basant sur ce type de simulation, questions auxquelles nous tentons de répondre dans le reste du manuscrit, en décrivant notre approche.

Le chapitre 3 présente l'état de l'art sur les différentes approches d'estimation et d'optimisation de la consommation existant aux niveaux algorithmique, système et architectural et en particulier au niveau architecture de système cycle-précis. Pour chacun d'entre eux, nous expliquons la méthode suivie, les modèles et les paramètres de la consommation, les techniques de réduction de la consommation appliquées, la façon de valider la méthode, les résultats obtenus ainsi que les avantages et les

inconvénients. Finalement, nous situons notre approche d'estimation au niveau cycle par rapport à celles déjà existantes.

Le chapitre 4 développe la méthode de modélisation et d'estimation de la consommation grâce à la simulation au niveau cycle-précis. Les avantages de ce type de modélisation sont expliqués dans la première partie du chapitre avec le flot d'estimation proposé. Après, nous analysons la façon de modéliser fonctionnellement un système embarqué au niveau cycle et comment obtenir la consommation en utilisant notre méthode générale d'estimation de la consommation. La façon de calculer les valeurs de consommation est expliquée à la fin du chapitre.

Le chapitre 5 explique en détail les modèles de consommation de chaque type de composant. Nous avons choisi de modéliser les plus utilisés dans un système embarqué typique : processeur, mémoires, accélérateurs matériels et interconnexions. Les modèles représentent les valeurs d'énergie par opération réalisée à chaque cycle et par composant selon différents modes de fonctionnement.

Le chapitre 6 présente TSS, l'outil utilisé pour la simulation fonctionnelle au niveau cycle d'un système embarqué. Nous présentons l'application de notre approche d'estimation dans cet environnement de simulation de systèmes, qui modélise les composants en langage C/C++. Nous montrons aussi comment modéliser et simuler la consommation au niveau cycle en utilisant SystemC. Nous illustrons les variables, les paramètres et les commandes à implanter pour rendre possible l'estimation de la consommation avec ces deux types de simulations.

Dans le chapitre 7, nous montrons l'application de notre approche sur un exemple de système embarqué très approprié pour des études d'estimation et de réduction de la consommation : un circuit décodeur vidéo MPEG4 destiné à la téléphonie mobile. Nous présentons d'abord la norme MPEG4 et le système matériel-logiciel choisi pour l'implantation du décodeur vidéo. Après, nous décrivons les modèles d'énergie de chaque composant du système, avec les valeurs retenues des paramètres et des variables d'énergie. Les évaluations de l'énergie du système et l'estimation de l'erreur des résultats sont ensuite montrées. Finalement, plusieurs techniques de réduction de la consommation sont illustrées et analysées de façon à trouver celles dont le gain en énergie est le meilleur.

Finalement, le chapitre 8 résume les résultats de cette thèse et les conclusions et propose la prolongation de nos travaux dans les domaines de l'estimation et l'optimisation de la consommation des systèmes embarqués complexes décrits à très haut niveau.

Chapitre 2

Problématique

Sommaire

Ce chapitre traite des difficultés rencontrées lors de l'estimation et l'optimisation de la consommation électrique des systèmes intégrés au niveau architecture de système.

L'estimation et l'optimisation de la consommation doit s'appliquer à tous les niveaux du flot de conception. Cependant, les gains les plus importants sont obtenus au niveau de la conception architecturale de système car dans ce stade nous allons définir le type et l'utilisation des composants dans le système. Pourtant, l'estimation de la consommation de haut niveau reste un problème très complexe car il faut posséder des outils capables de manipuler des informations de consommation avant de disposer d'une implantation physique du circuit. Pour cette raison, les outils existants évaluent souvent la consommation dans les étapes les plus avancées de la conception, quand les informations sur l'implantation sont plus précises, par exemple au niveau portes ou transistors. Les optimisations en consommation au niveau architecture de système restent fortement dépendantes de l'expérience des concepteurs et, sans outil adéquat, il est particulièrement difficile d'optimiser un système.

L'objectif de cette thèse est de proposer une méthode et un outil d'évaluation de la consommation au niveau architecture de système cycle-précis. L'estimation de la consommation de haut niveau cherche à fournir des valeurs suffisamment précises pour permettre l'exploration de l'architecture d'un système matérielle et l'évaluation de l'impact des techniques de réduction de la consommation existantes au niveau système. La démarche s'est basée sur un exemple réel de système embarqué, dans le cadre d'une application industrielle conçue pour le marché portable. Toutefois l'approche proposée reste suffisamment générale pour être utilisée dans une large gamme d'applications embarquées.

Dans la première partie de ce chapitre, nous allons décrire le type de système intégré visé et l'application industrielle utilisée comme exemple. Ensuite, nous montrerons le flot de conception typique pour la création de ce type de systèmes. Dans une deuxième partie, nous rappellerons l'origine de la consommation électrique des circuits CMOS et les techniques de réduction existantes. Finalement, dans la troisième partie, nous énumérerons les outils d'optimisation et d'estimation de la consommation existant dans le flot de conception jusqu'au niveau architecture de système, dans lequel il manque des outils pour le type de systèmes intégrés visés. Les caractéristiques demandées à un outil d'estimation à ce niveau et les difficultés soulevées seront analysées. Pour les résoudre, nous proposerons la simulation architecturale de système au niveau cycle-précis, ce qui permettra de disposer d'un certain nombre d'informations qui vont augmenter la précision de l'estimation que nous énoncerons. Enfin nous énumérerons les problèmes posés lors de la modélisation en consommation des systèmes intégrés, en se basant sur ce type de simulation, questions auxquelles nous tenterons de répondre dans cette thèse en décrivant notre approche.

2.1 Les systèmes intégrés actuels et leur flot de conception

2.1.1 Le système embarqué typique

Un système embarqué est une combinaison de composants de traitement numérique conçu pour exécuter une application particulière connue à l'avance. Il peut être intégré sur une même puce de silicium en s'appelant dans ce cas système sur puce ou *System-on-Chip* (SoC).

Un système embarqué peut être très hétérogène et contenir plusieurs composants matériels autour

d'un ou plusieurs bus ou réseaux d'interconnexions. L'architecture d'un système embarqué est par conséquent hybride : certaines fonctions sont réalisées par un logiciel s'exécutant sur un microprocesseur, d'autres étant réalisées par des coprocesseurs spécialisés ou accélérateurs implantés en matériel. Dans ce cas, la partie logicielle sert aussi à assurer le pilotage de la partie matérielle. Pour stocker le logiciel et gérer un ensemble de données permanentes et volatiles, le système dispose de mémoires. Les interconnexions entre les différents composants matériels du système sont assurées par un ou plusieurs bus ou réseaux d'interconnexions.

Les composants d'un système embarqué typique, illustrés dans la figure 2.1, sont les suivants :

- ⇒ Processeur général CPU, avec/sans mémoire cache, RISC, CISC.
- ⇒ Processeur spécifique : Digital Signal Processor DSP,...etc.
- ⇒ Processeur reconfigurable : FPGA, ASIP (Application Specific Instruction-set Processor).
- ⇒ Mémoire d'instructions non volatile : en lecture seule ROM ou reprogrammable FLASH.
- ⇒ Mémoire de données vive : à accès aléatoire statique SRAM, dynamique DRAM ou synchrone dynamique SDRAM.
- ⇒ Accélérateurs matériels spécifiques ASICs.
- ⇒ Composants analogiques.
- ⇒ Réseaux d'interconnexions, bus, contrôleur de bus BCU.
- ⇒ Périphériques d'entrée/sortie I/O, contrôleurs de mémoires, interfaces,...etc.

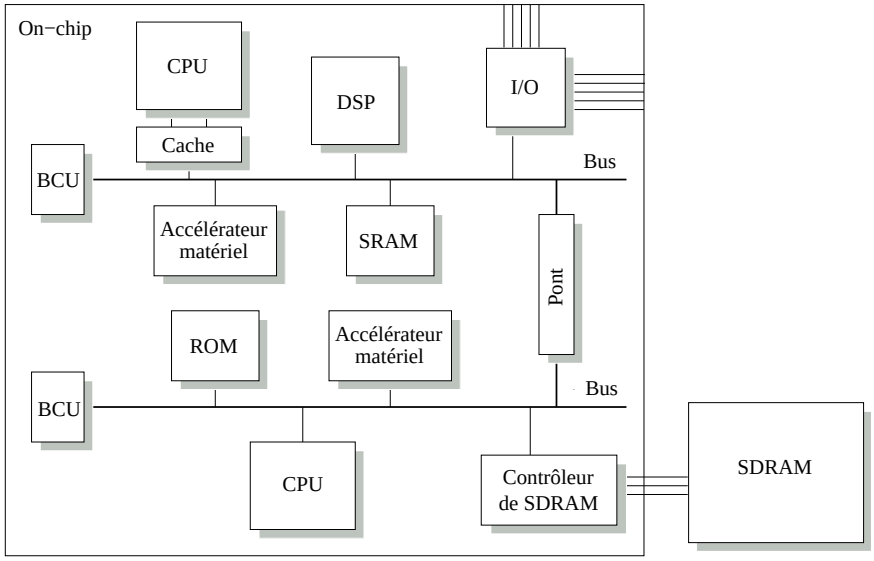

FIG. 2.1 Système embarqué typique

Les systèmes embarqués sont utilisés dans des applications multiples et très variées : machines à laver, cafetières, déclenchement d'airbags, téléphones sans fil, caméras numériques, etc. L'application industrielle visée dans le cadre de cette thèse est un exemple de circuit de décodage vidéo MPEG4 (Motion Picture Expert Group standard 4) [MPE] en cours de développement chez Philips.

Il s'agit d'un exemple typique de système embarqué destiné à être intégré dans un terminal portable (téléphone mobile). Particulièrement adapté pour des applications multimédia, le niveau de performances couplé à la variabilité des contenus et débits à traiter dans un terminal MPEG4 demande une gestion dynamique des ressources matérielles du système. Cette contrainte impose d'une part

l'exploration des différentes architectures de système possibles, d'autre part une gestion adéquate des composants choisis en vue du bon fonctionnement du circuit et de l'optimisation de la consommation.

2.1.2 Le flot de conception des systèmes intégrés

Un flot de conception typique utilisé dans les systèmes embarqués est illustré de façon simplifiée sur la figure 2.2. L'entrée du flot est la spécification du système. Elle consiste en une spécification fonctionnelle et une spécification non-fonctionnelle. La spécification fonctionnelle est l'algorithme qui décrit l'application, écrit dans un langage de programmation séquentiel standard comme le C ou le C++. La spécification non-fonctionnelle est l'ensemble des contraintes que le système devra supporter comme la vitesse, la surface de silicium ou la consommation.

La spécification fonctionnelle est partitionnée ensuite en tâches qui seront exécutées en parallèle, permettant ainsi l'accélération du traitement pour atteindre les contraintes imposées en vitesse ou en débit de données. Pour ce faire, le système est modélisé comme un ensemble des tâches communicantes que nous appelons description parallèle du système. Cette description peut être faite dans différents langages ; C/C++, YAPI [dKa00], Esterel [Ber91], Lustre [RH92], SpecC [DGG02], VHDL, etc.

Le choix de l'architecture du système cible intervient alors et consiste à déterminer le nombre et le type de processeurs que le système doit contenir, ainsi que l'organisation et les moyens de communications. L'affectation des tâches dans l'architecture de système se fait en analysant et définissant pour chacune d'entre elles, laquelle sera réalisée en matériel ou en logiciel et par quel processeur ou coprocesseur. Pour le développement de l'architecture de système et des interfaces entre blocs matériels, il s'avère nécessaire d'effectuer dans cette étape une simulation dans un simulateur au niveau architecture de système. En effet, les performances d'un circuit à l'intérieur d'un système intégré sont difficilement évaluables autrement que par la simulation. En conséquence, il faut construire un modèle au niveau matériel pour pouvoir évaluer les performances du circuit dans son contexte. Le simulateur utilisera les modèles fonctionnels des composants matériels du système et sera capable de simuler les communications entre modèles. Le niveau d'abstraction du simulateur doit être suffisamment élevé pour permettre la simulation rapide de millions de cycles, ce qui est très important dans le domaine de la vidéo. Cette simulation va permettre aussi le développement de la composante logicielle et son intégration dans le système.

Pour avoir un niveau de précision suffisant sur la performance, il est usuel d'utiliser un simulateur cycle-précis [The98] [Hom01]. Les modèles des composants génériques comme les processeurs et les mémoires sont disponibles dans des bibliothèques. Les modèles des coprocesseurs en développement sont crées à partir des spécifications fonctionnelles partitionnées en tâches. Le concepteur doit pouvoir tenter plusieurs architectures matérielles de son système afin de pouvoir choisir le meilleur compromis entre les performances et le coût du circuit.

Une fois l'exploration de l'espace de conception effectuée, les décisions d'implantation peuvent être prises et la conception du matériel commence. Chaque bloc matériel sera décrit séparément en VHDL ou Verilog au niveau RTL. Puis il sera traité par les outils de synthèse, fournissant la description d'un réseau de portes logiques (*netlist*). Toutes ces descriptions serons testées et vérifiées jusqu'à ce que chaque bloc accomplisse toutes les contraintes fonctionnelles et non-fonctionnelles. Une fois tous les blocs créés et testés, ils seront intégrés dans le système avec le reste des composants et les commu-

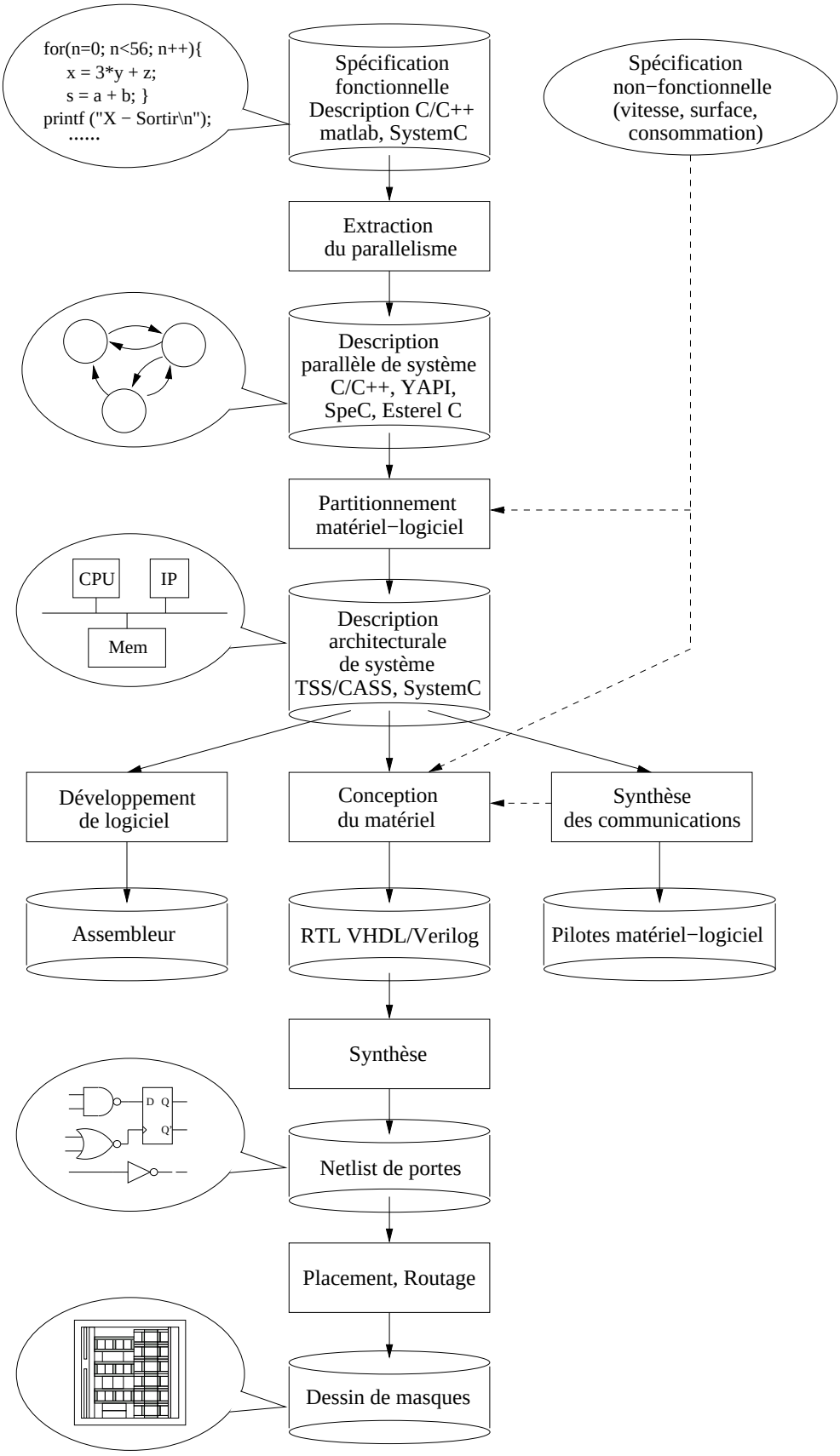

FIG. 2.2 Flot de conception des systèmes intégrés

nications : les processeurs, les mémoires, les interconnexions, etc. et l'ensemble sera synthétisé au niveau transistor, placé et routé sur une même puce en donnant le dessin de masques (*layout*) qui sera ensuite envoyé à la fonderie pour fabriquer le circuit.

2.2 La consommation des circuits CMOS

La technologie CMOS à logique complémentaire est la technologie la plus utilisée pour la fabrication des circuits intégrés numériques. Une des raisons principales de ce choix vient du fait que cette technologie permet une grande réduction de la consommation électrique des circuits. Contrairement aux technologies précédentes (par exemple NMOS), dans un circuit CMOS la majeure partie de la consommation a lieu pendant les transitions des noeuds et pendant l'état de repos, la consommation est pratiquement nulle. Malheureusement, cela n'est plus vrai pour les technologies fortement submicroniques où les courants de fuite deviennent critiques, surtout quand le circuit passe beaucoup de temps en mode repos.

En général, la consommation des circuits CMOS est divisée en deux parties distinctes : la consommation statique et la consommation dynamique [Pig04].

$$P = P_{statique} + P_{dynamique} \tag{2.1}$$

2.2.1 La consommation statique

La consommation *statique* se produit principalement dans l'état de repos du circuit. Elle est due aux courants sous le seuil et aux courants de fuites des transistors. Idéalement, elle devrait être nulle parce que dans l'état de repos d'un circuit CMOS, il n'y a pas de chemin entre l'alimentation et la masse. Cependant, un transistor CMOS n'est pas un interrupteur parfait et il y a toujours des pertes qui se produisent.

Dans les nouvelles technologies submicroniques utilisant des tensions de seuil très basses, trois types principaux de courants de fuites sont identifiés dans le transistor (illustrés dans la figure 2.3) [RMMM03] [Tur00] :

- le courant sous le seuil, I_{seuil} (subthreshold current),
- le courant de polarisation de diode en inverse, I_{diode} (reverse biased pn junction current),
- et le courant à travers la grille, I_{grille} (gate leakage current).

Il existe encore d'autres courants de fuites, comme le courant de drain induit par la grille (gate induces drain leakage GIDL) et le courant de perçage (drain source punch through current) [Tur00]. Enfin, d'autres courants proviennent de certains effets électriques submicroniques découverts au fur et à mesure de l'apparition de nouvelles technologies aux dimensions encore plus réduites, mais leur influence est pour le moment moins importante.

À température constante, la puissance statique d'un transistor dépend donc principalement de ces trois courants I_{seuil}, I_{diode} et I_{grille} et de la tension d'alimentation V_{dd} selon l'expression 2.2 [RMMM03].

$$P_{statique} = I_{fuites} \times V_{dd} = (I_{seuil} + I_{diode} + I_{grille}) \times V_{dd} \tag{2.2}$$

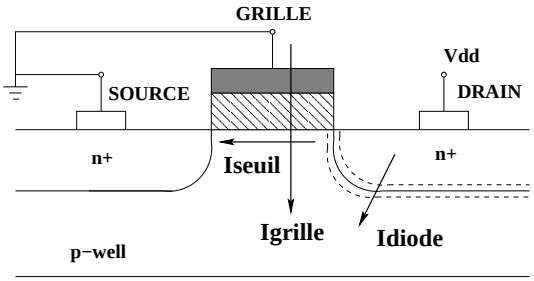

FIG. 2.3 Courants de fuites

Le courant sous le seuil

Le **courant sous le seuil**, I_{seuil} est un courant qui circule entre le drain et la source du transistor quand il y une différence de potentiel drain-source et le transistor se trouve en état OFF, c'est à dire, quand la tension grille-source V_{gs} est inférieure à la tension de seuil V_{th} [Tur00]. En ce moment, le transistor CMOS se comporte comme un transistor bipolaire laissant passer un courant qui dépend exponentiellement de la tension V_{gs}.

Ce courant est le composant le plus important de la consommation statique [RMMM03] et sa valeur peut être estimée pour un transistor à partir de l'expression 2.3 [Pig04].

$$I_{seuil} = I_s \times e^{\frac{-V_{th}}{nU_t}} \tag{2.3}$$

où I_s est le courant de fuites à $V_{gs} = V_{th}$, n un paramètre technologique et U_t la tension thermique.

Ce courant dépend donc de la tension de seuil V_{th} ainsi que de la température, des dimensions physiques du canal, de la surface de dopage, de la profondeur de jonction drain-source et de l'épaisseur de l'oxyde de la grille. En conséquence, il augmente considérablement dans les technologies fortement submicroniques dû à la diminution de la taille du transistor et surtout de la tension de seuil V_{th}.

Le courant de fuites des jonctions

Le **courant de fuites des jonctions**, I_{diode} (ou courant d'injection dans le substrat) correspond aux fuites dans les diodes des jonctions PN (illustré dans la figure 2.3). Les jonctions source-substrat (SB) et drain-substrat (DB) sont des diodes à polarisation nulle ou inverse. Cela provoque d'une part et d'autre de l'interface N-P, une zone déserte qui les isole du reste du substrat. Cependant, un courant de polarisation inverse, normalement très faible mais non nul, circule au travers de ces jonctions à tout moment, quel que soit l'état du transistor (ON ou OFF) [RMMM03]. Ce courant est indépendant de la tension de l'alimentation. En revanche, il dépend fortement des géométries de transistor [Tur00]. Cette consommation peut être non négligeable dans certaines applications, comme le point mémoire d'une DRAM et surtout dans les technologies fortement submicroniques, du fait de la réduction des dimensions de transistor.

Le courant à travers la grille

La réduction des dimensions du transistor et donc de l'épaisseur de l'oxyde de la grille, provoque aussi la rupture de l'impédance infinie entre la grille et le substrat. Dans ce cas, la différence de potentiel grille-substrat donne lieu à un effet tunnel avec un transfert d'électrons. Un nouveau courant de fuite apparaît entre la grille et le substrat, le **courant à travers la grille**, I_{grille}, qui circule toujours quand il y a une tension grille-substrat. Celle-ci peut être positive ou négative et se produire à tout moment quel que soit l'état du transistor (ON ou OFF) [RMMM03]. Ce courant augmente beaucoup dans les nouvelles technologies fortement submicroniques et devient l'une des composantes majeures de la consommation statique avec le courant sous le seuil [RMMM03].

Tous ces courants étaient faibles pour les technologies en dessus de 0.25 μm, ce qui fait que la consommation statique pouvait être négligée dans les circuits conçus pour ces technologies. Par contre, dans les nouvelles technologies submicroniques ces courants sont très importants et il devient nécessaire de les prendre en compte. Par exemple, pour un processeur Itanium d'Intel fondu dans la technologie 0.13 μm, cette consommation arrive à 14% de la consommation totale du circuit [Int03].

La figure 2.4 donne les prévisions de tensions d'alimentation, de seuil et de densité du courant de fuite par cm^2 (roadmap 2003 de l'ITRS, *International Technology Roadmap for Semiconductors*) [Roa03]. Selon cette source, à partir de l'année 2006 l'augmentation prévue des valeurs de courant de fuite sera tellement considérable, que les solutions de fabrication connues et prévues jusqu'à alors ne seront pas suffisantes pour s'en accommoder. Des nouvelles technologies devront être mises en place pour faire face à ce problème.

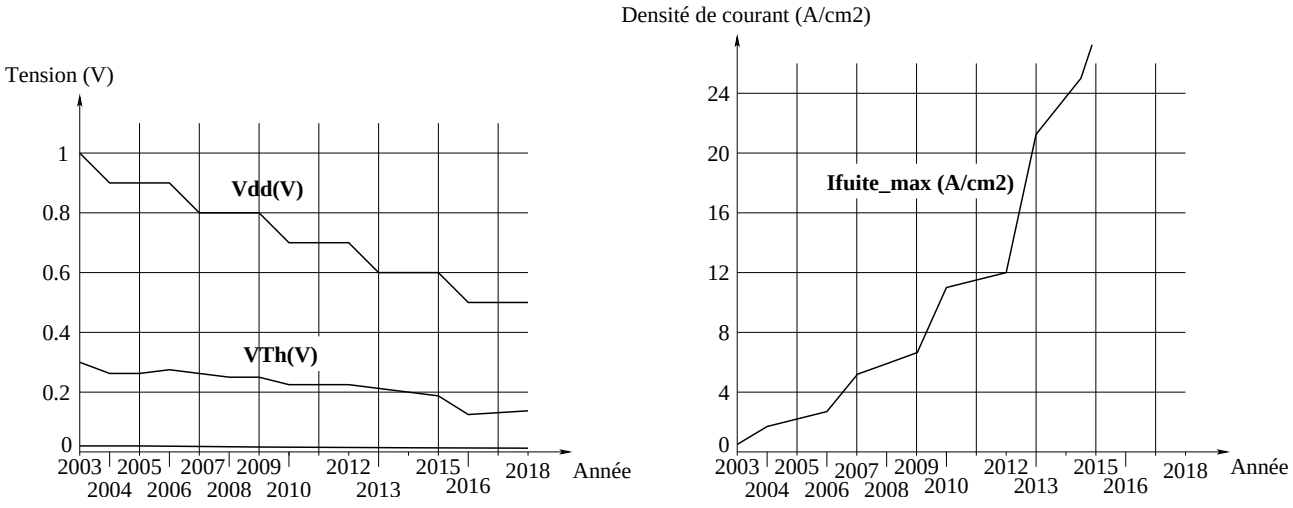

FIG. 2.4 Tensions et courant de fuite prévues dans la roadmap 2003 ITRS

2.2.2 La consommation dynamique

La consommation *dynamique* a lieu à chaque transition d'un noeud logique dans un circuit CMOS. Elle se divise en deux composantes : la consommation due aux courants de court-circuit et celle due aux courants de charge [Pig04].

$$P_{dynamique} = P_{court-circuit} + P_{charge} \tag{2.4}$$

La consommation de court-circuit

Lors de la commutation d'une porte CMOS, il arrive un moment où les transistors des réseaux PMOS et NMOS sont passants en même temps, ce qui crée un *courant de court-circuit* entre l'alimentation et la masse. La variation du courant de court-circuit dans le temps est illustrée graphiquement pour un inverseur dans la figure 2.5.

Lorsque le signal d'entrée est inférieur à V_{Tn} ou bien supérieur à V_{Tp} le courant de court-circuit est nul. Il augmente lorsque la tension d'entrée dépasse V_{Tn} et diminue au fur et à mesure que la tension d'entrée se rapproche de V_{Tp}. La puissance de court-circuit P_{cc} d'une porte est calculée à partir de l'expression 2.5 [Pig04].

$$P_{cc} = f \times I_{cc} \times V_{dd} \tag{2.5}$$

où I_{cc} est le courant moyenne de court-circuit déchargé à la masse lors d'une transition, f la fréquence des transitions et V_{dd} la tension d'alimentation.

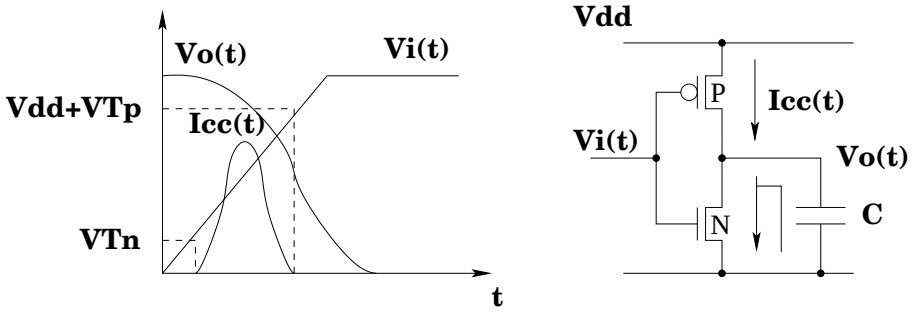

FIG. 2.5 Courant de court-circuit d'un inverseur

Son amplitude dépend du temps passé en court-circuit et cela dépend de l'équilibre entre le front du signal d'entrée et celui du signal de sortie, ce qui dépend de la capacité de sortie [Kel97]. D'une manière générale, le courant de court-circuit est significatif dans un circuit où le front du signal d'entrée est beaucoup plus long que celui du signal de sortie ou inversement. Pour que la consommation de court-circuit soit négligeable, il est important d'avoir des fronts rapides et bien équilibrés.

Les études de Nose [NS00] nous montrent que la consommation de court-circuit peut augmenter à mesure que la technologie évolue vers le submicronique profond si le rapport entre les tensions V_{th}/V_{dd} diminue. Cela arrive dans les circuits fonctionnant à basse tension V_{dd} et très basse tension de seuil V_{th} pour augmenter la vitesse de commutation. Dans ce cas, les courants de court-circuit augmentent

beaucoup et deviennent un composant important, supérieure à 20%, de la consommation totale du circuit. Selon les études de Vemuru [VS94], dans une technologie 0.17 μm, cette consommation peut atteindre un ordre de 12% à 60% de la consommation dynamique totale.

Par contre, si l'on maintien constant la proportion V_{th}/V_{dd}, le rapport entre la consommation de court-circuit et celle dynamique restera aussi stable. Et même, si le rapport V_{th}/V_{dd} augmente (haute V_{th}) cette consommation diminuera.

En conclusion, pour les circuits correctement conçus et utilisant des valeurs de tension V_{dd} et V_{th} bien calibrées, la valeur de la consommation de court-circuit sera normalement inférieure à 10% de la consommation dynamique totale même dans les technologies fortement submicroniques. Cependant, toute technique d'estimation et d'optimisation de la consommation CMOS devrait tenir compte de cette consommation pour éviter de commettre des erreurs.

La consommation de charge dynamique

La consommation *de charge dynamique* d'un circuit CMOS représente l'énergie nécessaire pour charger la capacité C de sortie de la porte à la tension d'alimentation, puis pour la décharger à la masse. Cette partie est la composante la plus importante de la consommation. Elle représente en moyenne plus de 90% de la consommation totale du circuit et c'est elle que l'on dénomme souvent puissance dynamique [Pig04].

La figure 2.6 montre les phases de charge et décharge d'un inverseur et les courants associés. On peut le modéliser pour la plupart des portes. Les courants $i_p(t)$ et $i_n(t)$ sont les courants nécessaires pour charger et décharger la capacité physique du noeud de sortie de la porte C à la tension d'alimentation V_{dd}. Le courant $i(t)$ représente le courant sur la capacité C. Pendant la phase de charge, la puissance fournie par l'alimentation correspond à $C \times {V_{dd}}^2$. La moitié est stockée dans la capacité C et l'autre moitié est dissipée sous forme de chaleur par le courant de charge $i_p(t)$ transitant au travers du transistor PMOS. Pendant la phase de décharge, le transit de courant $i_n(t)$ à travers le transistor NMOS provoque la dissipation sous forme de chaleur de l'énergie accumulée dans la capacité C.

De cette façon et suivant les calculs mathématiques illustrés dans [Pig04], on obtient pour un système synchrone de plusieurs millions de portes, l'énergie moyenne dissipée par cycle suivante :

$$E_{dynamique} = a \times N \times C \times {V_{dd}}^2 \tag{2.6}$$

où a est l'activité du circuit (pourcentage de portes qui commutent entre toutes les portes du circuit), N le nombre total de portes, C la capacité moyenne de sortie par porte et V_{dd} la tension d'alimentation.

La puissance correspond à l'énergie moyenne consommée par seconde à une fréquence de travail f. Cela nous permet d'obtenir la **formule générale de la dissipation de puissance dynamique** suivante :

$$P_{dynamique} = f \times a \times N \times C \times {V_{dd}}^2 \tag{2.7}$$

La relation entre tous les composants de la consommation pour une technologie supérieure à 0.35 μm, est illustrée graphiquement sur la figure 2.7 [Bou99] [Sak03]. Elle nous montre les pourcentages de consommation statique et dynamique, de charge et de court-circuit. Pour la consommation de charge,

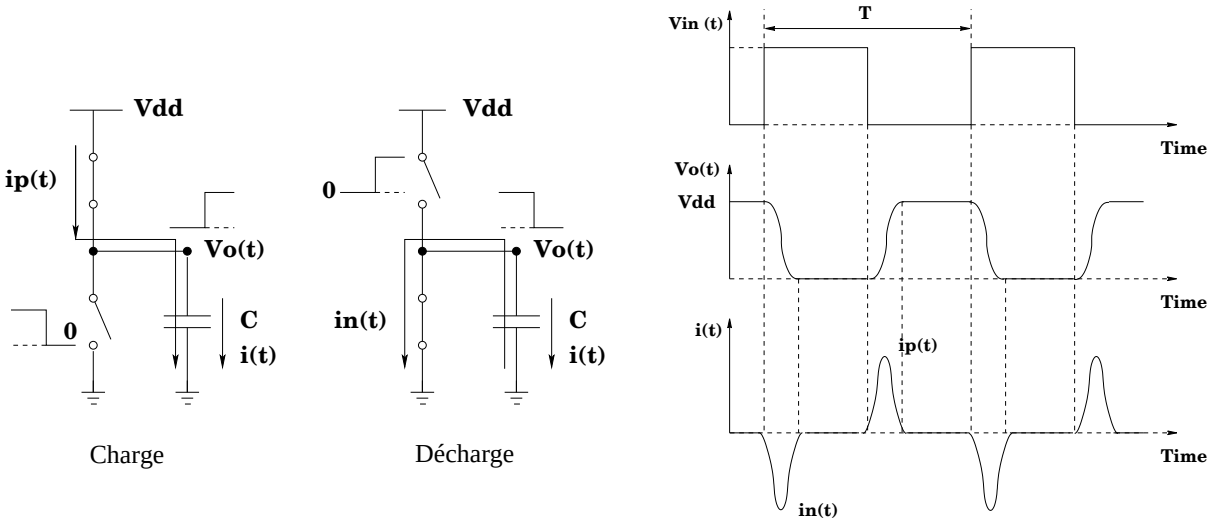

FIG. 2.6 Circuit équivalent d'un inverseur et les courants de charge et décharge

on peut observer la relation entre les transitions utiles et inutiles (*glitches*). Les transitions utiles correspondent au fonctionnement des unités logiques, des mémoires, de l'horloge et des éléments d'interconnexion, pour un circuit décodeur MPEG2 fondu à une technologie 0.18 μm. Ses répartitions sont sujettes aux changements de technologie ou d'application.

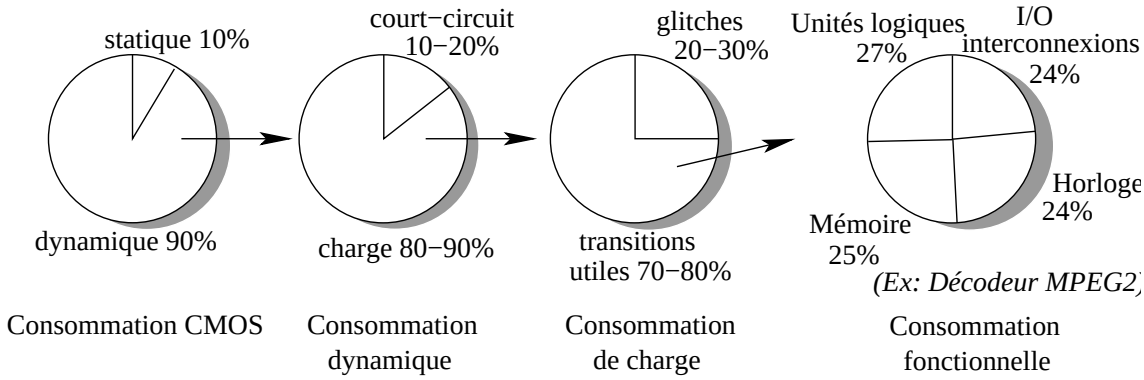

FIG. 2.7 Pourcentage de consommation pour la technologie CMOS

2.2.3 Les techniques de réduction de la consommation

Les techniques de réduction de la consommation dans les circuits CMOS concernent principalement la diminution de la composante dynamique, étant donné que la consommation de celle-ci est la plus importante. La consommation statique possède aussi ses techniques spécialisées.

La réduction de la composante dynamique est basée sur la diminution des valeurs des paramètres qui entrent dans la formule générale de la dissipation de la puissance dynamique (équation 2.7). Cette

équation nous montre que la consommation dynamique est proportionnelle à la fréquence, à l'activité du circuit, à la capacité et au carré de la tension d'alimentation. Si l'on réduit l'un de ces paramètres, on réduit la consommation dynamique. Le paramètre le plus important est la tension d'alimentation. La dépendance étant quadratique, sa réduction va diminuer de façon drastique la consommation dynamique. Étant donné qu'il s'agit d'un paramètre de l'équation 2.2, la consommation statique se trouvera aussi réduite.

La réduction des paramètres de l'équation 2.7 peut s'envisager à tous les niveaux de la conception : niveaux système, architecture de système, RTL, logique et physique. Une multitude de techniques qui jouent sur tous ces paramètres ont été présentées dans des articles scientifiques depuis une dizaine d'années [Ath99] [Pig04] [CB95]. Cependant seulement quelques unes sont arrivées à être acceptées dans le flot de conception actuel et à faire partie des outils de conception commercialisés [Bul] [Synb]. Étant donné que la consommation est un phénomène physique, beaucoup d'entre elles concernent les niveaux les plus concrets de la conception : niveaux physique/layout. Elles jouent sur les caractéristiques constructives des transistors comme la taille, les capacités parasites et la tension d'alimentation et de seuil [Pig04] [CB95]. Cependant, on observe que les meilleurs taux de réduction de la consommation sont obtenus aux niveaux les plus élevés : niveaux algorithmique, système et architecture de système. Des choix sur l'architecture de système qui vont influencer énormément la consommation sont faits à ce moment de la conception puisqu'on va décider sur le type et le taux d'utilisation des composants matériels : partitionnement en processeurs généraux et/ou coprocesseurs spécifiques, hiérarchie des mémoires, type des interconnexions, etc.

Chaque type de composant a un comportement différent du point de vue de la consommation. Cela dépend de sa fonctionnalité et sa capacité à être programmé. L'élection d'un processeur général ou d'un DSP spécialisé pour exécuter une tâche spécifique, va avoir un grand impact sur l'énergie nécessaire, comparé à l'exécution sur un composant reprogrammable du type FPGA ou sur accélérateur matériel. L'application du décodage MPEG2 par exemple, consomme 25 Watt en s'exécutant sur un processeur, 4 Watt sur un DSP et 0.4 Watt sur un SoC [Sak03]. Quelques exemples des taux d'efficacité par type de composant sont illustrés dans la figure 2.8.

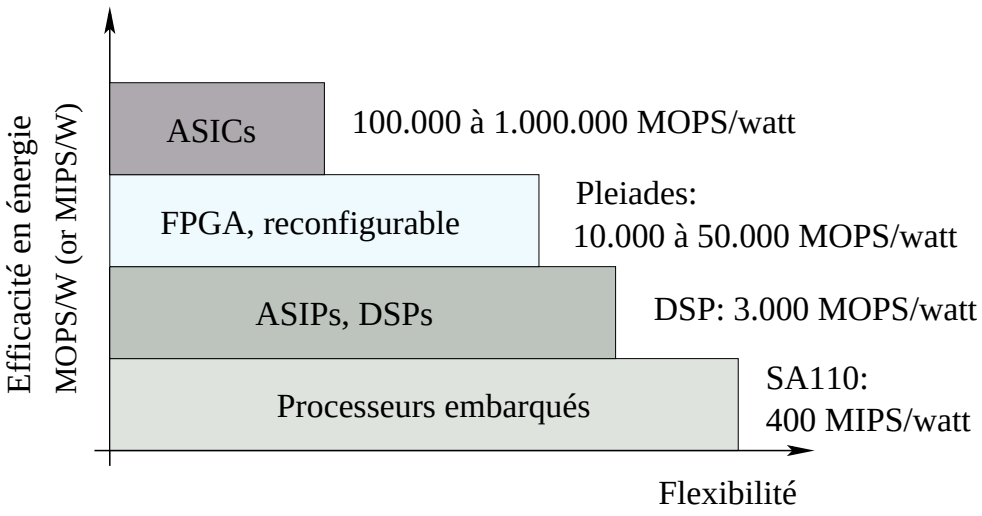

FIG. 2.8 Flexibilité vs performances

Une fois le partitionnement fait, d'autres techniques plus élaborées peuvent être appliquées pour permettre le contrôle de l'activité de commutation des composants, en adaptant leur fréquence et tension d'alimentation à la charge de travail [Usa98] ou en coupant les horloges dans les périodes inactives [ABR01]. La tension d'alimentation est un paramètre physique qui dépend de la technologie utilisée. Mais pour une même technologie, sa valeur peut être réduite jusqu'à une certaine limite [PBB00]. Plus on réduit la tension, plus le transistor va commuter lentement. En conséquence, pour maintenir la même fonctionnalité il faut réduire simultanément la tension et la fréquence de fonctionnement. On peut, dynamiquement et en fonction de la charge instantanée demandée au circuit, adapter au niveau architecture de système ces deux paramètres avec pour optimiser globalement la puissance consommée [PLS00] [ABR01] [PBB00].

D'autres techniques existent aussi au niveau algorithmique qui permettent des grandes réductions de la consommation. Elles cherchent l'optimisation du code logiciel grâce à la réduction du nombre d'instructions et d'accès mémoire. Cela consiste à nettoyer le code (en éliminant les parties non nécessaires), à séparer les codes de contrôle, les boucles et les opérations arithmétiques et à optimiser les boucles en les déroulant ou en les fusionnant [BFSS02] [Ca00] [BE95].

Dans la figure 2.9 quelques techniques sont montrées graphiquement par niveau avec une estimation qualitative des gains potentiels. Plus l'approche est de haut niveau, plus les degrés de liberté sont importants, ce qui explique que les gains potentiels soient plus élevés.

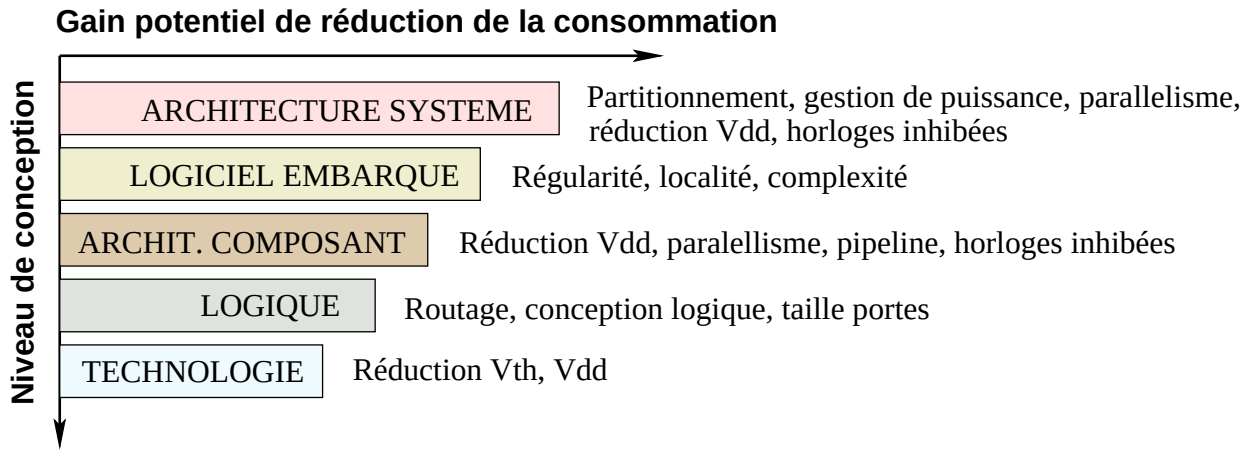

FIG. 2.9 Techniques de réduction de la consommation par niveau de conception

2.3 L'étude de la consommation

2.3.1 L'estimation de la consommation dans le flot de conception

La réduction de la consommation nécessite des outils d'analyse capables d'évaluer l'impact de l'application de toutes ces techniques. Cela signifie que nous avons besoin d'outils d'estimation et d'optimisation de la consommation à tous les niveaux du flot de conception comme le montre graphiquement la figure 2.10.

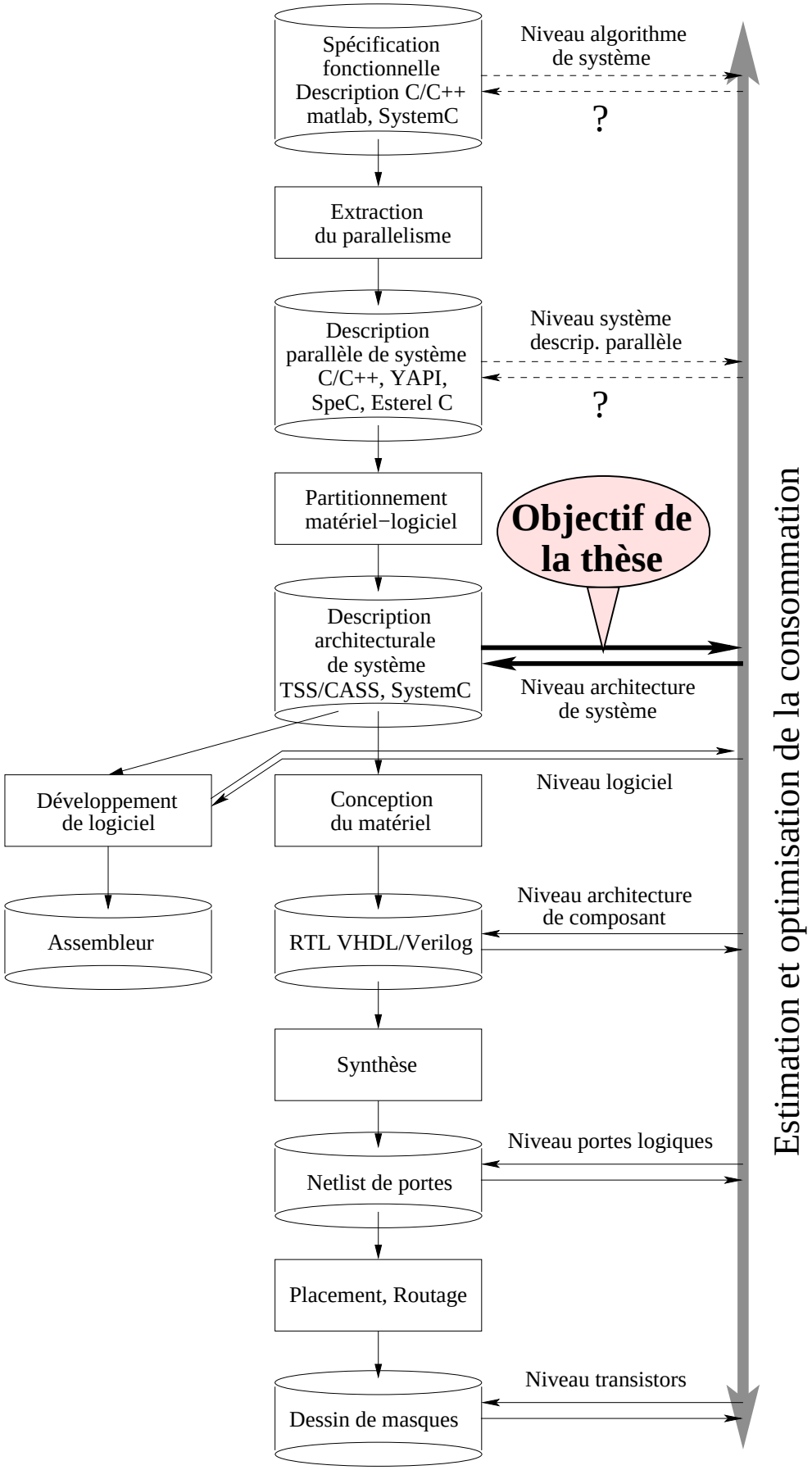

FIG. 2.10 Flot de conception avec estimation et optimisation de la consommation

Il existe beaucoup d'outils de CAO capables d'évaluer la consommation des circuits décrits à bas niveau comme SPICE au niveau physique [Uni], DIESEL au niveau portes [Phi01a] et Petrol au niveau RTL [PLG98]. Les précisions obtenues sont très bonnes car à ces niveaux on dispose des informations sur l'implantation physique, informations nécessaires à l'évaluation de la consommation. Ce sont les paramètres physiques de l'équation 2.7 comme le nombre de transistors ou leur activité.

Les évaluations de la consommation au niveau porte ou RTL synthétisable sont faites couramment dans l'industrie. Beaucoup emploient la simulation avec des vrais stimuli en entrée pour obtenir des vrais valeurs en activité et une bonne précision en consommation qui peut être à environ 10-15% d'erreur par rapport aux résultats obtenus sous SPICE [Phi01a] [PLG98] [SSN99]. La simulation d'un système complet à ces niveaux étant trop lente, on est obligé de faire des estimations composant par composant. En mesurant la consommation de cette façon, on ne connaît ni l'influence des interactions inter-blocs, ni l'évolution de la consommation pendant le fonctionnement normal de l'application dans le système. Si les évaluations ne donnent pas de résultats en consommation acceptables, on ne peut qu'optimiser en interne chaque composant car la conception est déjà trop avancée. Il sera trop tard pour changer l'architecture de système ou la gestion globale des composants. Pour ce faire, il faudra attendre la génération suivante du circuit.

Les estimations de la consommation au niveau algorithmique sur la composante logicielle du système s'exécutant sur un processeur général, sont faites depuis 1994 notamment par une méthode développée par Tiwari [TMW94]. Elle consiste basiquement à analyser le code en assembleur et à accumuler une consommation par instruction exécutée. Les valeurs de consommation par instruction ont été préalablement mesurés avec précision sur une implantation matérielle du processeur visé.

Aux niveaux plus élevés comme le niveau architecture de système ou description parallèle du système, il n'existe pas d'outil industriel d'estimation et d'optimisation de la consommation. Par contre, au vu des avantages que l'on peut en tirer, plusieurs méthodes d'estimation de la consommation de haut niveau ont vu le jour [HL02] [LRDL00] [GVH01] [LP01]. Quelques unes seront analysées plus en détail dans le chapitre suivant, l'*état de l'art*. Dans le cadre de cette thèse, nous nous sommes intéressés à l'estimation de la consommation au niveau architecture de système. Les raisons de ce choix, les caractéristiques spécifiques de ce type d'estimation et les problèmes posés seront analysés en détail dans les sections suivantes.

2.3.2 L'estimation de la consommation de haut niveau

L'estimation de la consommation aux niveaux les plus élevées de la conception est très avantageuse comme nous avons expliqué précédemment. Cependant, plus on monte dans le flot de conception plus il est difficile de faire des estimations avec une précision suffisamment raisonnable et par conséquent, moins d'outils d'estimation existent. Les problèmes viennent du fait que l'on ne connaît pas encore l'implantation physique du matériel. Il est trop tôt dans le flot de conception pour avoir les vrais valeurs des paramètres de l'équation 2.7 qui jouent sur la consommation comme la fréquence de travail, le nombre de transistors, son activité, la technologie dans laquelle le circuit sera implanté, etc. De plus, à ce niveau il est difficile de modéliser certains aspects des circuits qui vont influencer beaucoup la consommation comme les transitions inutiles (glitches), la consommation de l'arbre d'horloge et des interconnections.

Un autre problème vient de l'hétérogénéité de ce type de système. Un SoC est un ensemble de composants très différents, que ce soit par leur implantation ou par leur comportement en consommation. Il est généralement constitué d'une partie logicielle et d'une partie matérielle, ce qui nécessite de faire de la co-estimation matériel-logiciel avec les vrais stimuli en entrée. La modélisation de la consommation d'un ensemble si complexe doit se faire d'une façon simple et rapide pour ne pas trop rallonger le temps de développement du circuit. Il faut donc un outil d'estimation de la consommation qui soit :

- ✓ Suffisamment rapide pour permettre l'analyse d'un SoC tout entier dans un temps raisonnable.
- ✓ Suffisamment puissant pour permettre la modélisation en consommation de tous les composants du système, y compris la partie logicielle et avec les vrais stimuli en entrée.
- ✓ Suffisamment flexible pour permettre la manipulation des paramètres de la consommation de chaque composant et du système.
- ✓ Proposant une précision suffisante pour permettre la comparaison en consommation des différentes solutions architecturales de système.
- ✓ D'utilisation simple pour le concepteur, afin de ne pas augmenter le temps de développement du système.

Le seul moyen d'y parvenir est de relever le niveau d'abstraction des modèles, c'est à dire, de créer des modèles de consommation peu complexes et très abstraits des composants matériels, malgré l'hétérogénéité du système et le haut niveau de description. Le problème est la précision de l'estimation. Une excellente façon d'augmenter la précision en utilisant des modèles simples est de se servir de la simulation au niveau architecture de système. Pour cela et dans le cadre de cette thèse, nous avons fait le choix d'une estimation de la consommation au niveau architecture de système basée sur une simulation cycle-précis.

2.3.3 L'estimation de la consommation au niveau architecture de système basée sur une simulation cycle-précis

La simulation d'une architecture de système au niveau cycle est un moyen très efficace d'analyse des systèmes embarqués dans les premières phases de conception. Pour cette raison, les outils de validation et d'évaluation de performances à ce niveau sont de plus en plus utilisés comme les environnements en SystemC [Sys]. Cette simulation permet le développement de l'architecture d'un système tout entier, l'évaluation globale des choix de partitionnement, le développement des interfaces entre blocs et de la composante logicielle du système et en même temps, l'évaluation des performances du système grâce à la simulation rapide de millions de cycles.

Cette simulation nécessite la création de modèles abstraits des composants matériels, souvent dans des langages de description de l'application algorithmique comme le C/C++, mais suffisamment précis pour pouvoir simuler le comportement au cycle près du composant dans le système avec les vrais stimuli en entrée, d'une façon similaire à sa future implantation matérielle.

En profitant de ces caractéristiques, des études sur la consommation peuvent être envisagées grâce à la simulation architecturale de système au niveau cycle, en ajoutant des modèles de consommation aux modèles fonctionnels des composants. Ainsi la simulation permettrait la co-estimation matériel-

logiciel avec l'analyse en consommation de tout le système, tout en tenant compte des interactions entre composants pendant le déroulement normal de l'application. Certains paramètres de la consommation pourront être obtenus et manipulés en simulation comme la fréquence du fonctionnement et l'activité par composant. Tous ces facteurs nous font espérer que l'estimation de la consommation en utilisant la simulation architecturale de système au niveau cycle, aura une précision suffisamment bonne pour permettre l'exploration de tout un système en vue de l'application des techniques de réduction de la consommation à ce niveau.

Le choix de l'estimation de la consommation avec la simulation architecturale de système au niveau cycle a permis l'insertion dans le contexte industriel de développement d'un circuit chez Philips, dans lequel :

- ⇒ nous suivons la méthode générale de conception des systèmes sur puce,
- ⇒ nous utilisons un simulateur cycle-précis du matériel, et un simulateur de jeu d'instructions du processeur,
- ⇒ nous disposons des modèles de simulation de certains composants matériels chez Philips,
- ⇒ nous modélisons le reste de composants,
- ⇒ nous disposons de la partie logicielle du système.

L'estimation de la consommation au niveau architecture de système cycle-précis reste un problème complexe.

L'objectif de cette thèse est d'essayer de résoudre les problèmes que pose la modélisation de la consommation dans un simulateur au niveau cycle. Pour ceci, on doit être capables de répondre aux questions suivantes :

- ✓ De quel type d'informations sur la consommation pouvons nous disposer à ce niveau pour chaque type de composant ?
- ✓ Comment introduire dans un modèle fonctionnel au niveau cycle, les informations sur la consommation, d'une façon simple pour le concepteur ?
- ✓ Quels sont les paramètres de la consommation que le concepteur pourra changer en simulation ?
- ✓ Comment le calcul de la consommation est il fait dans le modèle et quelles sont les informations obtenues en simulation ?
- ✓ Comment les résultats en consommation obtenus peuvent être validés avec des valeurs obtenues à plus bas niveau ?
- ✓ Comment manipuler les informations de la consommation pour pouvoir tester différentes architectures de système et configurations du matériel ?

Nous allons présenter notre approche qui va tenter de répondre à toutes ces questions en utilisant comme exemple d'application industrielle un circuit incluant un décodeur vidéo MPEG4.

2.4 Conclusion

La réduction de la consommation des systèmes embarqués actuels nécessite de l'application des nouvelles techniques adaptées aux niveaux les plus élevés de la conception : niveaux description algorithmique, description parallèle de système et description architecturale de système. Cela repose

sur la capacité de faire des estimations suffisamment rapides et fiables de la consommation dans les premières phases de la conception.

L'objectif de cette thèse est de proposer une méthode d'estimation de la consommation des systèmes embarqués très tôt dans le flot de conception, sans augmenter notablement la charge du concepteur et permettant l'évaluation de l'application des techniques de basse consommation au niveau architecture de système. Pour cela nous utiliserons la simulation architecturale de système au niveau cycle-précis.

Pour répondre aux questions que cela pose, notre travail consistera donc à :

- ✓ Définir une méthode générale d'estimation de la consommation des systèmes embarqués en utilisant la simulation au niveau cycle.
- ✓ Définir les modèles et les paramètres de consommation nécessaires pour chaque type de composant dans le système.
- ✓ Définir comment faire évoluer les modèles cycle utilisés pour la validation fonctionnelle et l'évaluation des performances, vers des modèles qui permettront le calcul de la consommation.
- ✓ Quantifier la précision de l'évaluation au niveau cycle, par rapport à une évaluation à plus bas niveau.

Chapitre 3

État de l'art

Sommaire

Dans le chapitre précédant nous avons situé l'estimation de la consommation dans le flot de conception des systèmes embarqués actuels. Nous avons vu que l'estimation aux niveaux les plus abstraits permet l'application des techniques de réduction de la consommation les plus efficaces. Par contre, cette estimation est difficile à réaliser et risque de ne pas être suffisamment précise. Différentes approches ont été proposées ces dernières années pour résoudre les problèmes liés à ce type d'estimation. Nous allons les présenter dans ce chapitre.

Nous survolerons d'abord, de façon synthétique, les outils commerciaux d'estimation et d'optimisation de la consommation existants à tous les niveaux de conception. Puis nous présenterons les différentes approches aux niveaux les plus élevés et en particulier au niveau architecture de système cycle-précis. Pour chacun d'entre eux, nous expliquerons :

- ⇒ la méthode suivie,
- ⇒ les modèles et les paramètres de la consommation choisis,
- ⇒ les techniques de réduction de la consommation appliquées,
- ⇒ la façon de valider la méthode et les résultats obtenus,
- ⇒ les avantages et les inconvénients.

Finalement, nous conclurons avec un tableau comparatif pour situer notre approche par rapport aux autres approches.

3.1 Les outils commerciaux d'estimation et d'optimisation de la consommation

Les techniques de réduction de la consommation et les outils d'estimation et d'optimisation qui permettent leur implantation et leur évaluation ont évolué depuis les années 1990, aussi bien dans les milieux académiques qu'industriels. En effet, un outil d'estimation de la consommation est avant tout un calculateur de paramètres qui agissent sur les équations de la consommation dynamique et statique (équations 2.2, 2.5 et 2.7, chapitre 2). En plus, un outil d'optimisation de la consommation permettra la réduction des valeurs de certains de ces paramètres. À chaque niveau de conception l'outil va agir sur des paramètres différents. Parfois, le même paramètre peut être modifié à différents niveaux, pour des groupes ou types d'entités différentes. Par exemple, la tension d'alimentation est un paramètre physique qui peut être modifié au niveau transistors, au niveau portes logiques, pour un groupe de portes dans le même composant, pour le composant tout entier ou même sur tout le circuit. Il en est de même pour d'autres paramètres comme la fréquence ou l'activité.

Le flot de conception avec les outils existants par niveau est illustré dans la figure 3.1. Regardons plus en détail les paramètres affectés par niveau :

Niveau transistors

Les outils d'estimation existant au niveau transistors vont permettre le calcul des capacités des transistors ainsi que les tensions d'alimentation et de seuil pour la technologie choisie. Pour optimiser la consommation, il faudra donc choisir la technologie la plus convenable (à tension variable, SOI, adiabatique, etc) et une librairie de cellules basse consommation aux capacités réduites [Pig04] [CB95].

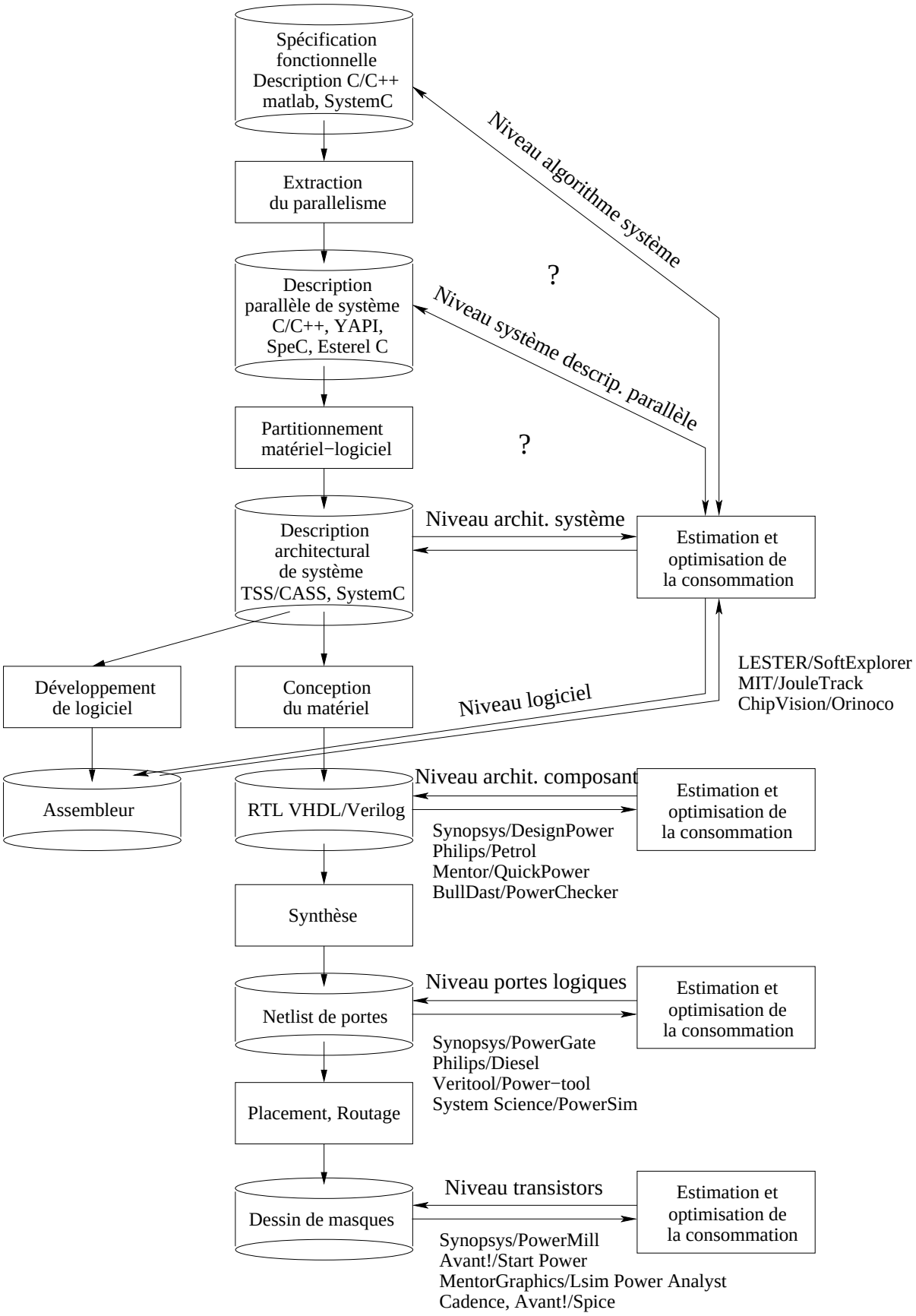

FIG. 3.1 Outils d'estimation et d'optimisation de la consommation dans le flot de conception

On pourra aussi optimiser la méthode de placement/routage des cellules et le dessin de masques final (*layout*), pour diminuer les capacités [BE95]. Parmi les outils de ce type on peut citer *SPICE* [Uni], *PowerMill* de Synopsys [Syna] ou *Lsim Power Analyst* de Mentor Graphics [Men04].

Niveau portes logiques

Au niveau portes logiques, les paramètres à réduire sont plutôt l'activité, les fréquences de travail et les tensions d'alimentation des portes. Certaines technologies permettent même de contrôler la tension de seuil afin de réduire les courants de fuites. Pour agir sur de tels paramètres, nous pourrons appliquer des techniques tels que les horloges inhibées (*clock gating*), l'échelonnage de la fréquence et la tension (*frequency/voltage scaling*), parallélisme, pipelines, architectures distribuées, etc [BE95] [CB95]. Dans cette catégorie on trouvera, entre autres, *PowerGate* de Synopsys [Synb] et *Diesel* de Philips [Phi01a].

Niveau architecture de composant

Au niveau architecture de composant, il s'agit aussi de détecter et de réduire l'activité de commutation, la fréquence et la tension, mais pour des sous-parties du composant. On pourra appliquer des techniques comme les horloges inhibées sur un bloc et sur l'arbre d'horloge [BDIM03], optimiser la taille de mémoires [BMP03], choisir le codage des données sur les communications [Pig04] ou réduire la fréquence et la tension des composants [Usa98]. Se situant dans ce niveau on trouve des outils comme *PowerChecker* de BullDast [Bul], *Petrol* de Philips [PLG98] et *DesignPower* de Synopsys [Synb].

Niveau architecture de système

Au niveau architecture de système, il s'agit aussi de détecter et de réduire l'activité de commutation, la fréquence et la tension, mais pour des composants entiers du système. On pourra optimiser le partitionnement, choisir le type de processeur et la localité des mémoires [PRO01] [LJSM04] [JLSM], changer le type de codage dans le bus [Kel97], la représentation des données et le contrôle sur les accélérateurs matériels [BE95]. On peut aussi implanter une gestion globale de la puissance du système, en contrôlant la fréquence et la tension des composants [BBM98] [CCCa02] [ABR01] [PLS00]. Les outils de cette catégorie serons expliqués dans les sections suivantes, comme *Avalanche* [HL02], *Simunic* [SBM99] et *Ptolemy* [LRDL00].

Niveau algorithmique

Le niveau algorithmique comprends plusieurs types de description qui sont : le code logiciel embarqué, l'algorithme à implanter sur un composant matériel (algorithme composant) et l'algorithme qui décrit l'application entière (algorithme système). A ce niveau, les outils vont chercher à minimiser l'activité des composants matériels. Dans ce but, on peut modifier l'algorithme [BFSS02] et optimiser

le code logiciel [PRO01] et les accès mémoire des instructions et des données [Ca00], comme peut être fait grâce aux outils *SoftPower* [Lau03], *JouleTrack* [SC01] et *Orinoco* [Chi].

Dans les sections suivantes, nous allons décrire les méthodes et les outils existants de nos jours pour l'estimation et l'optimisation de la consommation aux niveaux les plus abstraits du flot de conception : description algorithmique (système, composant ou code logiciel), description système en tâches parallèles et description de l'architecture de système.

3.2 Avalanche : co-estimation matériel-logiciel au niveau instruction

La première approche d'estimation de la consommation au niveau instruction des systèmes embarqués matériel-logiciel a été présentée par Li et Henkel de l'Université de Princeton (USA) en 1998, dans son outil *Avalanche* [LH98] [Hen99]. Ils utilisent un simulateur de jeu d'instructions (*Instruction Set Simulator ISS*) pour simuler un système composé d'un processeur, d'une mémoire cache, d'une mémoire DRAM principale et de coprocesseurs matériels dédiés. Pour tester cette approche, plusieurs algorithmes ont été exécutés dans le processeur ainsi que différentes configurations de taille de la mémoire cache.

La méthode d'estimation de la consommation

L'estimation est faite grâce à la simulation architecturale de système en utilisant un modèle fonctionnel du processeur, simulant le jeu d'instructions en faisant des requêtes aux mémoires cache, à la mémoire principale et aux coprocesseurs matériels. C'est une simulation au niveau instruction parce qu'on analyse les instructions exécutées et les requêtes aux mémoires faites par le processeur et avec ces informations, on calcule le nombre de cycles nécessaires d'exécution et d'attente de chaque composant.

Comme l'illustre la figure 3.2, les instructions exécutées sont détectées à la sortie du simulateur du processeur. Les accès mémoire sont aussi détectés par un traceur de requêtes, qui envoie ces informations à l'outil *Dinero*, pour les traiter et dire s'il s'agit de la lecture d'une instruction ou d'un défaut de cache (*cache miss*) des données ou des instructions. Des modèles d'énergie de chacun des composants ont été rajoutés et avec toutes ces informations l'outil *Avalanche* estime la performance et la consommation de chacun des composants et de tout le système. Dans [Hen99], le traceur détecte aussi les accès aux coprocesseurs dédiés pour permettre l'estimation de leur consommation.

Le modèle d'énergie du **processeur** est composé de la consommation de toutes les instructions et des cycles d'attente dus aux défauts de cache de données et d'instructions. Le courant consommé par instruction est mesuré de façon très précise avec la méthode de Tiwari [TMW94]. La méthode de Li [LH98] ajoute la consommation du processeur pendant les cycles de défaut de cache, durant lesquels le processeur reste inactif en attendant les données ou les instructions, mais en consommant toujours de l'énergie. Pendant la simulation du code, le traceur détecte les cycles d'accès mémoire et les envoie à travers l'outil *Dinero* aux modèles d'énergie et de performance pour calculer les cycles inactifs du processeur. Une autre trace des instructions exécutées sort du modèle de simulation du

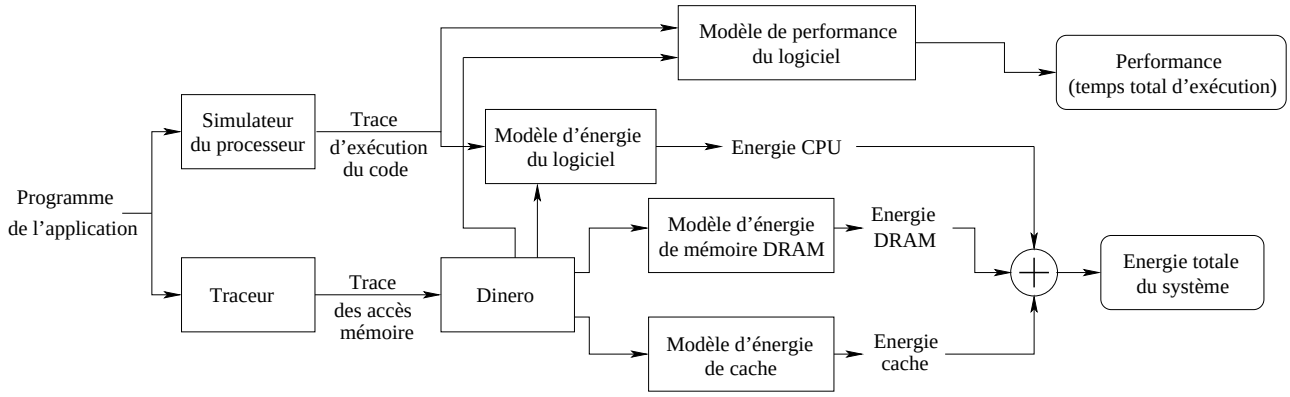

FIG. 3.2 Flot de l'estimation de l'énergie Avalanche, source [Hen99]

processeur et est envoyée aussi aux modèles d'énergie et de performance du processeur. Les modèles utilisent toutes ces informations pour calculer l'énergie totale consommée dans le processeur due aux instructions exécutées et aux cycles de défaut de cache et le temps total d'exécution.

Le modèle d'énergie de la **mémoire cache** est un modèle très précis basé sur des mesures au niveau transistor. Il calcule l'énergie consommée par accès au cache et selon le nombre de bits transférés. La trace des accès au cache et des défauts de cache est obtenue grâce au traceur qui envoie ces données au modèle. Ces valeurs permettent le calcul de la consommation due aux accès en lecture et écriture du cache et aussi celle due aux cycles de défaut de cache.

Le modèle d'énergie de la **mémoire DRAM principale** calcule le courant consommé par accès selon le nombre de cellules activées par transfert et la taille de la mémoire. La taille de la mémoire influence beaucoup sa consommation, qui croît avec. La trace des défauts de cache est envoyée au modèle d'énergie de la mémoire et grâce à cette valeur on détermine le nombre d'accès, ce qui va permettre le calcul de l'énergie consommée. L'énergie des périodes inactives n'est pas comptabilisée, se basant sur l'hypothèse d'une consommation nulle dans ce cas pour une mémoire DRAM.

La consommation des **coprocesseurs dédiés** est considérée comme constante [LH98] et elle n'est pas comptabilisée, ou bien elle est calculée grâce à un estimateur au niveau porte [Hen99], qui prend en compte les accès aux coprocesseurs depuis le processeur.

Les techniques de basse consommation applicables

L'outil *Avalanche* est utilisé pour l'exploration en consommation des transformations logicielles et de taille des mémoires cache et DRAM. Les modifications apportées sur ces composants non seulement changent la consommation du composant lui même, sinon qu'elles modifient aussi les rapports avec les autres composants du système et, en conséquence, leur consommation.

Les modifications opérées sur le **logiciel** sont le *inlining* et le déroulement des boucles. Le *inlining* élimine les appels et les retours de fonction, tandis que le déroulement des boucles élimine les boucles. Cela supprime une partie du code de contrôle et augmente le parallélisme au niveau instruction. Ces deux techniques augmentent la performance du code en diminuant le nombre d'accès mémoire mais provoquent indirectement l'augmentation de la taille du code et le *working set* en cache, nécessitant

dans ce cas des mémoires (cache et principale) plus grandes.

La **taille de la mémoire cache** peut être augmentée pour réduire le nombre de défauts de cache, diminuant dans ce cas, la consommation dans le processeur et dans la mémoire DRAM principale. Par contre, la consommation du cache augmente avec sa taille, donc il faut trouver un bon compromis entre la taille et le nombre de défauts de cache. Dans la figure 3.3, il est illustré la contribution à la consommation totale pour l'application MPEG, du logiciel, de la mémoire principale, du cache de données de 4Kbytes et du cache d'instructions de taille variable.

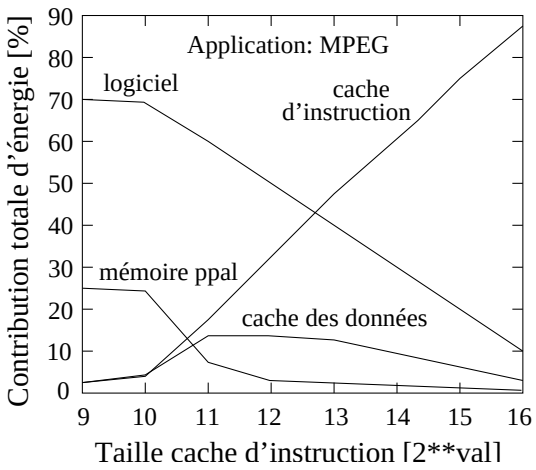

FIG. 3.3 Impact de la taille du cache d'instructions sur la consommation d'énergie, source [Hen99]

Une méthode de sélection d'algorithmes a été mise en place, en attribuant des priorités à chaque paramètre selon leur impact sur la consommation et la performance globale du système.

L'exploration du partitionnement matériel-logiciel a été étudiée en [Hen99] grâce à l'utilisation conjointe du simulateur de jeu d'instructions, pour calculer la consommation du processeur et d'un estimateur au niveau portes, pour calculer la consommation des coprocesseurs dédiés.

La validation de l'approche

Cette approche a été validée en utilisant un système composé d'un processeur SparcLite de Fujitsu, des caches d'instructions et des données et d'une mémoire DRAM. Les modèles de consommation ont été introduits dans le simulateur.

Aucune comparaison des résultats obtenus avec des mesures faites sur une implantation matérielle du système ou grâce à une estimation aux niveaux de description plus bas (niveau portes ou transistors), n'est citée dans l'article. Ainsi la précision de la méthode n'est pas connue. Cependant, les modèles d'énergie utilisés pour le processeur et les mémoires sont très précis. En ce qui concerne les coprocesseurs dédiés, un estimateur au niveau porte est utilisé, ce qui permet d'augmenter la précision pour ce type de composant difficilement modélisable en haut niveau. Il faut noter que les estimations sont prises à différents niveaux d'abstraction selon le type de composant.

Les avantages et les inconvénients

Cette approche a été la première à s'attaquer au problème de l'estimation de la consommation des systèmes embarqués au niveau architecture de système. Elle permet l'application d'optimisations algorithmiques, de hiérarchie mémoire et une première exploration des choix de partitionnement matériel-logiciel.

Un des principaux avantages est la précision des modèles d'énergie utilisés qui prend en compte des paramètres de très bas niveau comme le nombre des cellules actives par accès dans la mémoire DRAM, mais aussi l'impact des rapports entre matériel et logiciel grâce à la détection des défauts de cache et des accès mémoire. En outre, l'utilisation d'un estimateur de la consommation au niveau portes pour le matériel dédié peut donner une précision élevée à son estimation. Par contre, cela ralenti énormément la simulation.

De plus, sachant que l'erreur de l'approche n'a pas été évaluée, le fait de ne pas utiliser la co-simulation matériel-logiciel et de n'employer pour le calcul de la consommation que les traces des accès aux composants, nous laisse penser que la précision ne sera pas bonne. Il manque l'évaluation des effets des interactions temporelles entre les composants. Il en résulte que pour certains composants, on ne connaîtra pas la longueur des périodes d'inactivité, qui peut être variable en fonction de paramètres comme le taux d'occupation du bus, etc. En conséquence, la consommation de ces périodes ne sera pas prise en compte. Cela peut conduire à des erreurs considérables dans des systèmes guidés par les données, temps réel et/ou dépendants des entrées, où certains composants ne sont utilisés qu'à des moments bien précis et inactifs le reste du temps, mais en consommant toujours de l'énergie.

En outre, l'absence de modélisation des interconnexions rajoute aussi de l'erreur, puisque non seulement l'énergie qu'ils dépensent ne sera pas comptabilisée, mais également les délais qui peuvent être introduits dans le système ne seront pas pris en compte (par exemple par l'occupation du bus par d'autres composants).

Telle qu'elle est décrite, cette approche présente une limite supplémentaire, celle de ne pas analyser l'évolution de la consommation dans le temps. Les valeurs de consommation ne sont obtenues qu'à la fin de la simulation et pour des paramètres fixés au début de la simulation. Cela empêche l'analyse dans le temps des différentes configurations de certains paramètres comme la fréquence de travail, la tension du processeur et des mémoires ou l'implantation d'une stratégie dynamique de contrôle des horloges.

3.3 Simunic : estimation logicielle au niveau instruction cycle-précis

Une approche d'estimation de la consommation au niveau instruction cycle-précis est présentée par Simunic et al. de l'Université de Standford (USA) en 1999 [SBM99]. Le cadre de son étude est une application purement logicielle exécutée sur un système composé d'un processeur RISC, des mémoires cache, des mémoires de données et d'instructions et des interconnexions. Le but de son étude est de trouver la meilleure configuration en consommation et en performance de la hiérarchie

mémoire.

La méthode d'estimation de la consommation

La méthode d'estimation de la consommation choisie utilise un simulateur de jeu d'instructions d'un processeur (*ISS*), enrichi avec des informations sur la consommation des principaux composants du système : processeur, caches, mémoires et interconnexions. Les composants sont considérés comme des boîtes noires et les modèles de consommation par composant sont ajoutés au simulateur comme l'illustre la figure 3.4.

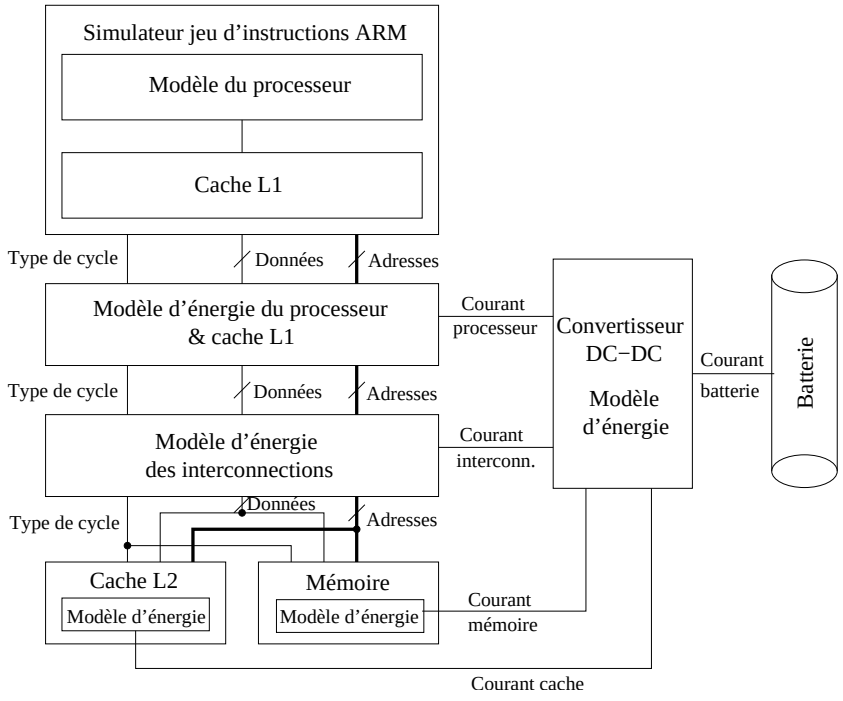

FIG. 3.4 Architecture du simulateur, source [SBM99]

Les modèles d'énergie sont construits à partir des informations de consommation recueillies dans les notices techniques de chaque composant. Ces informations sont la puissance moyenne, la fréquence et la tension d'alimentation, mesurées pendant le fonctionnement normal du composant. À partir de ces informations, l'énergie est calculée par cycle de travail et pour une valeur spécifique de tension d'alimentation.

Pendant la simulation et pour chaque instruction exécutée, le simulateur envoie à l'extérieur des informations sur l'état du processeur et sur les accès mémoire. Avec ces informations le modèle d'énergie de chaque composant calcule l'énergie consommée par cycle. Chaque composant peut se trouver dans un état actif ou inactif. S'il s'agit de l'exécution d'une instruction depuis le cache L1, le processeur et ce cache sont actifs et le reste (cache L2, mémoire et interconnexions) sont inactifs. S'il s'agit par contre d'un défaut de cache, donc d'un accès mémoire, le processeur et le cache seront en état d'attente de données ou d'instructions qui viennent de la mémoire, donc inactifs, et la mémoire et les interconnexions seront actives pendant la durée du transfert. Durant la simulation, on cumule les

énergies consommées par chacun selon son état et on obtient à la fin de la simulation la consommation d'énergie totale par composant et globale dans le système.

L'approche est similaire à celle d'Avalanche, sauf qu'ici on utilise la simulation architecturale de système au niveau cycle-précis et les modèles de consommation du processeur et des mémoires sont beaucoup plus simples.

Les techniques de basse consommation applicables

En vue de la réduction de la consommation, cette approche a été utilisée pour explorer différentes **configurations de mémoires**. Les mémoires testées sont les mémoires d'instructions et de données, avec ou sans une deuxième mémoire cache (cache L2). Comme mémoire d'instructions, une mémoire basse consommation FLASH et une mémoire FLASH rapide ont été testées. Comme mémoire de données, on a utilisé une mémoire SRAM basse consommation, une mémoire SRAM rapide et une mémoire SDRAM très rapide et à forte consommation energétique.

Différentes configurations de ces mémoires ont été testées. On a trouvé que la mémoire d'instructions consomme très peu d'énergie par rapport à la mémoire des données. On a choisi donc comme mémoire d'instructions la mémoire FLASH rapide, qui satisfait largement les contraintes en vitesse et en consommation. Comme mémoire de données, la SRAM basse consommation présente un problème de largeur de données, ce qui augmente notablement le nombre d'accès et, par conséquent, la consommation totale. Par contre, la mémoire SDRAM à grande vitesse n'a pas ce problème. Sa largeur de données est suffisante pour le système, ce qui permet d'accéder aux données plus rapidement avec moins d'accès et en utilisant globalement moins d'énergie.

La validation de l'approche

La méthode a été validée sur un système qui utilise le simulateur de jeu d'instructions de l'ARM, l'ARMulator [ARM96], avec les modèles fonctionnels des toutes les mémoires et ayant comme application logicielle un décodeur MPEG.

Les estimations obtenues en consommation et en performance ont été comparées aux mesures directement prises sur une plate-forme physique de prototypage, celle-ci contenant un processeur StrongARM-1100, un convertisseur de courant DC-DC, une mémoire FLASH et une mémoire SRAM, sur une carte de circuit imprimé. Le courant moyen consommé par le processeur et le courant total sortant de la batterie sont mesurés avec des multimètres numériques. Le temps d'exécution est mesuré avec le minuteur du processeur.

Ils ont mesuré les courants avec différentes valeurs de fréquence du processeur. Ils annoncent que les estimations de la simulation cycle-précis avec ARMulator et les modèles d'énergie ont une erreur de seulement 5% par rapport aux mesures sur la plate-forme physique. L'approche est donc très précise.

Les avantages et les inconvénients

Le principal avantage de cette approche est la très bonne précision par rapport à la simplicité des modèles d'énergie développés. Les modèles sont suffisamment abstraits pour être facilement implantés et le simulateur de jeu d'instructions permet la simulation rapide du processeur et des mémoires à ce haut niveau de description.

Par contre la simulation de tout un SoC embarqué avec d'autres composants matériels comme des processeurs DSP, des accélérateurs dédiés, etc., n'a pas été testée, ce qui permettrait de généraliser l'approche. Aussi, son apport reste assez limité puisque les moyens de tester les changements dynamiques sur des paramètres du système, comme la fréquence ou la tension d'alimentation, en vue de l'application d'une gestion dynamique de la puissance, n'ont pas été analysés.

Cette approche est très intéressante mais pas suffisamment exploitée. En poussant plus loin l'analyse, on pourrait tester différentes optimisations logicielles, la gestion du cache, l'ajout d'autres composants matériels et finalement l'application et l'analyse d'une gestion dynamique de la puissance des composants de tout le système. Notre travail de recherche sur l'estimation de la consommation d'un SoC complet au niveau architecture de système cycle-précis a été basé sur le travail de Simunic [SBM99].

3.4 Ptolemy : co-estimation matériel-logiciel à plusieurs niveaux

Ptolemy est un outil de développement des systèmes embarqués matériel-logiciel, développée par Lajolo et al. à l'Université Polytechnique de Turin (Italie) en 2000 [LRDL00]. Ces systèmes sont composés de processeurs, matériel dédié et interconnections. Entre autres applications, *Ptolemy* permet de faire de la co-estimation de la consommation au niveau système, grâce à l'utilisation de plusieurs outils d'estimation des différents composants du système, fonctionnant de façon concurrente et synchronisée.

La méthode d'estimation de la consommation

Pour l'estimation de la consommation, le système est décrit au niveau comportemental en langage esterel [Ber91] comme un ensemble de tâches communicantes. L'utilisateur affecte les tâches au matériel : coprocesseurs dédiés ou processeurs (logiciel embarqué) et définit les communications entre composants. A partir de cette information, l'outil de compilation *Polis* génère le code C à partir du code esterel pour chaque tâche qui sera implantée en logiciel et la netlist pour les tâches implantées en matériel dédié. Cette netlist est générée grâce à une synthèse logique rapide. En plus de cela, un modèle avec les événements de tout le système est crée, ce qui va permettre la synchronisation de l'exécution des simulateurs du logiciel et du matériel.

Le flot de co-estimation de l'outil est illustré dans la figure 3.5. Pour simuler la partie logicielle du système, un simulateur de jeu d'instructions du processeur (*ISS*) peut être utilisé, ainsi qu'un simulateur au niveau architectural RTL avec le modèle du processeur. La partie matérielle est simulée grâce à un simulateur au niveau architectural RTL ou portes. L'outil *Ptolemy* permet la simulation conjointe de tout système, en synchronisant le transfert des données entre les différents simulateurs.

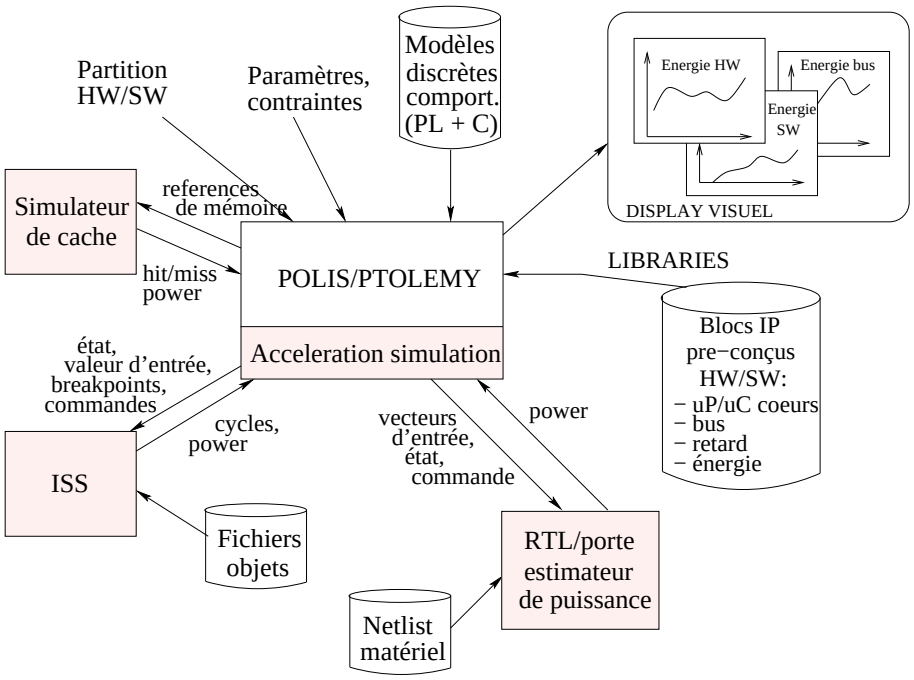

FIG. 3.5 Flot de co-estimation *Ptolemy*, source [LRDL00]

Les modèles de consommation utilisés pour chaque type de composant dans les différents simulateurs sont les suivants :

⇒ **Modèle du processeur** : le simulateur de jeu d'instructions est enrichi avec des modèles de consommation par instruction avec la méthode Tiwari [TMW94] ou bien la simulation RTL du modèle du processeur fournit l'estimation de consommation. La consommation des caches peut être ajoutée au modèle.

⇒ **Modèle du matériel dédié** : un estimateur de consommation au niveau RTL ou porte, donne les informations de la consommation de la netlist au niveau cycle-précis.

⇒ **Modèle du bus** : la simulation avec *Ptolemy* fournit l'activité des communications et un modèle simple du bus permet le calcul de sa consommation à partir de cette activité et de la capacité estimée des fils.

L'outil donne des estimations en consommation à différents niveaux et suffisamment précises pour permettre l'optimisation au niveau système. Par contre, la simulation étant trop lente, des techniques d'accélération ont dû être mises en place pour améliorer la vitesse de l'estimation, ce qui affecte la précision de l'estimation.

Les techniques de basse consommation applicables

Dans [LRDL00], des études sur les **communications sur le bus** ont été menées. En particulier, on a évalué la consommation du système en utilisant différentes priorités des accès au bus et plusieurs tailles des transferts. On a montré que seul les variations de ces deux paramètres ont un grand impact sur la consommation de tous les composants du système, même sans changer le partitionnement.

Cette méthode permet aussi de valider le **partitionnement des tâches et le choix de l'architecture du système cible**. Plusieurs options peuvent être testées avant de choisir la meilleure du point de vue des performances et de la consommation.

La validation de l'approche

Cette approche a été validée en utilisant le simulateur de jeu d'instructions du processeur SparcLite de Fujitsu et un environnement de synthèse logique appelé SIS, pour l'estimation de la consommation du matériel dédié.

L'erreur annoncée des modèles de consommation du simulateur d'instructions est de 5%, en comparaison aux mesures de courant sur le même processeur implanté en matériel. La précision de l'estimation du matériel dédié n'est pas mentionnée ; sachant que l'estimation est faite sur une description au niveau portes cycle-précis obtenue grâce à la synthèse rapide, la précision doit être bonne, sauf si cette synthèse donne des résultats très éloignés de ceux obtenus par la synthèse classique. Il faut noter que les estimations sont prises à différents niveaux d'abstraction selon le type de composant. Cela permet d'augmenter la précision des composants difficilement modélisables en haut niveau comme les nouveaux coprocesseurs dédiés et les interconnexions.

Les avantages et les inconvénients

Le principal avantage de cette approche est l'augmentation de la précision de l'estimation grâce à l'utilisation de la co-estimation. Cela permet de prendre en compte les effets des interactions temporelles entre composants grâce à la synchronisation des simulations entre différents simulateurs. De plus, la bibliothèque des composants est très riche.

Un autre avantage vient du fait que cette approche est la première à permettre une estimation de la consommation au cours du temps : elle permet de suivre l'évolution de la consommation de tout le système et en conséquence de trouver le maximum de dissipation de chaque composant en vue des optimisations dynamiques en consommation.

Le grand inconvénient est la lenteur de l'estimation provenant de l'utilisation de la simulation de chaque composant au niveau RTL ou portes. Ce problème est amélioré en partie par les techniques d'accélérations proposées, mais qui diminuent la précision de l'estimation. En outre, l'utilisation de la synthèse rapide des composants matériels dédiés peut introduire des erreurs dans les estimations. Et dans l'exemple montré, la consommation des mémoires n'est pas prise en compte, ce qui est un facteur très important de la consommation et de la performance des SoC embarqués actuels pour le marché portable.

3.5 DTSE : optimisation algorithmique en transfert et stockage de données

DTSE (*Data Transfer and Storage Exploration*) est une approche d'optimisation de la consommation au niveau algorithmique développée à l'IMEC (*Interuniversity MicroElectronics Center*) en Belgique par l'équipe de F. Catthoor [Ca00]. Développé principalement pour les applications multimédia dominées par les données, les optimisations du DTSE cherchent à diminuer la consommation des mémoires, en réduisant la fréquence des accès, augmentant la localité et la régularité des données et optimisant la hiérarchie des mémoires.

La méthode d'optimisation de la consommation

Sachant qu'une grande partie de la consommation dans les systèmes à forte composante des données, vient des accès à la mémoire, dans cette approche on cherche à réduire le nombre des accès. L'entrée de cette méthode est la spécification fonctionnelle de l'application, avec les accès aux signaux multidimensionnels qui doivent pouvoir être ordonnés statiquement. Après les traitements d'optimisation opérés, la sortie sera une netlist de mémoires et de générateurs d'adresses, combinée à la spécification initiale transformée. Cette netlist sera l'entrée à l'outil de synthèse de l'architecture de composant de haut niveau ou au compilateur du logiciel. Les tâches du traitement les plus longues à faire sont supportées par l'environnement *Atomium* [Ato]. Les générateurs d'adresses sont créés grâce à la méthode appelée *Adopt* [MCJM96].

La méthode d'optimisation algorithmique cherche à minimiser de façon systématique le stockage en mémoire et les transferts pour réduire la puissance consommée et la taille du circuit. Pour ce faire, les étapes suivies sont les suivantes :

- Diminuer la taille du code à optimiser pour se concentrer sur les parties importantes.
- Transformer l'algorithme en réduisant la largeur des données à stocker : changer l'algorithme et/ou le type des données (virgule flottante, etc.).
- Diminuer la redondance des accès des données.
- Augmenter la régularité et la localité grâce aux transformations dans les boucles.
- Exploiter la hiérarchie des mémoires en approchant les données les plus utilisées aux parties opératives et en utilisant des mémoires plus petites et moins gourmandes en énergie.
- Distribuer le budget des accès mémoire de façon équilibrée pour réduire la largeur de bande et des ports d'accès aux mémoires.
- Distribuer l'organisation des mémoires et de la hiérarchie.
- Exploiter la durée de vie des tableaux pour les chevaucher dans le temps.

La validation de l'approche

Cette approche a été testée sur une application de synchronisation démodulateur/décodeur canal pour la diffusion d'audio numérique (*Digital Audio Broadcast*). L'algorithme définissant la spécification fonctionnelle est optimisé avec la méthode DTSE. Ensuite, il est implanté sur une architecture de

système avec DSP en utilisant l'outil de synthèse haut niveau *Mistral2*. Le jeu d'instructions ainsi créé est spécifiquement adapté au programme qui sera exécuté, donc le circuit résultant sera du type ASIP (*Application Specific Instruction-set Processor*). Les modifications opérées sur l'algorithme ont des répercussions sur l'implantation architecturale du ASIP. Le résultat de la synthèse avec *Mistral2* est la description VHDL au niveau RTL. La consommation de la spécification de référence et de celle modifiée avec la méthode DTSE sont estimées avec l'outil *Petrol* [PLG98]. Selon les estimations fournies par cet outil, les techniques DTSE ont permis la réduction de la consommation de la mémoire SDRAM de 25%. La contribution en consommation de la mémoire d'instructions est identifiée comme la plus importante et quelques idées pour sa réduction ont été fournies.

Les avantages et désavantages

Cette approche montre des travaux très poussés en optimisation de la consommation au niveau algorithmique, très utiles pour des applications multimédia avec beaucoup de données comme le décodage MPEG4. Elle apporte de grandes réductions en consommation dans les mémoires et dans les interconnexions. Étant des optimisations algorithmiques, elles peuvent aider au partitionnement de façon à optimiser le choix des mémoires et à faciliter l'accès depuis n'importe quel processeur ou coprocesseur du système. Par contre, une fois le choix de l'architecture de système fait, des optimisations architecturales de système à un autre niveau doivent être assurées pour minimiser davantage la consommation. Le niveau de cette approche est plus élevé de celui que nous poursuivions.

3.6 JouleTrack : estimation logicielle pour processeur RISC

JouleTrack est une méthode et un outil d'estimation de la consommation du logiciel décrit au niveau instruction développé en 2001 par Sinha et al. de l'Institut de Technologie du Massachusetts (USA) [SC01]. L'outil permet de faire le profilage en consommation à partir du code C, à plusieurs niveaux d'estimation pour un processeur cible StrogARM SA-1100 ou Hitachi SH-4.

La méthode d'estimation de la consommation logiciel

La première méthode d'estimation de la consommation d'un programme logiciel a été développée à l'Université de Princeton par V. Tiwari [TMW94]. Cette méthode est souvent considérée comme référence dans l'estimation de la consommation des processeurs et elle est applicable théoriquement à tous processeurs que ce soit des processeurs généraux (Pentium, ARM, etc.) ou les processeurs spécifiques (DSP, etc.). Sachant qu'un programme est une suite d'instructions, cette méthode consiste à accumuler l'énergie dissipée par chaque instruction du code dans le processeur cible. Pour évaluer chacune, un petit programme en assembleur est créé, dans lequel cette seule instruction est répétée un grand nombre de fois. Le programme est donc exécuté en boucle pour permettre la mesure du courant moyen consommé. La précision est améliorée en prenant compte aussi de la consommation d'inter-instructions, qui vient de l'interaction entre deux instructions successives. Pour la mesurer, il faut exécuter toutes les combinaisons d'instructions deux à deux. La méthode de Tiwari, étant très précise, nécessite beaucoup d'analyses et de mesures et par conséquence, elle devient trop lente.

Les expérimentations du Sinha [SC01] sont faites sur les processeurs StrongARM SA-1100 [Cor00] et Hitachi SH-4 [Fa98], processeurs basse consommation pour l'embarqué. Ils ont mesuré la consommation par instruction en exécutant un programme en boucle jusqu'à trouver une mesure stable pour l'instruction en question, puisqu'à ce moment, l'instruction et la donnée se trouvent dans leur cache correspondant. Les mesures sont prises grâce à un multimètre placé sur l'alimentation du circuit, avec les caches déjà chargés et pour plusieurs modes d'adressage et entrées de chaque instruction.

La figure 3.6 illustre les valeurs moyennes de courant de toutes les instructions de l'ARM SA-1100. On observe que les différences de courant entre instructions sont assez petites. La variation la plus importante est de 0.072 A, ce qui correspond à 38% de la moyenne de l'ensemble. Les plus grandes valeurs de courant correspondent aux instructions *load* et *store*. Cela est dû à l'accès cache qu'elles génèrent pour aller lire ou écrire une donnée. Pour le processeur Hitachi la variation est la même à 38%.

De plus, d'autres mesures prises pendant l'exécution de différents programmes (FFT, FIR, DCT...), montrent que la variation maximale de courant entre programmes est de seulement 8%. Cela veut dire que les différences de consommation entre instructions sont atténuées par les effets d'inter-instruction pendant l'exécution d'un programme. En conséquence, la consommation des différents programmes devient assez uniforme. Par contre, les variations en consommation venant des changements de la tension d'alimentation et de la fréquence de travail du processeur sont très significatives pour tous les programmes.

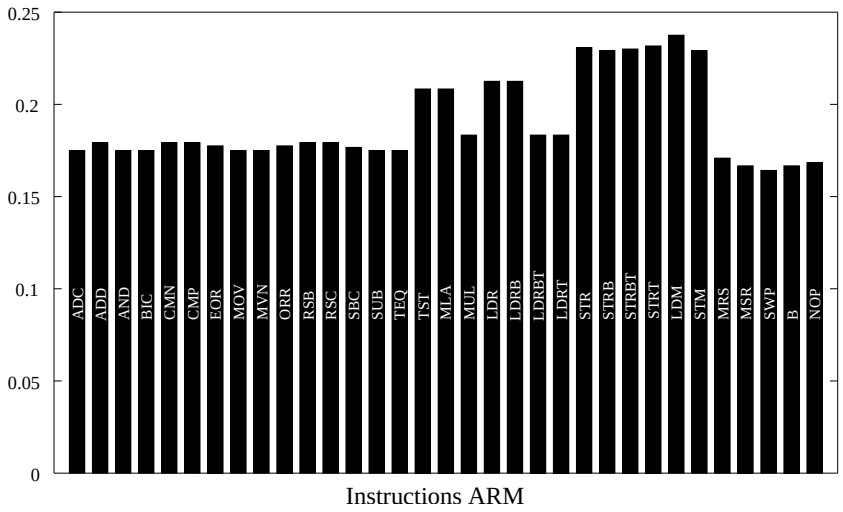

FIG. 3.6 Courant consommé par le jeu d'instructions du StrongARM SA-1100, source [SC01]

En conclusion, ce travail montre que l'estimation de la consommation des processeurs embarqués avec des modèles très élaborés, comme ceux de Tiwari, contenant la consommation de toutes les instructions avec les effets des inter-instructions est trop détaillée, puisque la consommation varie beaucoup plus en fonction de la fréquence et de la tension d'alimentation du processeur, qu'en fonction du programme.

En conséquence, le modèle de consommation qu'ils proposent pour ce type de processeur est très simple :

$$P_{tot} = V_{dd} I_0(V_{dd}, f) \Delta t \tag{3.1}$$

où V_{dd} est la tension d'alimentation, I_0 le courant moyenne par instruction du processeur à cette tension, f la fréquence et Δt le temps d'exécution du programme.

Ce modèle a été inséré dans un outil d'estimation de la consommation appelé *JouleTrack*. Il permet l'estimation de la consommation d'un code écrit en C et exécuté sur le processeur StrongARM SA-1100. Le code est compilé et exécuté sur le simulateur de l'ARM. Le calcul de la puissance est fait à partir de la formule 3.1 en choisissant une fréquence, pour laquelle l'outil trouvera dans une libraire la tension minimale nécessaire et le courant moyen et la simulation donnera le temps d'exécution. L'outil dispose de plusieurs niveaux d'estimation et il est disponible sur le web : http ://carls-berg.mit.edu/jouletrack/JouleTrack/.

Les techniques de basse consommation applicables

Les techniques de réduction de la consommation appliquées sur ce type de processeurs sont, comme nous venons de le voir, la réduction de la tension d'alimentation et de la fréquence de travail. Ces paramètres peuvent être réduits jusqu'à atteindre certaines limites technologiques et toujours respectant le temps d'exécution maximal permis par l'application.

La réduction de la tension à fréquence fixe, provoque une diminution en consommation d'énergie presque quadratique. La réduction de la fréquence à tension fixe, diminue linéairement l'énergie (et augmente linéairement le temps d'exécution). Plusieurs combinaisons des deux paramètres peuvent être appliquées jusqu'à certaines limites. Par exemple, à une tension de 1.4 V, la fréquence maximale dans laquelle le StrongARM peut travailler est de 206MHz. C'est selon l'application que l'on va décider les valeurs convenables de ces paramètres. Ils peuvent même être adaptés dynamiquement selon les besoins temporels de l'application.

Les avantages et désavantages

La méthode développée par Sinha et al. montre qu'un modèle simple, ne prenant en compte que la tension d'alimentation et la fréquence, peut donner des résultats relativement précis pour des architectures de processeurs simples. L'approche peut donc être appliquée pour l'estimation de la consommation logicielle d'un processeur RISC simple dans un système embarqué. Dès lors que l'architecture du processeur devient plus complexe, cette approche n'est plus intéressante puisqu'elle génère des erreurs d'estimation importantes.

3.7 SoftExplorer : estimation logicielle pour processeur DSP

SoftExplorer est un outil d'estimation de la consommation pour des processeurs commerciaux, spécialement pour ceux de traitement de signal DSP, présenté en 2001 par J. Laurent et al. de l'Université de Bretagne Sud (France) [LJSM04] [JLSM] [Lau03]. L'entrée de l'outil est soit le code C,

soit le code en assembleur généré par le compilateur ou écrit directement par le programmeur. L'outil fournit comme résultat la consommation totale et locale (de chaque boucle), ainsi que la distribution de la consommation dans le code. Il peut donner aussi la consommation d'énergie de la mémoire externe et la répartition de la consommation entre le processeur et la mémoire. Son but est de procurer au concepteur un retour rapide de la consommation de son algorithme, exécuté sur plusieurs types de processeurs, pour optimiser le code et/ou choisir le processeur le moins consommant pour son application.

La méthode d'estimation de la consommation

La méthode d'estimation est basée sur l'analyse fonctionnelle de la consommation dans le processeur (Functional Level Power Analysis : FLPA), dont le résultat est un modèle de consommation de la cible. Grâce à cette analyse, un nombre limité de mesures est suffisant pour déterminer le modèle de consommation. De plus, cette méthode prend en compte toutes les fonctions du processeur que ce soit le contrôle du pipeline, les unités de traitement, les mémoires internes ainsi que les communications du cache avec la mémoire.

La méthode d'estimation de la consommation de puissance, illustrée dans la figure 3.7, est constituée de deux parties : la définition du modèle et le processus d'estimation.

- **La définition du modèle de puissance** est basée sur l'analyse fonctionnelle du point de vue de la consommation, de l'architecture du composant cible. Cette analyse permet de créer un modèle de puissance basé sur des fonctions mathématiques. Ces fonctions sont des équations déterminées à partir d'un nombre réduit des mesures physiques réalisées sur la cible et variant quelques paramètres algorithmiques et de configuration.
- **Le processus d'estimation** est fait pour chaque algorithme et permet de connaître la consommation de l'application. Pour ce faire, un analyse du code est fait pour extraire les paramètres algorithmiques directement de l'algorithme par un analyse fait à la main ou automatisé dans l'outil *SoftExplorer*. Ensuite ces paramètres sont insérés sur les équations du modèle de puissance pour obtenir la consommation de l'application.

La méthode a été appliquée sur les processeurs DSP TI TMS320C6201, TI TMS320C6701, TI TMS320C5510 et ARM7TDMI. Le temps d'estimation est de quelques secondes. L'outil dispose d'un interface graphique très facile à utiliser.

Les techniques de basse consommation applicables

Cet outil permet d'envisager des **optimisations algorithmiques** visant à réduire le nombre d'instructions ou d'accès mémoire du code. Il facilite aussi **le choix du processeur** puisque le même algorithme peut être testé sur différents processeurs, pour trouver celui qui consommerait le moins pour une application particulière.

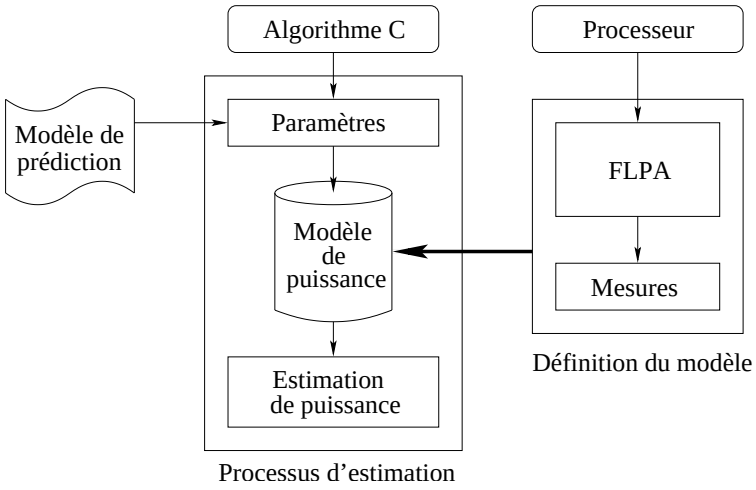

FIG. 3.7 Méthodologie d'estimation, source [LJSM04]

La validation de l'approche

Différentes applications ont été testées sur différents modèles des processeurs, pour valider la méthode afin de montrer sa précision. L'erreur moyenne annoncée de l'estimation est d'environ 3% pour le C55 et d'environ 2% pour le C62, par rapport aux mesures prises sur une plate-forme matérielle physique.

Les avantages et les inconvénients

Cette méthode est une approche intéressante pour plusieurs raisons. D'abord, l'utilisation du langage C suppose un grand avantage parce que cela permet une meilleure interactivité entre l'estimation de la consommation et le développement de l'application. En outre, le temps de développement d'un modèle de puissance d'un processeur est relativement court (de l'ordre d'un mois) pour des estimations de consommation très précises. L'architecture interne du composant cible ne nécessite pas d'être connue. De plus, cette méthode est non seulement applicable sur des processeurs VLIW mais également sur des processeurs RISC qui, par leur conception plus simple, permettent le développement du modèle dans un temps plus court.

Le seul inconvénient vient du fait que ces estimations ne sont pas couplées avec celles portant sur les éléments matériels du système (ASICs, systèmes multi-processeurs, différents types de mémoires, etc), autorisant ainsi la co-estimation matériel-logiciel des SoC complexes.

3.8 JouleDoc : coprocesseur d'estimation en exécution

JouleDoc est un coprocesseur spécialisé pour l'estimation de l'énergie des systèmes embarqués matériel-logiciel, développé par Haid et al. à l'Université de Graz (Autriche) et présenté en 2003 [HKSW03]. Il est capable d'estimer dynamiquement l'énergie consommée par un système déjà implanté en matériel et en train de s'exécuter, dont les composants ont été préalablement instrumen-

talisés pour l'estimation. Le but est de permettre l'implantation d'un bloc de gestion dynamique de la puissance, qui établirait une politique de contrôle des paramètres du système liés à la consommation, en fonction des informations de consommation reçues du coprocesseur pendant l'exécution de l'application.

La méthode d'estimation de la consommation

L'estimation de la consommation est faite à l'intérieur du coprocesseur JouleDoc, en utilisant des informations reçues des composants matériels du système dont on veut estimer la consommation. Des détecteurs sont installés à chaque composant, servant à intercepter et envoyer au coprocesseur des informations sur les événements produits. Ces événements sont, par exemple, le nombre de lectures d'instruction, de défauts de cache, d'opérations de lecture/écriture en mémoire, etc. L'estimation de la consommation est faite dans le coprocesseur à partir de ces informations, de façon parallèle à l'exécution de l'application dans le SoC. De cette façon la performance du système n'est pas affectée.

Dans la figure 3.8, nous illustrons le diagramme des blocs du coprocesseur JouleDoc. Le calcul de la consommation est fait en utilisant des macro-modèles d'énergie des composants. Les macro-modèles disposent des valeurs de consommation d'énergie par événement. Les valeurs sont obtenues soit à partir des notices techniques, soit par des calculs faits à partir du nombre de portes, activité, etc. ou encore à partir des mesures prises en simulation. Le coprocesseur accumulera les valeurs correspondantes aux événements survenus. Pour cela, un groupe de registres est alloué par événement, dans lequel sont stockées la valeur d'énergie de l'événement, l'énergie accumulée et une valeur du temps. Les valeurs d'énergie par événement de chaque composant matériel sont reconfigurables.

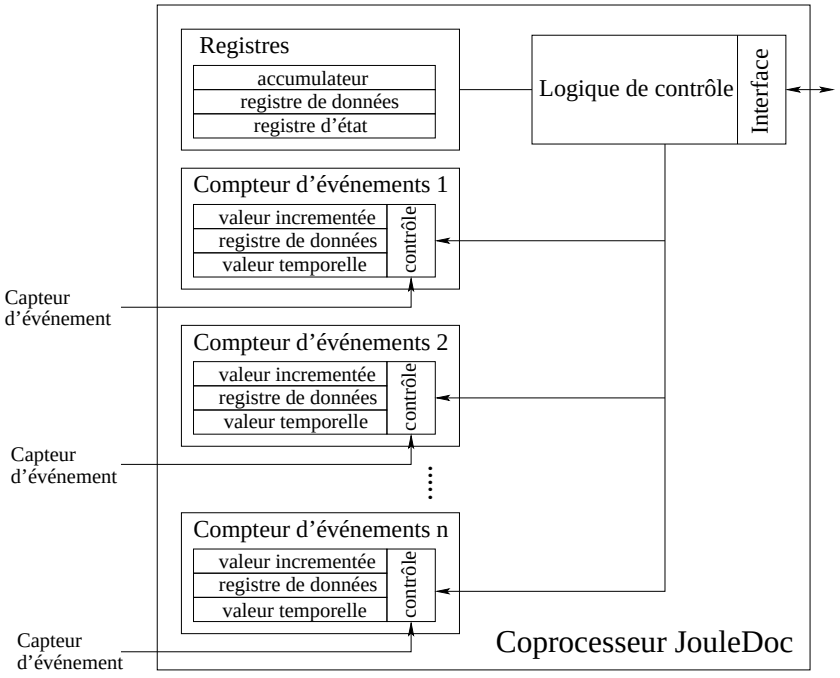

FIG. 3.8 Diagramme des blocs du coprocesseur JouleDoc, source [HKSW03]

Les techniques de basse consommation applicables

Le coprocesseur JouleDoc a été crée pour être utilisé avec un bloc de **gestion de puissance dynamique**. Grâce au contrôle des paramètres qui affectent la consommation (fréquence, volume du son, etc.), celle-ci pourrait être optimisée dynamiquement en fonction des conditions actuelles du système et suivant la politique de contrôle instaurée dans le bloc de gestion de puissance. Ce bloc n'a pas été encore implanté mais il est prévu de le faire dans le futur.

La validation de l'approche

L'étude a été menée sur un circuit exemple disposant d'un processeur ARC [ARC], un sous-système audio, une mémoire et plusieurs interfaces. On utilise comme application un décodeur audio MP3. Le coprocesseur a été implanté avec le reste du système sur une carte de prototypage avec des FPGA.

Ils ont estimé la consommation du système pour le décodage audio MP3 à différentes fréquences de travail et pour différents algorithmes. Du point de vue de la consommation, même s'ils ont implanté le système sur une carte de FPGA pour tester l'approche, les macro-modèles utilisés correspondent à l'énergie consommée par événement dans une implantation sur ASIC. Les macro-modèles et ses valeurs en consommation ne sont pas décrits dans l'article.

Les estimations en consommation obtenues de cette façon ont été comparés aux mesures avec un multimètre, que nous supposons prises sur une implémentation en ASIC (ce n'est pas spécifié). L'erreur relative annoncée est inférieure à 5%. Cela donne une très bonne précision des estimations.

Les avantages et les inconvénients

L'estimation de consommation proposée dans cette approche a pour but l'estimation de la consommation d'une application déjà implantée en matériel pendant son exécution. L'objectif est de réduire dynamiquement la consommation d'un système existant. Cela est très différent de l'estimation de la consommation pendant le développement de l'architecture de système, pour la création d'un système basse consommation, en choisissant les composants et la politique de gestion les mieux adaptés. Par contre, les idées développées ici pourraient être utilisées dans les phases de développement du système, en créant un modèle de simulation du coprocesseur avec les macro-modèles de consommation. Cela permettrait l'estimation dans les premières phases de conception utilisant la simulation architecturale de système, pour la définition du système et d'un bloc gestionnaire de puissance.

3.9 Poet : estimation/optimisation de haut niveau pour systèmes embarqués

Poet (Power Optimisation for Embedded sysTems) est un ensemble d'outils et une nouvelle méthode d'estimation et d'optimisation de la consommation dans les SoC embarqués hétérogènes. Il est développé au sein d'un projet européen coordonné par l'institut de recherche OFFIS (Allemagne)

et avec plusieurs partenaires européens universitaires et industriels (Université Polytechnique de Turin, Cefriel, IST, Atmel, Motorola, ChipVision, BullDast) [OFF02]. Son but est de permettre l'analyse et l'optimisation indépendantes et rapides de la puissance, des applications décrites au niveau algorithmique et/ou RTL, destinées à être implantés sur des SoC embarqués complexes. Pour cela, on utilise des modèles et bibliothèques de composants matériels créés à partir des mesures prises à bas niveau.

La méthode d'estimation de la consommation

Les outils développés et intégrés dans le projet *Poet*, comprennent plusieurs niveaux d'abstraction et s'attaquent aux éléments les plus gourmands en énergie du système. Les niveaux visés sont : algorithmique du composant, logiciel embarqué et RTL cycle-précis. *Poet* travaille au niveau algorithmique de composant pour optimiser la consommation de l'algorithme dans sa future implantation matérielle sur une architecture de composant concrète ; au niveau du logiciel embarqué, pour optimiser le logiciel de façon à réduire la consommation des processeurs ; et au niveau RTL cycle-précis, pour améliorer encore plus les optimisations sur les implantations des blocs matériels. Le code ainsi transformé aura combattu les causes principales de dissipation de la puissance, dès les premières phases de la conception des SoCs complexes, comprenant des coprocesseurs dédiés, processeurs, mémoires, communications et interfaces d'entrée/sortie.

L'outil cherche à automatiser le processus d'optimisation dès le début de la conception algorithmique des composants jusqu'au niveau RTL. Les langages de description de l'application sont le langage C et SystemC au niveau algorithmique et le VHDL au niveau RTL. Cela permet la compatibilité avec les outils bas niveau. Les interfaces de passage vers des outils industriels standards de synthèse sont aussi développés, pour permettre la suite automatique de la conception dans le flot.

La figure 3.9 illustre la façon dont les nouveaux outils sont insérés dans le flot de conception. Les boîtes plus foncées représentent les nouveaux outils de *Poet* et les boites moins foncées les prototypes déjà existants appartenant aux projets précédents.

Poet ne propose pas encore d'outil commercial, multi-niveau et intégré, pour la conception d'architectures de systèmes hétérogènes. Par contre, il utilise quelques outils indépendants, la plupart en forme de prototypes académiques, qui cherchent les optimisations en consommation dans des étapes distinctes du flot de conception. Ils sont illustrés dans le tableau 3.1.

Niveau d'abstraction	**R&D**	**Vendeur**
Algorithmique (Orinoco)	OFFIS, ChipVision	ChipVision
Logiciel	Cefriel	BullDAST
RTL cycle-précis (PowerChecker)	Polito, BullDAST	BullDAST

TAB. 3.1 Les outils développés dans le projet *Poet*

Deux des outils sont déjà commercialisés : *PowerChecker* et *Orinoco*. Nous allons les analyser plus en détail dans les paragraphes suivants.

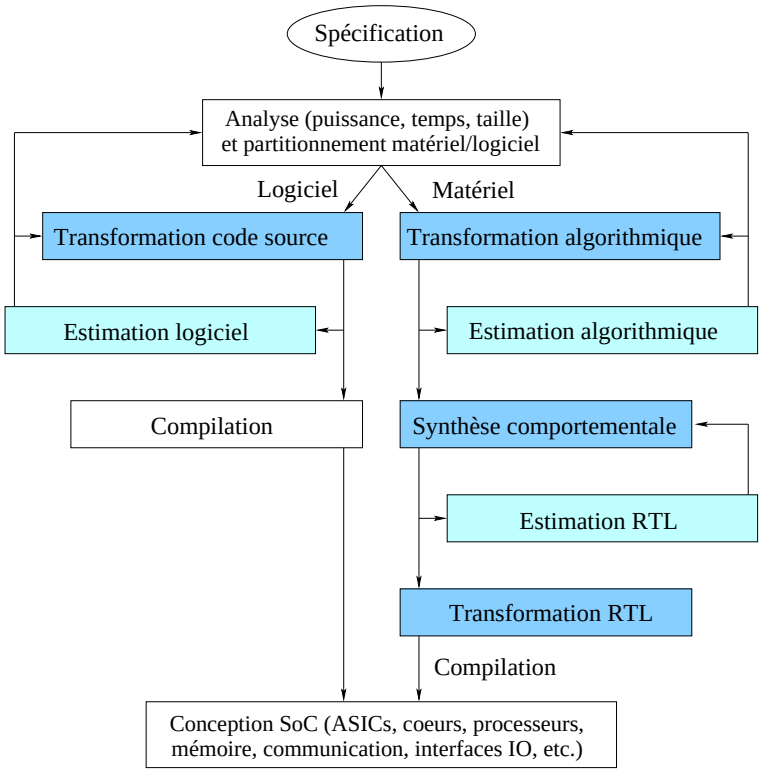

FIG. 3.9 Flot de l'estimation de l'énergie Poet, source [OFF02]

3.9.1 PowerChecker

PowerChecker est un environnement d'estimation et d'optimisation de la consommation sur une description au niveau RTL. Il a été développé à l'Université Polytechnique de Turin (Italie) et commercialisé par la société BullDast [Bul]. Il est basé sur les travaux de Benini et al. [PSZ04] [BDIM03] [BMP03].

Le processus d'estimation et d'optimisation est illustré dans la figure 3.10. L'entrée de l'outil est la description d'un bloc matériel en VHDL niveau RTL avant la synthèse, plus le résultat en activité obtenu par la simulation RTL. L'estimation de la consommation est faite en utilisant des modèles de consommation des opérateurs HDL, des mémoires et de la logique de contrôle. Les modèles sont construits par macro-modélisation (*macromodelling*), ce qui consiste à créer des modèles de consommation des composants, caractérisés en consommation grâce aux mesures prises en bas niveau. La précision de l'estimation dépendra de la précision des modèles, qui normalement sera très bonne.

En plus de l'estimation, PowerChecker permet de faire des optimisations en consommation. Plusieurs techniques peuvent être utilisées. Pour cela l'outil est composé de quatre sous-outils, chacun implantant une technique. Ces sous-outils et les techniques utilisées sont les suivants :

- **CGCap** : Outil implantant la technique de l'**horloge inhibée** (*clock gating*). Cela consiste en la détection des états d'inactivité de groupes de registres du bloc RTL et l'insertion d'un contrôle de l'entrée d'horloge pour ces groupes.
- **LPclock** : La technique s'appelle **planification de l'arbre d'horloge** (*clock tree planning*)

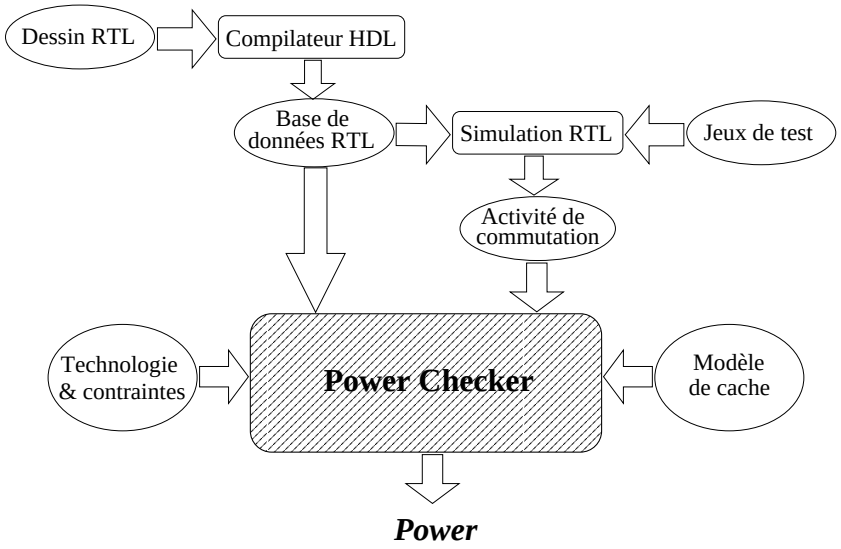

FIG. 3.10 Flot d'estimation et d'optimisation **Power Checker**, source [Bul]

[BDIM03]. Elle consiste à planifier la synthèse d'un arbre d'horloge, avec des horloges inhibées implantées de façon optimale pour réduire la consommation dans l'arbre.

– **MemArt** : Cet outil permet le **partitionnement de la mémoire** (*memory partitioning*) [BMP03]. Cela consiste à analyser les accès mémoire de la description RTL, pour générer des bancs de mémoire de la taille et caractéristiques précis pour chaque type d'accès. Par exemple, les adresses les plus fréquentes sont placées dans les mémoires les plus petites et en conséquence, les moins gourmandes en énergie.
– **CoolBus** : Outil permettant l'implantation de l'**encodage du bus** (*bus encoding*) [Kel97]. Cette technique cherche à réduire l'activité dans le bus, en utilisant des codeurs pour coder les données sur le bus, de façon à réduire le nombre de transitions entre deux données consécutives.

3.9.2 Orinoco

Orinoco est un outil d'estimation et d'optimisation de la consommation au niveau algorithmique développé à l'OFFIS (Allemagne) et commercialisé par la société ChipVision [Chi]. Il permet au concepteur de comparer différents algorithmes implantés sur différentes architectures de composant en respectant les spécifications en consommation. Il est basé dans les travaux de Brandolese [BFSS02] et Ye [YVKI00].

L'outil vise des applications caractérisées par un grand nombre d'accès mémoire et des données et il s'intègre bien dans les flots de conception existants. L'entrée est la spécification écrite en langage C ou SystemC du composant. Les principales tâches qu'il peut accomplir sont :

⇒ Sélection/transformation de l'algorithme,
⇒ Encodage des données de basse consommation,
⇒ Identification des points chauds,
⇒ Placement en mémoire optimisé en consommation,
⇒ Partage des ressources et des décisions d'allocation,
⇒ Sélection des blocs de basse consommation,

⇒ Qualité de service contre puissance.

Orinoco permet l'analyse de toutes les architectures de composant possibles à partir des ressources et des contraintes temporelles spécifiques, et pour des limites en consommation prédéfinis, d'horloge, interconnections, contrôleur et d'autres composants du système. L'outil exécute le code source originaire avec les vrais stimuli en entrée et estime la consommation pour chaque processus. Pendant cette opération, l'outil prend en compte tous les états possibles de l'algorithme et calcule les valeurs de consommation maximum et minimum. De cette façon, l'outil permet de faire une exploration de la description, pour différents algorithmes s'exécutant sur différentes architectures de composant.

Les avantages et désavantages

Cet ensemble d'outils est le projet le plus complet et le plus ambitieux développe à ce jour, pour l'estimation et l'optimisation de la consommation des SoC embarqués complexes décrits en haut niveau. Le but est d'automatiser la conception d'un SoC depuis le niveau spécification algorithmique du système jusqu'au niveau RTL synthétisable des composants matériels et la compilation du code logiciel, permettant l'application des techniques de réduction de la consommation aux différents niveaux. Il sera compatible avec des outils de conception classiques de plus bas niveau.

Les deux outils déjà commercialisés, *PowerChecker* et *Orinoco*, permettent l'estimation et l'optimisation au niveau RTL et algorithmique des composants. Au niveau algorithmique, *Orinoco* permet le choix de l'architecture de composant qui consomme le moins pour un algorithme défini. Une fois le choix fait, le composant sera synthétisé directement du SystemC ou C vers le VHDL en RTL cycle-précis. A ce niveau, *PowerChecker* permettra d'autres optimisations directement sur l'architecture de composant elle-même.

Cependant, le flot n'est pas encore complet. Il manque des outils pour guider le partitionnement matériel-logiciel du système et pour optimiser l'architecture de système choisie. L'estimation et l'optimisation de la puissance consommée par un SoC entier n'a pas donc été prise en compte encore.

En partant de l'algorithme complet de l'application (spécification en figure 3.9), il faudrait disposer d'un outil d'aide au partitionnement de l'application en tâches algorithmiques parallèles. Orinoco pourrait ensuite être appliqué sur chacune des ces tâches pour trouver l'architecture de chaque composant la mieux adaptée. Après cela, on pourrait se servir de la simulation architecturale de système cycle-précis, pour optimiser encore plus la consommation en définissant une stratégie de gestion de la consommation au niveau système (fréquence, tension, etc., des composants) en fonction des entrées, de l'état ou d'autres paramètres. Cela compléterait le flot d'optimisation de la consommation des systèmes embarqués tel qu'il semble être prévu. Malgré cela, ces fonctionnalités ne semblent pas être prévues pour l'instant.

En conclusion, cet ensemble d'outils comprend des optimisations en consommation dans beaucoup d'étapes du flot de conception, depuis le niveau algorithmique du composant jusqu'au niveau RTL synthétisable. L'approche est très prometteuse, mais le travail n'est pas fini, il doit se développer encore plus.

3.10 Conclusion

Nous avons présenté les différentes méthodes et outils proposés d'estimation de la consommation des systèmes embarqués aux niveaux description algorithmique, description architecturale de système et description architecturale de composant. La plupart sont basés en co-simulation matériel-logiciel au niveau instruction (ISS) du processeur [SBM99] [SC01] et au niveau RTL des accélérateurs matériels [HL02] [LRDL00] [Bul], ce qui donne une très bonne précision aux estimations, mais rallentit la simulation. Ils utilisent des modèles de consommation parfois très abstraits des composants du système comme le processeur, la mémoire et quelquefois les coprocesseurs matériels. Les valeurs d'énergie utilisées par composant sont souvent des mesures physiques ou très bas niveau, ce qui fait que la précision soit très élevée et que tous les composants de la consommation : dynamique et statique, soient pris en compte. Cela est surtout vrai quand la méthode permet aussi le calcul de la consommation pendant les périodes inactives [SBM99] [LRDL00] [SC01] [HKSW03].

En général, les outils ne fournissent pas l'évolution temporelle de la consommation, mais la consommation totale à la fin de la simulation, sauf dans l'outil *Ptolemy* [LRDL00] et le coprocesseur *JouleDoc* [HKSW03]. La plupart des méthodes ont été développées dans les milieux académiques, sans avoir encore abouti sur des outils industriels, sauf les travaux du projet *Poet*, donnant les outils Power Checker [Bul] et Orinoco [Chi]. Tous ces travaux ont été faits pour permettre l'évaluation des différentes techniques de réduction de la consommation de haut niveau. Dans la figure 3.11, toutes les approches sont présentées avec les caractéristiques principales de chacun.

Chaque approche couvre un point dans le spectre du flot de conception de haut niveau. Cependant, il manque celle permettant de tester les optimisations en consommation au niveau architecture de système : gestion dynamique fréquence/tension des composants, horloges inhibées sur les composants, etc. Pour cela, il nous faut un outil possédant les caractéristiques suivantes :

- ⇒ Être capable de simuler et d'estimer la consommation de tous les éléments d'un système-on-chip embarqué décrit au niveau architecture de système, avec toutes ses communications fonctionnant de façon concurrentielle et très rapidement.
- ⇒ Pouvoir simuler les nouveaux accélérateurs qui n'ont pas été encore implantés et dont les modèles fonctionnels et de consommation doivent être créés, en étant suffisamment précis pour ne pas nécessiter l'estimation au niveau RTL.
- ⇒ Donner l'évolution dynamique de la consommation, pour permettre le test des techniques dynamiques de réduction de la consommation au niveau architecture de système.

Dans cette thèse nous présentons une nouvelle approche, qui se basant sur quelques des travaux présentés dans ce chapitre, tente d'atteindre les objectifs évoqués précédemment.

APPROCHE	EQUIPE	NIVEAU	PRISE EN COMPTE					PRECISION ANNONCEE	TECHNIQUES DE BASSE CONSOMMATION	FACILITE DE DEVELOPPEMENT DES MODELES CONSOMMATION	RAPIDITE MISE EN OEUVRE ET SIMULATION	MESURE DE L'EVOLUTION CONSOMMATION DANS LE TEMPS
			PROCESSEUR	CACHE	MEM	HW	INTERC.					
AVALANCHE	Princeton	SW:Instruction HW:RTL	X	X	X	X		——	Transf. logiciels Taille cache Partitionnement	Difficile	SW rapide HW moyenne	Non
SIMUNIC	Standford	SW:Instruction	X	X	X		X	5%	Config. mémoires	Facile	Rapide	Non
PTOLEMY	Turin	SW:Instruction HW:RTL	X	X		X	X	5% SW	Communication bus Partitionnement Choix archit. système	Difficile	SW rapide HW lente	Oui
DTSE	IMEC	Système SW/HW: algorithmique	X	X	X		X	——	Opt. algorithmiques Allocation mémoires	——	Lente	Oui
JOULETRACK	MIT	SW:Instruction	X	X				8%	Réduction fréquence et tension	Facile	Rapide	Non
SOFTEXPLORER	LESTER	SW:Instruction	X		X			3%	Transf. logiciels Choix processeur	Moyenne	Rapide	Oui
JOULEDOC	Graz	Système SW/HW: physique	X	X	X	X		5%	Gestion dynamique de la puissance	Moyenne	Rapide	Oui
POWERCHECKER	Turin	HW:RTL				X		——	Horloge inhibé Partition. mémoire Encodage bus	Fournis par l'outil	——	Oui
ORINOCO	OFFIS	Système SW/HW: algorithmique	X	X	X	X		——	Opt. algorithmiques Placement mémoire Choix archit. compos.	Fournis par l'outil	——	Oui
NOTRE APPROCHE	LIP6/PHILIPS	Système SW/HW: cycle−précis	X	X	X	X	X	Discuté en Thèse	Partitionnement Choix archit. système Gestion dynamique,....	Facile	Rapide	Oui

FIG. 3.11 L'état de l'art de l'estimation de la consommation de haut niveau

Chapitre 4

Méthode de modélisation de la consommation au niveau cycle

Sommaire

L'analyse de la consommation d'un système embarqué est complexe parce que nous devons prendre en compte :

- la modélisation de tous les éléments au niveau système,
- l'estimation des paramètres de consommation,
- et la simulation temporelle donnant l'évolution de la consommation.

Comme nous venons de voir dans le chapitre 3, l'analyse de la consommation au **niveau instruction** dans un simulateur de jeu d'instructions, est utile pour les processeurs avec des architectures simples. Pour des processeurs plus complexes où plusieurs instructions peuvent être exécutées en parallèle, cette méthode demande trop de temps de modélisation, devenant alors prohibitif. Pour les processeurs simples, elle peut être exploitée en co-simulation avec un simulateur au **niveau RTL** pour modéliser les composants matériels du système. Les résultats ainsi obtenus seront très précis, mais la simulation RTL est beaucoup trop lente pour les systèmes complexes. En outre, cela demande le développement en VHDL des composants matériels, ce qui limite les modifications possibles de l'architecture de système déterminées lors de la simulation car la conception sera déjà trop avancée. La modélisation de la consommation de tout un système au **niveau cycle** permet de trouver un bon compromis entre la précision, la rapidité dans la modélisation et la vitesse d'estimation.

Les avantages de ce type de modélisation seront expliquées dans la première partie de ce chapitre, pour ensuite décrire le flot d'estimation proposé. Après, nous verrons la façon de modéliser fonctionnellement un système embarqué au niveau cycle, pour ensuite expliquer comment obtenir la consommation en utilisant notre méthode générale d'estimation de la consommation pour ce type de modélisation. La façon de calculer les valeurs de consommation sera expliquée à la fin du chapitre.

4.1 L'estimation de la consommation au niveau cycle

La modélisation de la consommation au niveau cycle cherche à représenter la consommation par cycle de tous les composants d'un système. La consommation modélisée sera surtout la consommation dynamique mais aussi la consommation statique dans les cas où celle-ci deviendra non négligeable. Le but est l'obtention d'une estimation de la consommation globale et locale grâce à la simulation. Pour cela, un simulateur fonctionnel cycle près est utilisé qui simule l'application matériel-logiciel avec les vraies entrées et dans lequel on rajoute des valeurs d'énergie par cycle. Cette simulation permettra d'obtenir des estimations rapides, suffisamment fiables et à partir de modèles construits rapidement.

4.1.1 Les avantages

La **vitesse de la simulation** fonctionnelle au niveau cycle est très élevée [Hom01]. Une simulation en SystemC est environ 200 fois plus rapide qu'une simulation VHDL au niveau portes logiques [SBT00] [JKLa04]. C'est la conséquence du haut degré d'abstraction des modèles et du langage de haut niveau utilisé (C/C++). L'estimation de la consommation en utilisant cette simulation ne ralentira pas notablement la simulation. En conséquence, cette estimation sera aussi beaucoup plus rapide que les estimations aux niveaux plus bas.

La **vitesse dans la modélisation de la consommation** sera élevée grâce au haut niveau d'abstraction des modèles. Elle consistera d'abord à modéliser fonctionnellement le composant au niveau cycle et puis à ajouter les valeurs de consommation. Ces valeurs seront rapides à calculer, à ajouter au modèle et à paramétrer. Il se peut que les modèles fonctionnels des composants les plus utilisés (processeur, mémoire, IP) soient disponibles à l'avance car la simulation fonctionnelle au niveau cycle est une étape nécessaire dans le flot de conception. Dans ce cas, la modélisation de la consommation demandera seulement d'ajouter les valeurs de consommation au modèle et sera donc encore plus rapide à faire.

La **précision du modèle de consommation** de chaque composant dépendra de la précision comportementale du modèle fonctionnel et de la précision des valeurs de consommation associées au modèle. Étant donné que la modélisation au niveau cycle est faite pour développer l'architecture de système et les interfaces entre blocs matériels, le comportement au cycle de l'implantation future sera normalement très proche de celle-ci. La précision des valeurs de consommation dépendra donc de la précision du modèle de consommation disponible. Si l'on dispose d'un modèle très précis (avec des valeurs mesurées au niveau bas), on peut l'intégrer et exploiter cette précision, ce qui nous permettra d'obtenir d'estimations autant précises de celles obtenues aux niveaux les plus bas.

La **précision de l'estimation de la consommation du système** dépendra donc de la précision des modèles de consommation de tous les composants, mais aussi des rapports et communications temporelles entre eux pendant la simulation de l'application. La simulation fonctionnelle au niveau cycle offre un paramètre très important : *le taux d'utilisation* des composants dans le système. Ce paramètre va beaucoup augmenter la précision des estimations parce qu'il donne l'activité réelle de chaque composant pour une application particulière. Il est très difficile de l'obtenir sans utiliser la simulation temporelle. Par exemple, les simulateurs non-temporels au niveau algorithmique simulent le comportement des composants sans comptabiliser les cycles d'attente de chacun, ce qui peut ajouter beaucoup d'erreur aux estimations.

Ce taux d'utilisation est un facteur très important, non seulement parce qu'il augmente notablement la précision, mais aussi parce qu'il permet d'analyser l'évolution dynamique de la consommation. Désormais, on pourra mieux identifier les points chauds de l'architecture de système, faciliter le partitionnement, la localité temporelle des mémoires, les décisions de gel des horloges ou de réduction de la tension et de la fréquence, des composants inactifs sur de longues périodes et l'utilisation d'autres techniques dynamiques de réduction de la consommation.

4.1.2 Le flot d'estimation de la consommation

Le flot d'estimation de la consommation d'un système décrit au niveau cycle près est illustré de façon synthétique dans la figure 4.1. L'estimation de la consommation avec notre approche part de la description parallèle de l'application au niveau algorithmique (C/C++, YAPI, SystemC). Sur cette description, on fait le partitionnement et le choix de l'architecture du système matérielle. Les tâches matérielles sont affectées aux processeurs ou aux accélérateurs matériels, lesquels seront modélisés au cycle près pour la simulation choisie (TSS, SystemC, CASS). Certains modèles pourront être déjà disponibles, comme ceux des processeurs ou des mémoires.

À ce moment, on calcule la consommation par cycle des composants du système qui nous intéressent

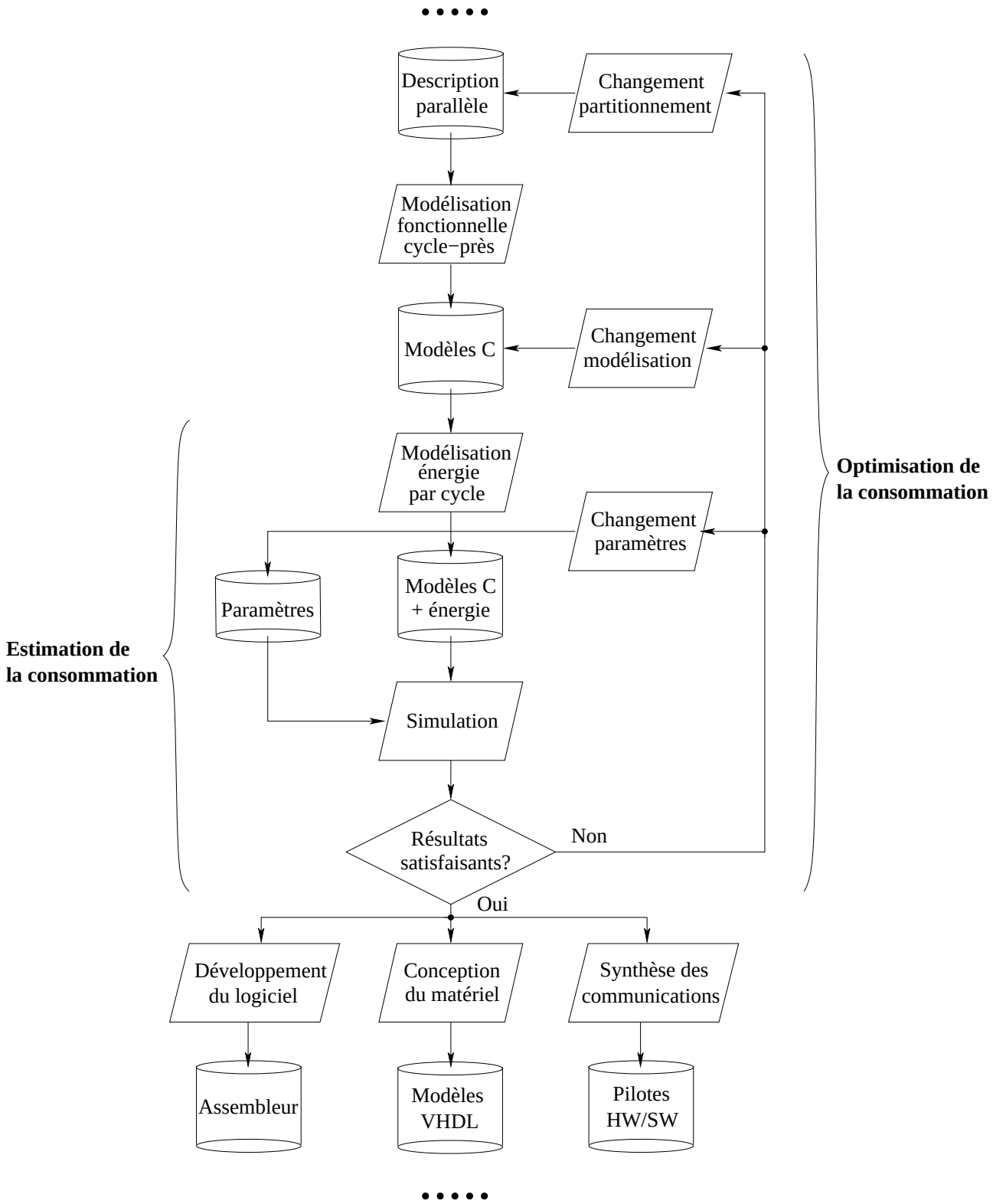

FIG. 4.1 Flot d'estimation de la consommation d'un système au niveau cycle

du point de vue de la consommation et on ajoute ces valeurs dans chaque modèle fonctionnel. Pour ce faire, chaque modèle est instrumentalisé avec des variables et des paramètres qui prendront les valeurs de consommation et les calculs nécessaires à faire par cycle. Ces informations seront utilisées par le simulateur qui aura été préalablement modifié, pour le calcul des valeurs globales de consommation et pour l'affichage des résultats.

À ce stade, on peut procéder à la simulation. Les paramètres fonctionnels et de consommation sont affectés et l'on simule avec les vrais stimuli de l'application en entrée. Pendant la simulation et surtout à la fin, on obtiendra les résultats en performance et en consommation du système. Ces résultats seront analysés en détail pour observer si le système atteint les contraintes temporelles et de consommation fixées à l'avance. Si c'est le cas, la conception du matériel peut commencer. Si ce n'est pas le cas ou si l'on veut améliorer encore plus les résultats, il faut trouver les points critiques et la raison de cette défaillance pour proposer une solution.

Il peut exister plusieurs solutions au même problème. La première et la plus simple, sera la modification des paramètres des modèles et du système ; par exemple la taille des mémoires, la largeur des données, la largeur du bus, la fréquence et la tension de fonctionnement, etc. Si ces changements ne résolvent pas le problème, alors il faudra modifier le modèle lui même. On peut chercher à réduire le nombre de cycles nécessaires à l'exécution d'une tâche, à implanter un mode de fonctionnement de basse consommation, etc. Si après ces changements, les résultats continuent à être insuffisants, il faudra redéfinir l'interface entre les composants ou le partitionnement et le choix de l'architecture de système.

Dans les sections suivantes, nous allons expliquer la façon de modéliser le fonctionnement d'un composant matériel pour la simulation au niveau cycle et puis comment modéliser la consommation d'énergie pour ce type de simulation.

4.2 La modélisation fonctionnelle

La modélisation fonctionnelle au niveau cycle consiste à représenter de façon abstraite tous les composants d'un système complexe, de façon à pouvoir simuler leur comportement et celui de toutes leurs intercommunications cycle par cycle. Cette modélisation est de plus en plus utilisée comme méthode d'évaluation des systèmes embarqués, grâce à sa grande vitesse de simulation, au haut niveau de précision (qui peut être similaire à la précision du niveau portes) et parce qu'elle permet l'exploration d'un système entier permettant de trouver leurs problèmes et les analyser directement de la source [JKLa04].

Plusieurs méthodes de modélisation existent depuis quelques années comme CASS [Hom01], TSS [The98] et plus récemment les environnements en SystemC [Sys], qui se développent très fortement. CASS et TSS sont des simulateurs pilotés par événement (*event driven*) (dont l'événement peut être l'horloge donnant ainsi des simulations au niveau cycle), qui simulent les communications et le fonctionnement des modèles de composants décrits en C/C++. SystemC est une librairie C++ d'un noyau de simulation et des communications de composants, décrits à différents niveaux d'abstraction ; des modèles fonctionnels haut niveau aux modèles détaillés RTL cycle-précis. La méthode de simulation est différente dans chaque approche, mais le principe de modélisation des composants au niveau

cycle-précis reste similaire dans toutes ces approches.

Les systèmes que l'on cherche à modéliser sont constitués de composants comme des processeurs, des coprocesseurs, des mémoires et des interconnexions, comme l'illustre la figure 4.2. Ces composants sont interconnectés par des signaux dont l'interface sera précise au bit près.

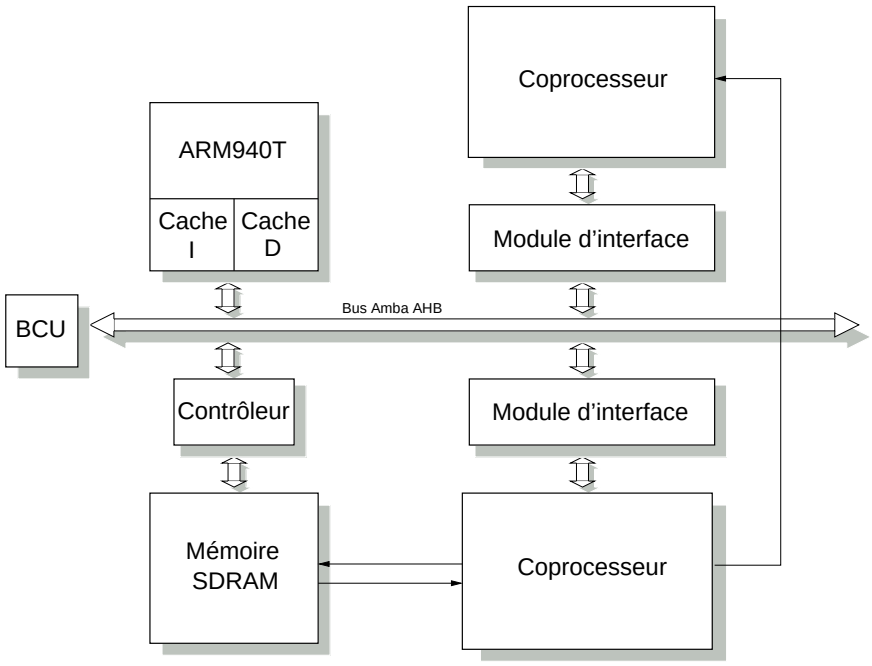

FIG. 4.2 Exemple d'architecture de système

Tous les composants du système sont simulés en parallèle. La façon de modéliser le comportement de chaque composant au niveau cycle est d'utiliser un automate d'état synchrone (ou machine à états) [Hom01]. Dans le paragraphe suivant, nous définirons l'automate d'état. Simuler un système au niveau cycle revient donc à simuler un ensemble d'automates synchrones communicant par des signaux. La figure 4.3 représente les automates du système de la figure 4.2. Plus la modélisation des automates d'état sera proche de l'implantation réelle, plus sa précision fonctionnelle sera bonne.

4.2.1 La définition d'un automate d'état

Un automate d'état est défini par le quintuplet $A = \{I, O, S, t, g\}$, dans lequel :

- I est l'ensemble des valeurs possibles des entrées de l'automate d'état. On note **I** le vecteur de bits représentant l'encodage de I ;
- O est l'ensemble des valeurs possibles des sorties. **O** est le vecteur de bits représentant l'encodage de O ;
- S est l'ensemble des états possible de l'automate. **S** est le vecteur de bits représentant l'encodage de S ;
- t est la fonction de transition. Elle calcule l'état suivant en fonction de l'état courant et des entrées de l'automate. t est telle que $t : I \times S \rightarrow S$. **T** est un vecteur de fonctions booléennes calculant **S** en fonction de **I** et de **S** ;

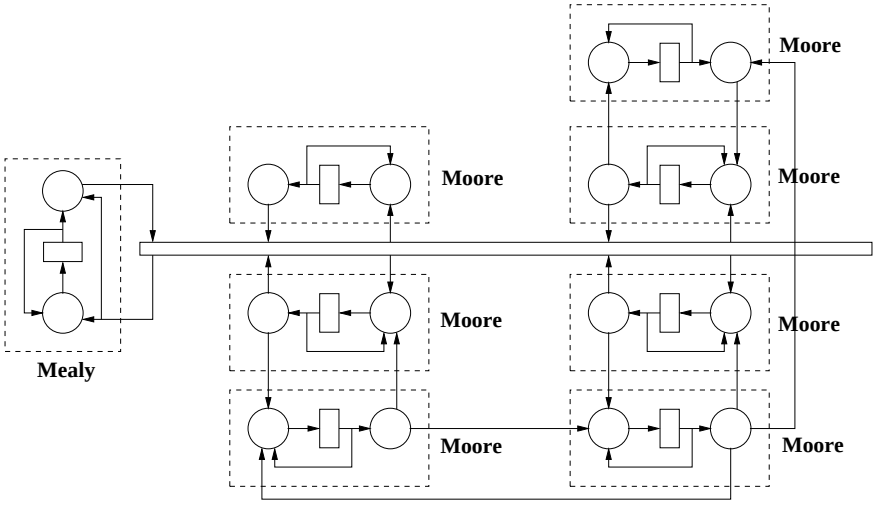

FIG. 4.3 Automates d'un système

– g est la fonction de génération. Elle calcule l'ensemble des sorties de l'automate en fonction de l'état courant et de l'ensemble des entrées. g est tel que $g : I \times S \rightarrow O$. $\mathbf{G}$ est un vecteur de fonctions booléennes calculant $\mathbf{O}$ en fonction de $\mathbf{S}$ dans le cas des automates dits de Moore et en fonction de $\mathbf{I}$ et de $\mathbf{S}$ pour les machines dites de Mealy.

Le comportement du matériel est déterministe et sa modélisation sous forme d'automate d'état impose donc que le comportement de l'automate d'état soit lui même déterministe. Soit $S_i \in S$ l'état courant, on note S_j les successeurs directs de S_i dans le graphe d'états. On appelle $C_{ij} \in T$ la condition de transition portant sur les entrées de l'automate qui permet de passer de l'état S_i à l'état S_j, j pouvant être égal à i si l'automate reste dans l'état S_i durant plusieurs cycles. Les deux conditions nécessaires et suffisantes pour qu'un automate soit déterministe portent sur les conditions de transitions en sortie de l'automate. La première condition est la *complétude*, qui dit que le **ou** logique (noté $+$) des conditions de transitions est égal à 1 :

$$\sum_{\forall j \text{ successeurs } i} C_{ij} = 1.$$

La seconde est l'*orthogonalité*, qui dit que pour une combinaison donnée des entrées, il n'existe qu'un unique successeur. Dit autrement, le **et** logique (noté $\cdot$) des conditions de transitions prises deux à deux est nul :

$$C_{ij} \cdot C_{ik} = 0, \forall j, k \text{ successeurs } i \text{ tels que } j \neq k.$$

Dans la modélisation précise au cycle, la fonction d'évaluation du composant est appelée à chaque front montant de l'horloge. L'état futur, qui peut bien sûr être le même état que l'état courant, est calculé à cet instant. Les calculs du vecteur de sortie $\mathbf{O}$ sont alors effectués, en fonction de l'état nouvellement calculé et éventuellement des entrées.

4.2.2 L'estimation de la consommation

La consommation ayant lieu lors des transitions [1] associées aux changements d'état des registres, nous avons choisi d'associer conceptuellement la prise en compte de la consommation aux changements d'états, c'est à dire lors du franchissement d'une condition de transition dans l'automate de chaque composant.

Dans la modélisation en automate d'états, on considère qu'il y a toujours une transition par cycle, même si l'on reste dans le même état. Cela provoque toujours une consommation due à l'évaluation du nouvel état et à l'exécution des opérations associées à la transition. Si l'on change d'état, on exécutera l'opération associée au nouvel état. Si l'on reste dans le même état on va exécuter à nouveau l'opération correspondante à l'état : lecture, écriture, attente, calcul, etc., mais avec de nouvelles données, ou éventuellement des opérations différentes dans le cas des automates de Mealy. Si l'on arrive à bien identifier l'énergie dissipée par transition de l'automate d'état, on pourra obtenir en simulation des valeurs de la consommation totale et instantanée suffisamment précises pour tous les composants d'un système embarqué.

La puissance versus l'énergie

A ce stade, on doit bien distinguer la consommation de puissance de la consommation d'énergie. Le terme consommation regroupe, en fait, les dissipations de puissance et d'énergie. Mais chacun de ces paramètres est employé pour des études de la consommation bien différentes.

La consommation de puissance est un paramètre important si l'on s'intéresse à la dissipation thermique dans un système. En effet, pour pouvoir dimensionner les circuits de refroidissement, il faut connaître avec précision la puissance dissipée par ce système (surtout pour les applications embarquées dans lesquelles le système de refroidissement doit avoir une surface, une masse et un coût minimal).

L'énergie est un paramètre à prendre en compte si l'on veut étudier la durée de vie des batteries. Ce paramètre est bien sûr lié à la puissance consommée par l'application mais, également au temps d'exécution. Deux applications peuvent avoir la même puissance dissipée mais des consommations d'énergie différentes comme l'illustre l'exemple du tableau 4.1.

Application	Puissance	Temps d'exécution	Energie
Algo 1	2.5 W	10 μs	*25 μJ*
Algo 2	2.5 W	25 μs	*62.5 μJ*

TAB. 4.1 La consommation de puissance et d'énergie

Dans ce tableau, nous observons que l'*algorithme 1* et l'*algorithme 2* consomment exactement la même puissance ; nous aurions donc tendance à dire que les deux algorithmes sont identiques du point de vue de la consommation. Or, les temps d'exécution de ces algorithmes sont sensiblement

[1] Ne pas confondre transition dans l'automate d'état et transition du transistor.

différents puisque le premier est exécuté en 10 μs et le second en 25 μs. Cette différence en temps d'exécution aura une répercussion sur l'énergie consommée par l'algorithme puisque le deuxième consomme 2,5 fois plus d'énergie que le premier à puissance égale ($E = P/f = P \times \Delta T$).

Ce petit exemple nous montre bien que pour les applications embarquées, la consommation de puissance est un paramètre nécessaire mais pas suffisant puisque la durée de vie des batteries dépend de l'énergie consommée. De plus, la modélisation de la consommation au niveau cycle ne peut se faire qu'en modélisant l'énergie consommée par cycle. La puissance sera calculée à partir de l'accumulation de toutes les énergies par cycle pendant la durée d'exécution de l'application.

4.3 La modélisation de l'énergie

La modélisation de l'énergie au niveau cycle demande la création des modèles d'énergie de chaque composant, suivant les caractéristiques particulières de chacun, mais restant tous décrits au même niveau. Le fonctionnement par cycle et par composant doit être analysé pour y associer une valeur d'énergie à chaque transition de l'automate.

En fonctionnement normal le composant réalise une tâche spécifique par cycle déterminée par sa fonctionnalité, les entrées et l'état d'avancement de l'application. Cela provoque une consommation dynamique due aux transitions dans les portes, dans les flipflops et sur la ligne d'horloge, ainsi qu'une consommation statique due aux courants de fuites tout au long du composant. De même, pendant les périodes d'inactivité, le composant va dissiper une énergie dynamique provenant des transitions sur la ligne d'horloge et sur l'entrée d'horloge des flipflops, ainsi qu'une énergie statique provenant des courants de fuites. Si l'entrée d'horloge est coupée pendant ces périodes alors on ne dissipera que l'énergie statique.

Toutes ces consommations dynamiques et statiques doivent être estimées pour tous les composants, pour ensuite calculer l'énergie locale et globale consommée dans le système en utilisant le modèle mathématique présenté ensuite.

4.3.1 Le modèle mathématique

Le modèle mathématique associé à la modélisation de l'énergie au niveau cycle peut être défini à différents échelons. Nous allons descendre de l'échelon le plus élevé au plus bas. Dans le plus haut, nous trouvons l'énergie totale nécessaire pour exécuter une application sur l'architecture du système cible. Si m est le nombre total de composants du système, l'énergie totale est obtenue en accumulant l'énergie totale dissipée par chaque composant, selon l'équation 4.1.

$$E_{totale} = \sum_{i=1}^{m} E_{composant_i} \tag{4.1}$$

Au deuxième échelon, nous considérons l'énergie totale dissipée par composant, ce qui correspond à l'accumulation de l'énergie consommée dans chaque transition de l'automate d'état de chaque composant. Si l'exécution de l'application nécessite n transitions, cette énergie est représentée par

l'équation 4.2. On note que nombre de transitions et des cycles est le même, grâce à la propriété des automates d'état qui dit que l'on a toujours une transition par cycle.

$$E_{composant} = \sum_{j=1}^{n} E_{composant,transition_j} \tag{4.2}$$

Chaque composant du système utilise un ou plusieurs automates d'état pour simuler son fonctionnement. Pour simplifier la notation, dans la suite nous considérerons un seul automate par composant, mais la méthode restera valide dans tous les cas. Les transitions entre les états représentent la fonctionnalité du composant par cycle de travail. Dans chaque transition, une ou plusieurs opérations basiques seront réalisées : lecture, écriture, opération1, opération2, attente, etc. et chaque opération aura une consommation d'énergie dynamique et statique associée. Pour r opérations, cette énergie est représentée par l'équation 4.3.

$$E_{transition} = \sum_{k=1}^{r} E_{composant,operation_k} \tag{4.3}$$

Pour avoir une estimation correcte de la consommation, le plus important est de bien identifier la ou les opérations réalisées dans la transition de l'automate et de leur associer une énergie. L'énergie associée à chaque opération du composant $E_{comp,oper}$, correspond à l'échelon le plus bas de notre modèle. L'ensemble des énergies par opération pour x opérations possibles constitue le modèle d'énergie du bloc matériel. Il est représenté dans la formule 4.4.

$$E_{composant} = \{E_{comp,oper1}, E_{comp,oper2}, ..., E_{comp,operx}\} \tag{4.4}$$

Ainsi, nous définissons l'énergie dissipée par transition, dépendant de l'opération ou des opérations exécutées lors de la transition actuelle de l'automate. Un exemple est illustré pour n transitions dans les équations 4.5.

$$\begin{cases} E_{composant,transition_1} = E_{comp,oper_1} \\ E_{composant,transition_2} = E_{comp,oper_2} + E_{comp,oper_3} \\ ..... \\ E_{composant,transition_n} = E_{comp,oper_x} \end{cases} \tag{4.5}$$

4.3.2 La simulation

Pendant la simulation du système, chaque composant transite par cycle dans le même état ou dans un nouveau selon la fonctionnalité de l'application et les entrées. Dans la figure 4.4, on illustre l'automate d'un composant exemple x.

Suivant cet exemple, la somme des énergies correspondant aux opérations effectuées dans la transition courante de l'automate est calculée, donnant l'énergie instantanée par cycle du module. Le détail du

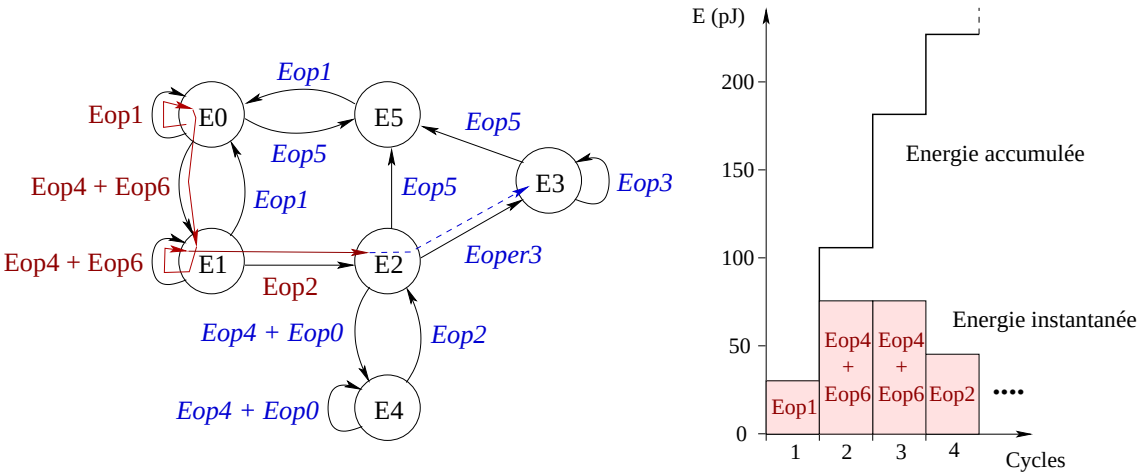

FIG. 4.4 L'automate d'états d'un composant x

calcul est donné par le groupe d'équations 4.6.

$$\begin{cases} E_{comp_x,transition_1} = E_{comp_x,oper_1} \\ E_{comp_x,transition_2} = E_{comp_x,oper_4} + E_{comp,oper_6} \\ E_{comp_x,transition_3} = E_{comp_x,oper_4} + E_{comp,oper_6} \\ E_{comp_x,transition_4} = E_{comp_x,oper_2} \\ \ldots\ldots\ldots \end{cases} \tag{4.6}$$

Cette énergie accumulée donne l'énergie totale par module et de tout le système. De cette façon, on obtient l'évolution de la consommation d'énergie dans le système.

$$E_{comp_x} = \overbrace{E_{comp_x,oper_1}}^{E_{comp_x,transition_1}} + \overbrace{E_{comp_x,oper_4} + E_{comp_x,oper_6}}^{E_{comp_x,transition_2}} + \overbrace{E_{comp_x,oper_4} + E_{comp_x,oper_6}}^{E_{comp_x,transition_3}} + \overbrace{E_{comp_x,oper_2}}^{E_{comp_x,transition_4}} + \ldots \tag{4.7}$$

Si le système est composé de m composants, l'énergie totale dissipée sera la somme des énergies accumulées de chacun, comme le montre l'équation 4.8.

$$E_{totale} = \overbrace{E_{comp_x} + E_{comp_y} + E_{comp_z} + \ldots + E_{comp_m}}^{m \text{ composants}} \tag{4.8}$$

4.3.3 Le cas particulier des macro-états

Il se peut, pour des raisons de simplicité de modélisation, que l'on préfère grouper des états identiques dans un seul macro-état où l'on va exécuter la même opération un nombre de fois déterminé par l'application. La modélisation en automates d'état permet de le faire. Dans ce cas, il peut arriver que dans la même transition de l'automate et en fonction des entrées des opérations différentes puissent être exécutées. Ainsi, la même transition de l'automate aura différentes valeurs d'énergie associées.

Par exemple, dans la figure 4.5, le macro-état appelé *req-data* exécute 32 lectures de données. Les opérations possibles sont la *lecture* et l'*attente* des données, chacune avec une consommation d'énergie associée. Un cycle de lecture peut être suivi ou précédé d'un ou plusieurs cycles d'attente, puisque les données arrivent de l'extérieur à n'importe quel moment. La consommation des cycles de lecture est celle correspondante à la lecture d'un mot de 32 bits, *Erd*. La consommation des cycles d'attente est celle due aux transitions de l'entrée d'horloge sur les registres et les transitions inutiles résiduelles (*glitches*), *Ewt*. Ces deux consommations sont associées aux flèches du macro-état, comme l'illustre la figure 4.5.

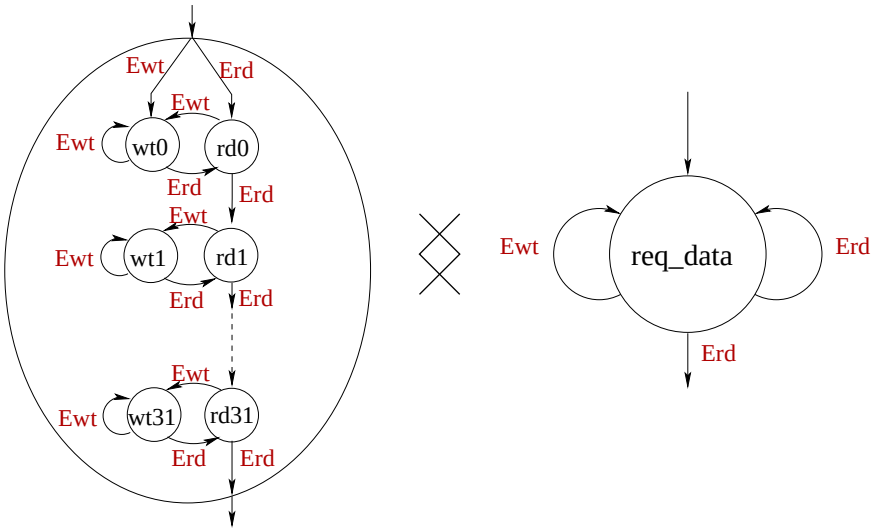

FIG. 4.5 L'énergie par transition dans un macro-état

4.3.4 Les valeurs d'énergie

Les valeurs d'énergie par opération du modèle d'énergie de chaque composant correspondent à l'énergie moyenne dissipée par le composant en exécutant physiquement l'opération ou les opérations associées à la transition de l'automate. Cela inclut la modélisation de la composante dynamique ainsi que statique. La précision d'une telle modélisation dépendra de la précision des valeurs d'énergie ainsi que de la précision du modèle fonctionnel au niveau cycle. De manière générale, la façon dont la consommation d'énergie d'un composant est modélisée peut être envisagée selon différentes méthodes et à différents niveaux.

Macro-modélisation (*macro-modelling*)

Cette méthode consiste à créer une librairie de consommation des composants ou des sous-parties des composants (ALU, registres, mémoires, etc.) avec des valeurs d'énergie ou de courant dynamique et statique consommées par opération et mesurées à très bas niveau (physique, transistors ou portes) [PLG98] [BBB+98] [JKLa04]. Ces valeurs correspondent aux moyennes obtenues après plusieurs exercices de mesure en fonction des entrées et des paramètres spécifiques du bloc. Elles peuvent être

fournies par le constructeur ou bien mesurées par le concepteur s'il dispose de l'environnement de mesure adéquat.

Cette approche convient pour les composants avec des architectures fixes comme les processeurs (RISC, DSP, etc), les mémoires ou les blocs IP, dans lesquels il n'y a pas de changements architecturaux en synthèse ou pendant l'écriture du code VHDL. Toutes les modifications faites en synthèse, qu'elles soient technologiques ou structurelles, impliquent une nouvelle caractérisation de la librairie qui peut être assez coûteuse en temps de développement.

Cette méthode est la plus précise, surtout si les valeurs proviennent des mesures prises directement sur le matériel. Dans ce cas et si le modèle fonctionnel est bien conçu, l'**erreur** peut être inférieure à **5%** (erreur des mesures). Si les valeurs sont prises par un estimateur au niveau portes logiques, l'erreur peut atteindre **15-20%** (erreur de l'estimateur).

Modélisation rapide

Cette méthode consiste à utiliser les équations de la puissance dynamique et statique presentées dans le chapitre 2 pour une modélisation rapide de l'énergie dissipée par opération du composant. Le calcul est fait en fonction des paramètres fonctionnels, structurels et technologiques du composant [LH98] [SBM99].

À la différence de la macro-modélisation, les informations fournies au modèle ne seront pas directement les valeurs d'énergie mesurées mais plutôt des paramètres comme le nombre de portes, l'activité ou la tension. Ces paramètres seront appliqués aux équations insérées dans le modèle lui-même pour calculer l'énergie par opération correspondante. Les modèles seront simples et rapides à faire. Plus on connaît la structure et son comportement mieux on pourra définir les paramètres et en conséquence meilleure sera la précision obtenue. Nonobstant, elle sera toujours inférieure à celle obtenue avec la macro-modélisation. Selon nos expériences, on estime que l'**erreur** sera d'environ **20-40%**. Pour l'améliorer, toutes les valeurs des paramètres pourront être calibrées une fois que la description VHDL sera disponible. Et quand on aura les mesures physiques, on les insérera directement dans le modèle par macro-modélisation.

Cette approche est utile pour modéliser rapidement les nouveaux composants comme les accélérateurs, les blocs IP ou les interconnexions dont on ne dispose pas encore des mesures bas niveau mais on peut quand même estimer quelques paramètres par l'analyse de la complexité du composant. Cette méthode peut être aussi utile pour les nouvelles versions de composants, dont certains paramètres seront connus de la version précédente et d'autres pourront être estimés à partir des modifications structurelles, comportementales ou technologiques de la nouvelle version. Par contre, pour les composants dont l'estimation des paramètres devient trop difficile voir impossible, on utilisera pour les estimer la méthode de l'analyse de l'entropie expliquée dans le point suivant.

Analyse de l'entropie

L'entropie (en théorie de l'information) permet de mesurer la quantité d'information moyenne d'un ensemble d'événements et de mesurer son incertitude (l'occurrence d'un événement peu probable est plus informative que l'occurrence d'un événement probable). Dans notre cas, l'entropie est utilisée

pour estimer le *nombre de portes* et *l'activité* par instruction ou par cycle du composant à partir des événements sur les entrées et les sorties [CCa02] [NN96]. Ces valeurs seront ensuite insérées dans les équations de la modélisation rapide pour le calcul de l'énergie par opération.

Dans cette méthode, les composants sont considérés comme des boîtes noires dont on ne connaît que la fonctionnalité. Leur simulation comportementale permet d'obtenir l'activité des entrées/sorties du composant par cycle. Avec ces valeurs et les équations de l'entropie, on obtient le nombre de noeuds internes du composant et leur activité. Ces valeurs seront ensuite introduites dans les équations de la modélisation rapide pour l'estimation de la consommation dynamique et statique.

Cette approche est utile si l'on ne dispose d'aucune information structurelle du bloc. Par contre, la précision obtenue est très approximative. Par exemple, dans [CCa02], l'erreur des estimations avec l'entropie par rapport aux mesures sur SPICE est de 58%. Dans cet exemple, en insérant l'activité de quelques noeuds internes dans les équations l'erreur est réduite à 37%. Selon nos analyses, on estime que l'**erreur** obtenue pour un composant de type IDCT sera d'environ **40-70%**. Cela peut paraître une très mauvaise précision mais peut servir à évaluer un premier choix d'architecture de système. Nonobstant, on n'utilisera pas cette méthode pour modéliser les composants plus complexes et gourmandes en énergie et on se limitera aux accélérateurs les plus simples.

Dans le tableau 4.2, on récapitule pour chaque type de méthode : les composants concernés, les informations nécessaires et la précision attendue.

Méthode de modélisation	**Composants concernés**	**Informations nécessaires**	**Précision attendue**
Macro-modélisation	Processeurs, mémoires, blocs IP	Mesures très bas niveau	5-20%
Modélisation rapide	Nouveaux blocs IP	Paramètres équations de puissance	20-40%
Analyse de l'entropie	Nouveaux blocs IP	Activités des entrées et sorties	40-70%

TAB. 4.2 Les méthodes de modélisation des valeurs d'énergie

Le **temps de modélisation de l'énergie** par composant dépend de plusieurs facteurs difficiles à estimer à l'avance. Pour cela, il faut prendre en compte :

- La compléxité du composant : processeur RISC, VLIW, mémoire, accélérateur, etc.
- L'expérience du concepteur.
- L'environnement de simulation disponible :
  - le modèle fonctionnel existe-t-il déjà ou il faut le créer ?
  - le système (bus, vecteurs d'entrée, etc.) est-il opérationel ou faut-il aussi le modéliser ?
- Les mesures d'énergie et les paramètres :
  - disposons-nous du circuit physique et d'une plateforme adéquate de mesure ?
  - ou bien, disposons-nous d'une version du composant en VHDL et d'un outil d'estimation niveau portes ?
  - ou bien, pouvons-nous estimer facilement les paramètres des équations ?

Dans le cas de cette thèse, la modélisation d'un composant et la mise en oeuvre dans notre environnement système ont nécessité entre un jour de développement (modèle de l'espion du bus) et un mois (modèle de la mémoire SDRAM DDRC).

4.4 Conclusion

Dans ce chapitre, nous avons présenté les principes de la modélisation de la consommation d'énergie d'un système décrit au niveau cycle. Cette modélisation est appliquée sur un simulateur fonctionnel temporel. Cette approche permet l'estimation de la consommation à haut niveau pour tester les optimisations en consommation au niveau architecture de système : partitionnement, gestion dynamique fréquence/tension, horloges inhibées, etc. Les caractéristiques principales de cette approche sont les suivantes :

- ⇒ il est possible de simuler et d'estimer la consommation de tous les éléments d'un systeme-sur-puce embarqué avec les interconnexions de façon très rapide, simple et suffisamment précise.
- ⇒ il est possible d'estimer avec une précision suffisante la consommation des nouveaux accélérateurs qui n'ont pas été encore implantés.
- ⇒ La modélisation des systèmes très hétérogènes et complexes se fait par une description au niveau automate d'état dans lequel il suffit de définir une valeur d'énergie par cycle pour chaque composant du système.
- ⇒ il est possible d'obtenir l'évolution dynamique de la consommation, pour tester des techniques dynamiques de réduction de la consommation au niveau architecture de système.

Les systèmes que l'on cherche à modéliser sont composés d'éléments avec des fonctionnalités très variées, donc ils auront des modèles d'énergie très différents. Nous avons présenté trois méthodes de modélisation des valeurs d'énergie : macro-modélisation, modélisation rapide et analyse de l'entropie. Dans la modélisation de la consommation des systèmes complexes, ces trois approches seront utilisées conjointement en fonction du composant et des informations disponibles. Et même, si les méthodes prennent en compte des informations à différents niveaux et ont des précisions différentes, cela permettra d'avoir le maximum de précision dans le système à chaque moment, ce qui reste compatible avec l'objectif d'estimation de la consommation d'un système embarqué décrit en haut niveau. En même temps, la précision de chaque modèle pourra être affinée au fur et à mesure que la conception avancera et que davantage d'informations deviendront disponibles.

Nous n'avons pas connaissance de l'existence d'un autre outil comportant de telles caractéristiques. Désormais, notre approche permet de faire une exploration de l'architecture de système complète facilitant la prise de décision avant de commencer l'implantation du système.

Chapitre 5

Les modèles d'énergie des composants

Sommaire

Dans le chapitre précédent nous avons décrit la méthode générale d'estimation de la consommation au niveau cycle. Dans ce chapitre, nous allons décrire de manière détaillée les modèles d'énergie choisis pour chaque composant d'un système embarqué typique : le processeur, les mémoires, les nouveaux accélérateurs matériels et les interconnexions.

Le modèle de consommation d'énergie d'un composant est composé par l'ensemble des valeurs d'énergie dissipées par cycle. Ces valeurs sont affectées aux transitions de son automate d'état. Pendant la simulation, l'automate transite par cycle dans le même ou dans un nouveau état, accumulant ainsi la valeur d'énergie associée à la transition correspondante. L'accumulation des valeurs d'énergie de tous les composants donne la consommation totale du système.

Les valeurs d'énergie sont calculées à partir des modes de fonctionnement et des fonctionnalités de chaque composant et ils correspondent à l'énergie dynamique dissipée par les opérations réalisées dans le cycle et à l'énergie statique dissipée par les courants de fuites. Les valeurs d'énergie par opération sont calculées par macro-modélisation, créant les modèles d'énergie directement à partir des mesures obtenues au bas niveau et par modélisation rapide, à partir des formules qui calculent les valeurs d'énergie par opération à partir de certains paramètres. Les formules et le calcul des paramètres seront faciles à faire pour permettre la modélisation rapide de la consommation.

Nous allons ensuite montrer comment construire les modèles d'énergie des composants les plus utilisés d'un système embarqué typique.

5.1 Le modèle d'énergie du processeur

Le processeur est responsable de la consommation de la partie logicielle du système embarqué. Les systèmes embarqués utilisent des processeurs bien adaptés aux spécifications du système en performance, taille et consommation. Une famille de processeurs très populaire dans le marché du portable pour sa basse consommation est ARM. Dans le cadre de notre étude, nous utilisons un ARM940T (ARM9TDIM avec caches d'instructions et de données). Cependant, le modèle d'énergie proposé peut être adapté à d'autres types de processeurs.

Les modes de fonctionnement

Le modèle d'énergie du processeur fournit l'énergie de chaque opération exécutée par le processeur par cycle et par mode de fonctionnement. Le modèle fonctionnel simplifié de l'automate d'état d'un processeur peut effectuer en gros deux types d'opérations : l'exécution d'une instruction ou l'attente des données ou d'instructions. La consommation de ces deux opérations est différente et elle dépend du mode de fonctionnement du processeur qui peut être :

- **Actif nominal**, exécution des instructions avec les valeurs nominales de tension et fréquence.
- **Actif à tension et fréquence différentes**, exécution avec des valeurs différentes.
- **Inactif nominal**, mode d'attente.
- **Inactif à tension et fréquence différentes**, mode d'attente avec des valeurs différentes.
- **Endormi**, mode d'attente basse consommation.
- **Hibernation**, mode d'attente très basse consommation.

– **Éteint** (*off*), mode avec le processeur éteint.

Les modes de fonctionnement actifs et inactifs ont une consommation d'énergie dynamique et statique associée à chaque transition de l'automate d'état. Par contre, les modes d'attente basse consommation n'ont pas de transitions associées puisque l'horloge dans ces modes est gelée et l'automate arrêté, donc la consommation dynamique est nulle. Par contre, il y aura toujours une consommation statique qui dépendra du temps passé dans le mode. Pour pouvoir prendre en compte cette consommation dans la simulation cycle-précis, il faudra faire tourner l'horloge à une fréquence quelconque prédéfinie et associer une valeur d'énergie statique par cycle de cette fréquence, à la transition de l'horloge.

L'ensemble de toutes les énergies constitue le modèle d'énergie du processeur. Les modes et leur énergie par transition sont représentés dans l'automate illustré dans la figure 5.1. Cette automate est un ensemble de sous-automates, chacun représentant un mode de fonctionnement à une fréquence et tension différente (donc vitesse de l'horloge de l'automate modifiée).

Le passage d'un mode à un autre se produit dynamiquement pendant l'exécution de l'application, et il est contrôlé par le système d'exploitation ou par un bloc externe de gestion dynamique de la puissance, en fonction de la charge instantanée. Au niveau de la modélisation cycle-précis du composant, cela demande d'ajouter une variable de contrôle du passage d'un mode vers l'autre, ainsi que :

– un changement de la fréquence de l'automate,
– la modification de la valeur d'énergie par cycle,
– l'ajout du surcoût en temps ΔT et en énergie de la mise en oeuvre du changement de mode.

Les valeurs d'énergie

Les valeurs de toutes ces énergies sont obtenues par **macro-modélisation** : à partir des mesures directes sur le matériel ou à partir des valeurs données dans les notices techniques. On privilégie pourtant les mesures sur le matériel pour avoir un maximum de précision. Le temps et l'énergie de passage entre modes doivent être aussi mesurés.

Si le modèle fonctionnel cycle-précis est bien fait, l'**erreur** de l'estimation d'énergie sera donc très basse, on l'estime inférieure à **5%** (erreur des mesures). L'**impact de la consommation** du processeur sur le système risque d'être important, on l'estime de **10-50%**. Pour cela, il est souhaité d'avoir une bonne précision dans son modèle.

Regardons plus en détail les caractéristiques de chaque mode de fonctionnement et leurs valeurs d'énergie associées.

5.1.1 Le mode actif nominal

Le mode actif nominal est le mode de fonctionnement normal du processeur pendant l'exécution des instructions, avec les valeurs de tension et de fréquence nominales pour lesquelles le processeur a été conçu. Dans notre modèle d'énergie, nous considérons qu'en fonctionnement normal le processeur ARM consomme une énergie moyenne dynamique et statique par instruction exécutée.

Nous avons vu dans le chapitre 3, que la consommation du processeur **StrongARM SA-1100** est homogène pour l'ensemble des instructions. Le StrongARM SA-1100 est une implantation d'Intel

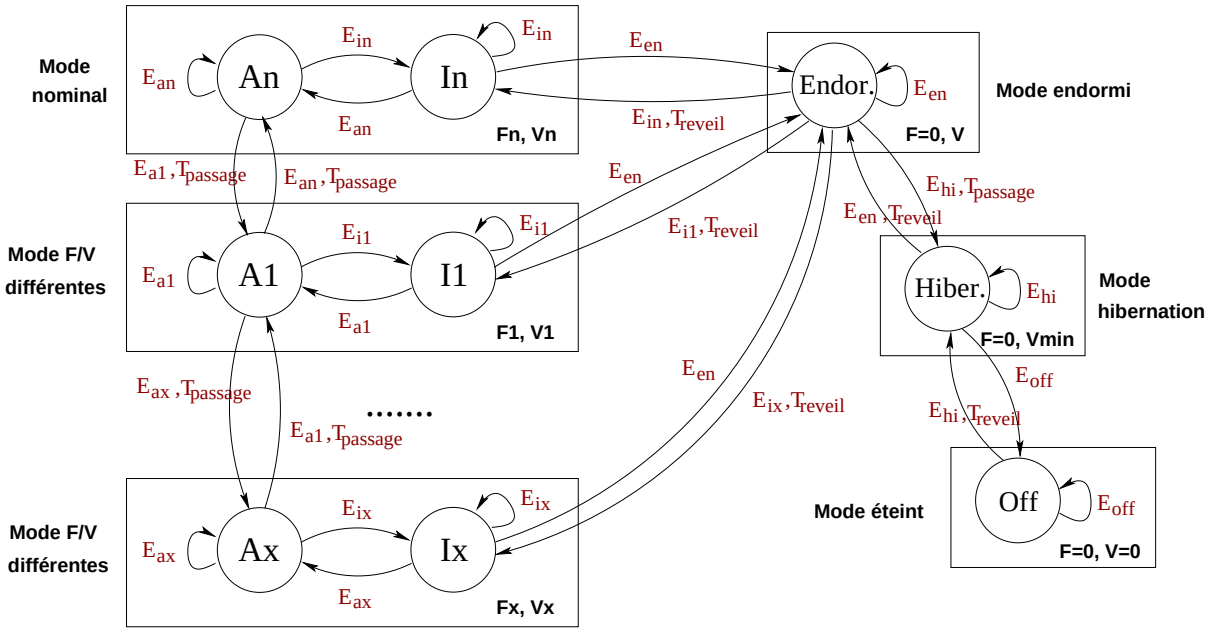

FIG. 5.1 Les modes de fonctionnement et les états du processeur

du processeur ARM710a avec caches [Cor00]. Selon les études de Sinha [SC01], la variation maximale de courant entre instructions dans ce processeur, pendant l'exécution d'un programme est de seulement 8%. On peut généraliser ce comportement pour d'autres implantations de la famille de l'ARM7 et même de la famille de l'ARM9 puisque l'architecture du processeur présente les mêmes caractéristiques [Adv98]. D'ailleurs, d'autres processeurs embarqués avec des architectures simples présentent ce même comportement, comme le **Hitachi SH-4** [SC01]. Par conséquent, pour les architectures de processeurs RISC simples type ARM, un modèle de consommation considérant seulement le courant moyen par instruction, s'avère suffisant.

Dans notre modèle pour l'ARM, si le courant par instruction est considéré constant, nous pouvons dire que l'énergie dissipée par cycle dans une description cycle-précis, est aussi constante pendant les cycles d'exécution des instructions. Cette énergie est donnée dans les notices techniques, en mW par MHz ou nJ, pour des valeurs de tension V_{nom} et fréquence f_{nom} nominales. Dans le tableau 5.1 on montre les valeurs de tension, fréquence, puissance et énergie nominales des processeurs ARM7 et ARM9. La puissance est obtenue à partir de l'énergie et de la fréquence ($P = E \times f$). La valeur d'énergie correspond à l'énergie consommée par cycle dans les cycles actifs pour ces deux processeurs.

Processeur	Vdd (V)	f (MHz)	P_{active} (mW)	$\mathbf{E_{active}}$ **(nJ)**
SA-1100 (ARM7)	1.5	200	400	**2**
ARM940T	1.8	160	136	**0.85**

TAB. 5.1 La consommation nominale

Les autres processeurs MIPS, DSP

Pour d'autres processeurs, où la consommation par cycle n'est pas si homogène, un modèle plus complet doit être créé. Celui-ci comprendra la consommation de toutes les instructions ou de groupes d'instructions par cycle d'exécution [TMW94]. Pendant la simulation, l'instruction ou les instructions exécutées par cycle seront détectées et on accumulera les énergies correspondantes.

Un tel modèle a été développé par Philips pour une implantation du processeur MIPS, le **MIPS PR1900** [SBT00]. Dans ce processeur, l'énergie dissipée par cycle par les instructions a été mesuré de 1 à 9 nJ, mais pour la plus part d'instructions n'est que de 2 à 4 nJ. Ces différences de consommation entre instructions par rapport à l'ARM proviennent d'une implantation architecturale interne du MIPS très différente de celle de l'ARM. Les instructions ont été groupés selon leur consommation et l'on a mesuré les effets d'inter-instruction, c'est à dire, la consommation additionnelle provenant du changement d'instruction [TMW94]. Les mesures ont été prises sur une implantation du processeur au niveau portes avec l'outil d'estimation *Diesel* [Phi01a], et ensuite intégrées dans le simulateur système cycle-précis *TSS*. De cette façon, on dispose d'un modèle de consommation au niveau cycle de ce processeur qui nous permet de réaliser des estimations au niveau architecture de système. L'estimation au niveau cycle s'avère 200 fois plus rapide que l'estimation au niveau portes et avec une réduction de la précision de seulement 1.4%.

Les architectures plus complexes comme les processeurs de traitement de signal type **DSP**, présentent des variations de consommation par cycle encore plus importantes, dus à la possibilité d'exécution de plusieurs instructions en parallèle (architectures VLIW). Dans ce cas, et pour avoir une précision acceptable, une modélisation détaillée des opérations exécutées par cycle s'avère nécessaire. Si cela est trop coûteux en termes de temps de modélisation, les instructions peuvent être groupées selon leur consommation ou bien une analyse fonctionnelle peut être envisagée comme celle proposée par J. Laurent [LJSM04], présentée dans le chapitre 3. Cette analyse permettrait de calculer de façon statique la consommation d'énergie totale d'un algorithme exécuté sur le processeur VLIW proposé. Cette valeur devrait être ensuite divisée par le nombre total de cycles nécessaires à l'exécution, donnant ainsi l'énergie moyenne dissipée par cycle. Cette valeur constituerait le modèle d'énergie du processeur à intégrer dans le simulateur cycle près. Tout changement sur l'algorithme impliquerait un nouveau calcul. Les cycles d'attente du processeur dus aux accès mémoire ou au matériel dédié devraient être aussi modélisés.

5.1.2 Le mode actif à tension et fréquence différentes

Tous les processeurs peuvent fonctionner à différentes valeurs de tension et l'énergie dissipée par cycle varie selon la tension appliquée [PLS00] [PBB00] [SBM99]. À une tension inférieure, les transistors commutent plus lentement donc la fréquence maximale de travail diminuera aussi avec la tension.

Quelques exemples des valeurs de tension et fréquence possibles pour les processeurs StrongARM 1100, ARM8 et ARM940T sont montrés ensuite.

Par exemple, le **StrongARM 1100** a été conçu pour fonctionner à 1.5 V à une fréquence maximale de 200 MHz, mais les expérimentations de Pouwelse à l'Université de Delft (Pays Bas) [PLS00]

démontrent qu'il peut travailler aussi entre 0.79 et 1.65 V, avec des fréquences entre 59 et 251 MHz.

Le tableau ci-dessous nous montre les valeurs de consommation de puissance et d'énergie par cycle pour différentes valeurs de tension et fréquence, selon les mesures de Pouwelse.

Vdd (V)	f (MHz)	P_{active} (mW)	**E_{active} (nJ)**
1.65	251	696.7	**2.78**
1.5	59	105.8	**1.79**
0.79	59	33.1	**0.56**

TAB. 5.2 La consommation du processeur StrongARM 1100

Le surcoût en temps pour un changement tension/fréquence est de 140 μs, qui correspond au temps nécessaire pour stabiliser le signal d'horloge à l'intérieur du processeur à une nouvelle fréquence. Il est indépendant de la fréquence. Il n'y a pas de surcoût en énergie car ce processeur continue à travailler pendant l'ajustement tension/fréquence.

Dans son article [PLS00], Pouwelse nous fait noter que les changements fréquents de fréquence/tension n'ajoutent pas un surcoût temporel important à l'application. Par exemple, un *video player* MPEG fonctionnant à 30 images par seconde et dont l'ajustement de la vitesse se fera une fois par image, aura un surcoût temporel inférieur à 1%.

Un autre exemple est le processeur **ARM8**, conçu à l'Université de Berkeley [PBB00] spécialement pour utiliser la technique de l'échelonnage de la tension (*Voltage Scaling*). Il a 16KB de cache, sa tension et sa fréquence peuvent varier dynamiquement entre 1.2 et 3.8V dans un temps inférieur à 70 μs, donnant une performance de 6-85 MIPS avec une consommation d'énergie de 0.54-5.6 mW/MIPS. La variation de fréquence est contrôlée par le système d'exploitation qui a été enrichi avec un "ordonnanceur de tension" (*voltage scheduler*) qui examine l'état du système et communique la vitesse souhaitée au matériel. Le matériel alors ajuste sa fréquence et sa tension. Le tableau 5.3 nous montre les valeurs de consommation du processeur ARM8 avec cache.

Vdd (V)	f (MHz)	P_{active} (mW)	**E_{active} (nJ)**
3.3	100	220	**2.2**
1.1	10	1.8	**0.18**

TAB. 5.3 La consommation du processeur ARM8 plus cache

Le surcoût en temps pour un changement tension/fréquence est de 25 μs. Il n'y a pas de surcoût en énergie car ce processeur continue à travailler pendant l'ajustement tension/fréquence.

Ces deux exemples nous montrent que le processeur peut effectivement fonctionner à différentes valeurs de tension et de fréquence selon les besoins de traitement, sans ajouter un surcoût en temps et en énergie considérables et diminuant ainsi notablement la consommation totale.

En général, pour un processeur quelconque, les valeurs nominales d'énergie et de tension sont fournies par le constructeur. Néanmoins, on peut ne pas disposer de mesures de consommation à tensions différentes effectuées directement sur le matériel. Dans ce cas, l'énergie moyenne par cycle pour une

tension donnée peut être déduite à partir des valeurs nominales. Pour ce faire on calcule d'abord la capacité équivalente du circuit actif avec l'équation 5.1.

$$C_{proc} = \frac{E_{nom}}{{V_{nom}}^2} \tag{5.1}$$

À partir de la capacité, l'énergie par cycle à une tension V_2 différente peut être calculée avec l'équation 5.2. La fréquence de travail maximale f_2 à une tension V_2 doit être obtenue en expérimentation.

$$E_{V_2,f_2} = C_{proc} \times {V_2}^2 \tag{5.2}$$

À partir des valeurs réelles, les capacités équivalentes à différentes tensions ont été calculées grâce à l'équation 5.1 pour le processeur StrongARM et ARM8. Ils sont montrés dans le tableau 5.4. On observe que les valeurs de capacité varient de 20 %. Ce pourcentage représente l'erreur possible de modélisation d'énergie à partir des équations 5.1 et 5.2.

Processeur	Vdd (V)	C_{active} (nF)
StrongARM 1100	1.65	1.02
”	1.5	0.80
”	0.79	0.90
ARM8	3.3	0.20
”	1.1	0.15

TAB. 5.4 Les capacités équivalentes

Nous ne disposons pas des mesures physiques réelles du processeur **ARM940T** (ARM9TDIM avec caches d'instruction et données), nous avons donc calculé l'énergie par cycle à tensions et fréquences différentes, grâce aux équations 5.1 et 5.2 et les valeurs nominales trouvés dans les notices techniques (1.8V, 160 MHz et 0.85 nJ). Les nouvelles valeurs calculées sont montrées dans le tableau 5.5. Le surcoût en temps du changement fréquence/tension n'est pas disponible non plus, mais on considère qu'il doit être similaire à celui du StrongARM de 140 μs.

Vdd (V)	f (MHz)	P_{active} (mW)	**E_{active} (nJ)**	C_{active} (nF)
1.8	160	136	**0.85**	0.26
1.8	10	8.5	**0.85**	0.26
1.1	10	3.2	**0.32**	0.26

TAB. 5.5 La consommation du processeur ARM940T

5.1.3 Le mode inactif

Le mode inactif (*idle*) correspond aux cycles d'attente dus aux défauts de cache ou à l'attente des données venant directement de l'extérieur [PBB00] [Sim01]. Le passage vers ce mode est généré

automatiquement pendant l'exécution du programme et le coût d'entrée ou de sortie est de seulement un cycle **ARM8**.

Le courant dans le processeur diminue pendant ces cycles d'attente parce que le processeur reste dans un état de non exécution. Il s'arrête et attend les données ou les instructions venant du cache ou de l'extérieur. Cette période peut donc durer un certain temps pendant lequel l'exécution des instructions est bloquée et, en conséquence, la consommation dynamique du processeur est réduite. Cependant, l'horloge fonctionne toujours donc l'arbre d'horloge et les entrées d'horloge des registres consomment une énergie dynamique par cycle non négligeable qu'il faut comptabiliser. Il y aura aussi une consommation statique à prendre en compte, surtout dans les technologies en dessous de $0.13\mu m$. Ces cycles seront donc des cycles inactifs avec une consommation d'énergie propre $E_{inactif}$.

La consommation dans le mode inactif à tension nominale d'un processeur StrongARM 1100 a été mesurée à 0.85 nJ [SBM99]. À partir de cette valeur, on calcule la valeur pour l'**ARM940T** par extrapolation avec les valeurs d'énergie en mode actif E_{actif} (on ne dispose pas de mesures physiques) et l'on obtient une énergie en mode inactif à tension nominale $\mathbf{E}_{inactif}$ égale à **0.36 nJ**. Ces énergies changent avec la tension de la même façon que les énergies en mode actif. Les valeurs d'énergie en mode inactif à tensions différentes peuvent être calculées à partir des équations 5.1 et 5.2, en substituant E_{nom} par $E_{inactif}$. Par exemple, à une tension de 1.1 V, on obtient une énergie $\mathbf{E}_{inactif,2}$ égale à **0.14 nJ**.

5.1.4 Le mode endormi

Le mode endormi (*sleep ou standby*) est un état de basse consommation qui correspond aux périodes d'attente où les horloges sont bloquées (*clock gating*) pour diminuer encore plus la consommation [Adv98]. L'énergie dissipée $E_{endormi}$, correspond aux courants de fuites et est normalement très faible, sauf dans les circuits fondus à technologies fortement submicroniques.

Pour bénéficier de ce mode, un contrôle extérieur s'impose par un module de gestion de la puissance. En analysant l'application, il se peut que des longues périodes d'inactivité du processeur soient détectées, par exemple, quand le processeur attend des données venant des accélérateurs qui effectuent de longs traitements. Dans ce cas, le module de gestion peut envoyer au processeur un signal le faisant entrer dans l'état endormi où les horloges seront gelées et la consommation sera minimale.

Le processeur ARM rentre dans le mode endormi de façon instantanée quand on écrit dans le registre 7. Ensuite, il y reste jusqu'à l'arrivée d'une interruption qui le réveille. Le temps de réveil T_{reveil} pour le strongARM 1100 fonctionnant à 3.686 MHz est de **10 ms** [Cor00], ce qui signifie environ 36000 cycles en plus et avec leur consommation ajoutée. Il faut donc bien analyser le passage en mode endormi à chaque fois pour être vraiment sûr de gagner en consommation sans trop perdre en performance.

Notre approche de simulation cycle-précis permet la modélisation de ce mode de fonctionnement ainsi que de son surcoût en temps de réveil. Cette évaluation est très importante dans la définition d'une architecture de système, puisqu'elle permet de bien définir les temps d'attente et d'exécution des différents composants matériels fonctionnant en parallèle, de façon à optimiser non seulement la performance, mais aussi la consommation du système entier.

La consommation par cycle dans ce mode est calculé à partir des courants de fuites I_{fuite}, de la tension V et de la durée du cycle ΔT selon l'équation 5.3.

$$E = I_{fuite} \times V \times \Delta T \tag{5.3}$$

La consommation dans le mode endormi à tension nominale d'un StrongARM 1100 est de 0.5 pJ [Sim01] (presque 2000 fois inférieure à la consommation en mode inactif). À partir de cette valeur, on calcule par extrapolation la valeur pour l'**ARM940T** et l'on obtient une énergie en mode endormi $\mathbf{E}_e$ égale à **0.2 pJ**.

Cette consommation par cycle est très faible, mais au fur et à mesure de l'apparition des technologies submicroniques, elle risque de devenir importante dû à la consommation statique, surtout dans les systèmes avec des longues périodes endormies. Pour cela, un nouveau mode de fonctionnement à vu le jour, le mode en hibernation.

5.1.5 Le mode en hibernation

Le mode en hibernation (*drowsy*) est un mode inactif qui cherche à réduire la consommation statique due aux courants de fuites, en réduisant la tension d'alimentation du circuit. C'est une nouvelle technique développée pour les technologies fortement submicroniques, qui profite du fait que l'on peut réduire beaucoup les courants de fuites en diminuant la tension appliqué sur le circuit en mode inactif. La tension peut être réduite jusqu'au minimum permettant de maintenir l'état des registres. Cette énergie est calculée grâce à l'équation 5.3 et les nouvelles valeurs de courant de fuites et de tension.

La réduction de la tension est utilisée normalement pour diminuer la consommation dynamique, comme on l'a expliqué dans le chapitre 2. Par contre, Kim et al. [KFBM04] utilisent cette technique pour réduire la consommation statique d'une mémoire cache, en mettant les lignes inactives dans un état d'hibernation à tension réduite. De cette façon, l'énergie dissipée par les courants de fuites diminue d'entre 60% et 75%.

En principe, cette technique peut être utilisée dans n'importe quel composant pendant ses états inactifs. L'entrée et la sortie de ce mode seront contrôlées depuis l'extérieur. Cependant, nous n'avons pas trouvé dans la littérature des exemples d'implantation de cette technique sur les processeurs. En conséquence, nous ne disposons pas de détails sur des valeurs de consommation en mode d'hibernation, ni du surcoût en temps de passage et de réveil. Malgré cela, ce mode de fonctionnement reste modélisable avec notre approche de simulation cycle-précis ainsi que son surcoût en temps et en énergie de passage et de réveil, dans le cas qu'il soit utilisé dans le futur.

5.1.6 Le mode éteint

Dans l'état éteint l'alimentation du processeur est coupée donc la consommation est nulle. Le passage vers cet état est contrôlé extérieurement, mais la sauvegarde de l'information et surtout le redémarrage prennent tellement de temps que ce n'est pas un mode adapté à la réduction de la consommation pour un processeur en fonctionnement.

5.2 Le modèle d'énergie de la mémoire

La mémoire est responsable d'une grande partie de la consommation des systèmes embarqués, surtout dans ceux de traitements multimédia, due à la grande quantité des données transitant entre mémoire et processeurs ou coprocesseurs.

Il existe plusieurs types de mémoires, chacun avec un fonctionnement et une consommation différentes. Nous allons décrire les modèles d'énergie développés dans le cadre de cette thèse pour les mémoires du type RAM/SRAM/FIFO/ROM et celui des DRAM/SDRAM.

Les valeurs d'énergie

Les valeurs d'énergie par opération de la mémoire sont obtenues par **macro-modélisation** : à partir des mesures directes sur le matériel ou à partir des valeurs données dans les notices techniques. On privilégie pourtant les mesures sur le matériel pour avoir un maximum de précision.

Si le modèle fonctionnel cycle-précis est bien fait, l'**erreur** de l'estimation d'énergie sera donc très basse, on l'estime inférieure à **5%** (erreur des mesures). L'**impact de la consommation** de la mémoire sur le système est certainement très important, on l'estime de **50-90%**. Pour cela, il est important d'avoir une très bonne précision dans son modèle.

5.2.1 Le modèle d'énergie de la SRAM

Les mémoires du type SRAM/FIFO/CACHE ou ROM/PROM/FLASH, ont toutes le même comportement du point de vue de la consommation. Elles dissipent toutes une énergie moyenne par cycle qui dépend du type d'accès : écriture, lecture ou pas d'accès, et non pas de la donnée traitée à ce niveau. On va donc utiliser le même modèle d'énergie pour toutes elles dont la seule différence sera les valeurs de consommation qui en conséquence deviendront des paramètres du modèle.

Nous distinguons trois types d'accès différents : les accès en lecture, les accès en écriture et quand il n'y a pas d'accès et le dispositif se trouve donc inactif. Une grande partie de ces mémoires disposent aussi d'un état endormi de basse consommation (*standby*), où l'énergie dissipée est minimale. Un mode en hibernation peut être aussi disponible, pour tout le dispositif ou bien, pour les parties qui ne sont pas utilisées très souvent [KFBM04].

Les modes de fonctionnement de la mémoire du type SRAM sont les suivants :

- **Lecture**, lecture de données de la mémoire.
- **Ecriture**, écriture des données dans la mémoire.
- **Inactif**, mode d'attente.
- **Endormi**, mode d'attente basse consommation.
- **Hibernation**, mode d'attente très basse consommation.

Chaque cycle passé dans un de ces modes de fonctionnement aura une dissipation d'énergie dynamique et statique associée à la transition de l'automate correspondante. Les modes endormi et en hibernation n'auront pas de transitions proprement dite, puisque dans ces modes l'horloge est gelée

et l'automate arrêté, donc la dissipation dynamique sera nulle. Par contre, ils auront toujours une dissipation statique qui sera prise en compte par la transition de l'horloge en dehors de l'automate.

Ces modes et leurs énergies par transition sont illustrés dans la figure 5.2. Dans le mode nominal, on trouve les états de lecture, écriture et inactif, à une fréquence et tension nominales. Le mode endormi correspond à l'état avec l'horloge gelée, ainsi que dans le mode d'hibernation, où en plus de cela, la tension est réduite.

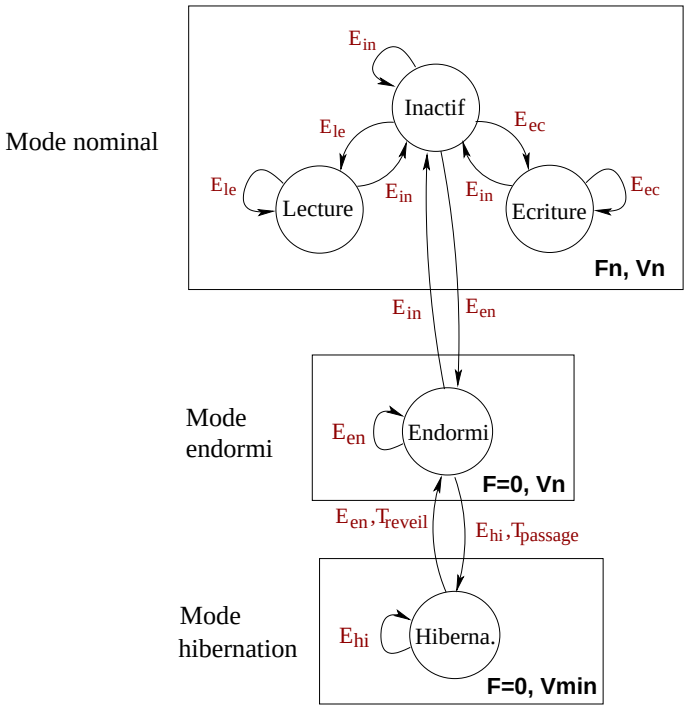

FIG. 5.2 L'automate d'état de la mémoire RAM

Le passage d'un mode à un autre se produit dynamiquement pendant l'exécution de l'application et il est contrôlé par le système d'exploitation ou par un bloc externe de gestion dynamique de la puissance en fonction de la charge instantanée. Au niveau de la modélisation cycle-précis de la mémoire, cela demande d'ajouter une variable de contrôle du passage d'un mode vers l'autre, ainsi que :

- l'arrêt ou démarrage de l'horloge,
- la modification de la valeur d'énergie par cycle,
- l'ajout du surcoût en temps ΔT et en énergie de la mise en oeuvre du changement de mode.

Le passage au **mode endormi** se produit instantanément par la désactivation de l'entrée *clock enable (CKE), chip enable (CE) ou chip select (CS)* depuis l'extérieur. Le temps de réveil à partir de cette état est de seulement un cycle mémoire (SRAM Samsung K6F3216R6M), donc normalement il est utilisée chaque fois que la mémoire n'est pas accédée.

Le **mode en hibernation** fait rentrer en état de très basse consommation les lignes de la mémoire très peu utilisées [KFBM04]. La technique consiste à réduire la tension de la cellule mémoire au minimum possible à ne pas perdre la donnée stockée. Pour ce faire, toutes les cellules sont alimentées

à deux tensions (nominal et basse tension) et c'est une entrée de contrôle (contrôlée extérieurement) qui décide laquelle des deux tension alimente la cellule à chaque moment. L'entrée/sortie dans ce mode se produit de façon presque instantanée par activation/désactivation de cette entrée de contrôle et le temps de passage nécessaire est de l'ordre de seulement d'un ou deux cycles. C'est pourquoi, le surcoût temporel ne sera pas critique pour la performance du système si le contrôle est bien fait. Dans [KFBM04], plusieurs politiques de contrôle ont été testées sur une mémoire cache et dans le pire cas, la performance temporelle du système est réduite de seulement 2.8%, pour des réductions en puissance statique d'entrée 60% et 75%.

Ce mode en hibernation est parfaitement modélisable avec notre approche d'estimation au niveau cycle-précis, ce qui permettra de tester différentes politiques de gestion dynamique du passage pour chaque application particulière.

Les **valeurs de courant** par mode se trouvent dans les notices techniques de chaque mémoire. Principalement, on trouve ces valeurs pour deux modes : le mode nominal ou *operating*, $I_{nominal}$ (I_{cc}) et le mode endormi ou *standby*, $I_{endormi}$ (I_{sb}). L'énergie pourra être facilement calculée à partir de l'équation 5.4 et les valeurs nominales de fréquence et de tension qui se trouvent aussi dans les notices.

$$E = \frac{I \times V}{f} \tag{5.4}$$

Dans ce cas, l'énergie nominale correspond à l'énergie moyenne dissipée dans les états de lecture et écriture. Dans les états inactifs, normalement la consommation diminue puisqu'on n'accède pas aux cellules de la mémoire, sauf si les adresses en entrée continuent à commuter en dissipant la consommation nominale (dans les notices techniques, cette valeur est rarement donnée). Le mode endormi correspond à l'état endormi de basse consommation. Pour ce type de mémoires, et pour diminuer la consommation, l'application doit positionner la mémoire dans ce dernier état chaque fois qu'on n'y accède pas. Le mode en hibernation n'est pas encore implanté dans les mémoires commercialisées, donc on n'a pas des détails sur la consommation. Quelques exemples des valeurs sont montrés dans le tableau 5.6.

Mémoire	Taille	Tension	Fréquence	$I_{nominal}$	$I_{endormi}$	$\mathbf{E}_{nominal}$	$\mathbf{E}_{endormi}$
SRAM Samsung (K6F3216R6M)	2M x 16 bit	1.8 V	14 MHz	25 mA	40 μA	**3.21 nJ**	**5.14 pJ**
PROM Atmel (AT27BV010)	128K x 8 bit	3.6 V	5 MHz	8 mA	20 μA	**5.76 nJ**	**14.4 pJ**
FLASH Nec (uPD29F032)	4M x 8 bit	4 V	5 MHz	10 mA	0.2 μA	**8 nJ**	**0.16 pJ**

TAB. 5.6 La consommation des mémoire SRAM, PROM et FLASH

5.2.2 Le modèle d'énergie de la DRAM/SDRAM

Les mémoires DRAM ou SDRAM sont plus difficiles à modéliser puisqu'elles ont plus d'états et de fonctionnalités. Leur modèle d'énergie cherchera à associer une valeur d'énergie par cycle à chacune

de ces différentes fonctionnalités. A ce niveau, cette valeur est aussi indépendante de la donnée à traiter.

Les mémoires DRAM/SDRAM sont des grandes structures qui utilisent des bancs divisés en rangées pour stocker l'information. Un accès mémoire provoque plusieurs opérations. Au début, la mémoire se trouvera dans un état de précharge ou dans un mode de basse consommation. Dès que l'adresse arrive, on effectuera le décodage de l'adresse, l'activation du banc et de la rangée correspondant à l'adresse et le transfert de toute cette rangée dans l'amplificateur de lecture (*sense amplifier*).

Les opérations de lecture et d'écriture se succéderont alors dans l'amplificateur de façon ininterrompue pour les accès en mode rafale (*burst*) ou avec des périodes d'attente intercalées (qui pourront être aussi de basse consommation) pour les accès en mode normal. Ces opérations s'arrêteront quand une des adresses appartiendra à une rangée différente ou quand on aura dépassé le nombre maximal de cycles permis avant de faire un *refresh*. En ce moment, toute la rangée sera préchargée dans le banc correspondant et une opération de rafraîchissement (*refresh*) se succédera dans tout le banc [Mic01a]. Si la mémoire dispose de plusieurs bancs, toutes ces actions pourront être faites de façon parallèle sur tous les bancs de façon à accélérer les accès.

Ces opérations sont contrôlées par des commandes venant du contrôleur mémoire qui sont les suivantes :

- **Inactive** : état en attente.
- **Inactive endormie** (*precharge power down*) : mode d'attente basse consommation.
- **Active** : Activation de tout le banc et transfert de la rangée à la mémoire tampon.
- **Active attente** : décodage de l'adresse du mot dans la rangée.
- **Active endormie** (*active power down*) : la mémoire attend des adresses en mode basse consommation.
- **Lecture** : opération de lecture d'un mot dans la mémoire tampon.
- **Écriture** : opération d'écriture d'un mot dans la mémoire tampon.
- **Précharge** : écriture de la rangée dans le banc correspondant.
- **Précharge_tous** : écriture de chaque rangée dans chaque banc correspondant.
- **Rafraîchissement** (*refresh*) : précharge des capacités des cellules du banc.

Les commandes sont codées dans les entrées *cs, ras, cas* et *we* provenant du contrôleur mémoire. À chaque cycle, la mémoire va décoder ces entrées et exécuter la commande correspondante, ce qui va provoquer des actions dans la mémoire qui se traduisent par une consommation d'énergie dynamique et statique. Dans le modèle fonctionnel de la mémoire, nous allons associer une valeur d'énergie par commande exécutée, qui sera représentée par une transition dans l'automate d'état.

Dans les modes endormis actif et inactif, il n'y a pas de commandes puisque l'horloge est gelée et l'automate arrêté, donc la consommation dynamique est nulle. Par contre, il aura une consommation statique qui sera prise en compte par la transition de l'horloge en dehors de l'automate.

Le passage à ces modes endormis est contrôlé extérieurement par le processeur ou par tout autre élément ayant accès à la mémoire, par l'entrée *clock enable, cke* et il se produit instantanément. Si l'on habilite l'entrée *cke* quand tous les bancs sont inactifs, on passe au mode inactif endormi (*precharge power down*). Par contre, si on le fait quand il y a une rangée active dans un banc, alors on passe au mode actif endormi (*active power down*). La durée maximale dans ce mode est limitée par

le rafraîchissement automatique des bancs. Le temps de réveil à partir de cette état est de seulement un cycle mémoire (DDRC SDRAM 64M x 32 Micron), donc il n'y a pas de surcoût temporel.

Au niveau de la modélisation cycle-précis de la mémoire, cela demande d'ajouter une variable de contrôle du passage vers les modes endormie, ainsi que :

- l'arrêt ou démarrage de l'horloge,
- la modification de la valeur d'énergie par cycle,
- l'ajout du réveil automatique pour faire le rafraîchissement des bancs.

Nous n'avons pas trouvé que le mode en hibernation soit utilisé dans les mémoires SDRAM. Par contre, d'autres nouveaux modes de basse consommation y sont implantés actuellement, le *Partial Array Refresh, PAR* et le *Deep Power Down operation, DPD*. Ils seront expliqués un peu plus loin.

Dans la figure 5.3, on illustre l'automate d'état d'une mémoire SDRAM DDRC avec les énergies associées par transition dans chaque mode de fonctionnement. Le mode nominal regroupe toutes les opérations en fonctionnement normale de la machine d'états. Les modes actif endormi et inactif endormi représentent les opérations en mode basse consommation avec l'horloge arrêtée.

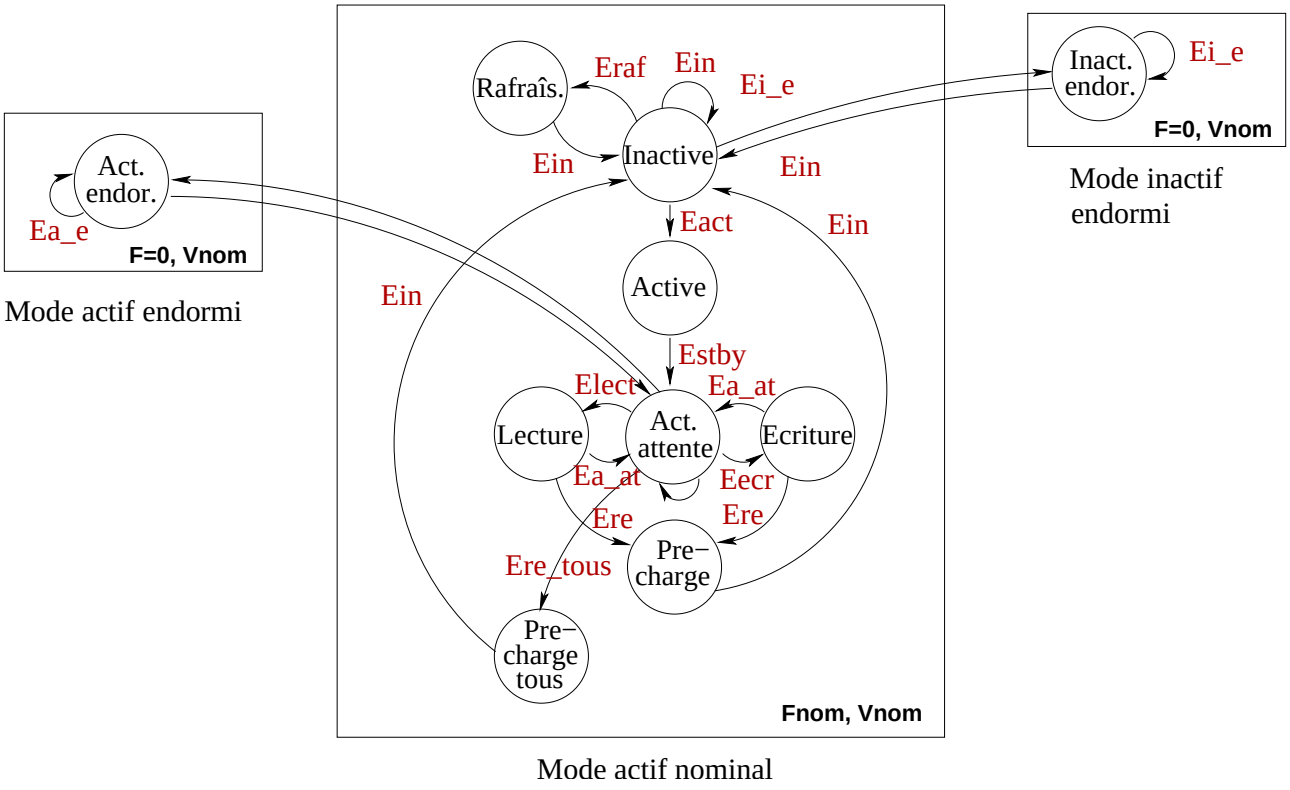

FIG. 5.3 L'automate d'état de la mémoire SDRAM

Chaque commande aura une consommation d'énergie par cycle associée. L'ensemble constituera le modèle d'énergie de la mémoire. Les valeurs de toutes ces énergies seront obtenues par macro-modélisation à partir de mesures directes sur le matériel ou à partir des valeurs données dans les notices techniques. La principale difficulté dans ce dernier cas est de bien identifier les courants donnés

dans les notices techniques avec l'état de fonctionnement correspondant, puisque chaque constructeur utilise des noms différents pour décrire ces états.

Le calcul à partir des valeurs de courant, tension et fréquence données dans les notices techniques est fait à partir de la formule 5.5.

$$E_{nom} = \frac{I_{nom} \times V_{nom}}{f_{nom}} \tag{5.5}$$

La mémoire peut travailler à d'autres valeurs de tension et de fréquence dans un rang permis et donné dans les notices. À différente fréquence, l'énergie dynamique par cycle reste invariable. Par contre, à différente tension cette valeur change. Dans ce cas, les nouvelles valeurs d'énergie par cycle seront calculées grâce à la formule 5.6.

$$E = E_{nom} \times \left(\frac{V}{V_{nom}}\right)^2 \tag{5.6}$$

De cette façon, on obtient les valeurs d'énergie par commande exécutée à chaque cycle. Un exemple est montré dans le tableau 5.7 pour une mémoire DDRC SDRAM de 64M x 32 bits de Micron. On illustre les valeurs de courant nominales mesurées à une tension de 2.6 V et données dans la notice technique [Mic01b]. On calcule l'énergie à une tension de 2.5 V et une fréquence de 83 MHz.

Mode	**Courant (mA)**		**Energie (pJ)**
Inactif (*precharge standby*)	$I_{dd2N/F}$	90	1407
Inactif endormi (*precharge power down*)	I_{dd2P}	3	47
Actif (*operating*)	I_{dd0A}	100	1615
Actif attente (*active standby*)	I_{dd3N}	95	1485
Actif endormi (*active power down*)	I_{dd3P}	55	860
Lecture (*read*)	I_{dd4R}	295	4610
Écriture (*write*)	I_{dd4W}	220	3438
Précharge	I_{dd0R}	55	808
Rafraîchissement (*refresh*)	I_{dd5}	245	3828

TAB. 5.7 La consommation d'une mémoire DDRC SDRAM Micron

Les nouveaux modes de basse consommation

Les nouvelles mémoires DRAM pour le marché portable utilisent des nouveaux modes de fonctionnement de basse consommation qui permettent de réduire encore plus la consommation. Un exemple est la mémoire CellularRAM de Micron [Mic04]. Elle implante deux nouveaux modes : *Partial Array Refresh, PAR* et *Deep Power Down operation, DPD*. Tous les deux sont modélisables avec notre approche.

Le premier mode, **PAR**, consiste à rafraîchir seulement une zone de la mémoire (la moitié, un quart ou les trois quarts). Cela permet de réduire les courants utilisés par cette opération. Cependant, les

adresses des données stockées doivent être bien maîtrisées car les zones mémoire non rafraîchies seront corrompues. On rentre dans ce mode en désactivant l'entrée *ZZ*, quand le bit *sleep* du registre de configuration est actif.

Le deuxième mode, **DPD**, consiste à désactiver tout le rafraîchissement de la mémoire. De cette façon toutes les données seront perdues. Ce mode est donc utilisé seulement dans les périodes où la mémoire n'est pas employée. Le passage est faite en désactivant l'entrée *ZZ*, quand le bit *sleep* du registre de configuration est inactif. Le démarrage est coûteux en temps et en consommation, donc l'utilisation de ce mode doit être bien justifiée.

5.3 Le modèle d'énergie des accélérateurs matériels

Les nouveaux accélérateurs matériels sont modélisés fonctionnellement en automates d'état cycle près à partir de l'algorithme séquentiel du composant. La modélisation dépend du simulateur choisi, mais toujours le modèle simule l'exécution d'une ou de plusieurs opérations par cycle, chacune avec une énergie associée. Ces énergies seront affectées à la transition de l'automate correspondante.

L'ensemble d'énergies par opération constituera le modèle d'énergie qui sera différent par composant. Par exemple, on peut trouver : $E_{inactif}$, $E_{ecriture}$, $E_{lecture}$, E_{oper1}, E_{operx}, etc. Un exemple d'automate d'état d'un accélérateur matériel avec les énergies associées par transition est illustré dans la figure 5.4.

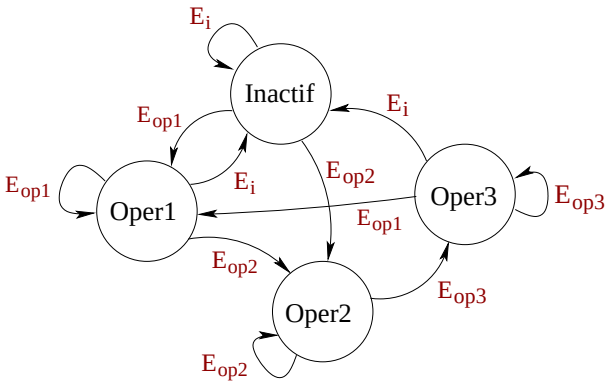

FIG. 5.4 Automate d'état d'un accélérateur matériel

5.3.1 Les modes de fonctionnement basse consommation

Le modèle fonctionnel du composant peut être modifié pour introduire des états de basse consommation. Le nouveau automate d'état ainsi obtenu est illustré dans la figure 5.5. Dans ce dessin on trouve, l'automate du mode de fonctionnement nominal et des autres modes de basse consommation avec leurs automates correspondants. Ces modes sont les suivants :

1. **Actif à tension et fréquence différentes** : correspond aux cycles fonctionnels à fréquence et tension inférieures, quand le temps d'exécution totale le permet. L'énergie consommée par cycle sera calculée avec les équations 5.9 et 5.10, mais avec les valeurs de E_{porte} et E_{ff} adaptées à la nouvelle tension.
2. **Endormi** : correspond aux cycles d'attente pendant lesquels les horloges sont gelées (*clock gating*) pour diminuer encore plus la consommation. L'énergie dissipée par cycle correspond juste aux courants de fuites et elle est calculée avec l'équation 5.11. Le temps de passage et de réveil est de seulement un cycle, donc il devrait être utilisée chaque fois que le composant n'est pas accédé.
3. **Hibernation** : correspond aux cycles d'attente pendant lesquels les horloges sont gelées et la tension est réduite pour diminuer au minimum les courants de fuites. Sa valeur est calculée avec l'équation 5.11 et les nouvelles valeurs de courant de fuites et de tension. Le temps de passage et de réveil doivent être pris en compte.

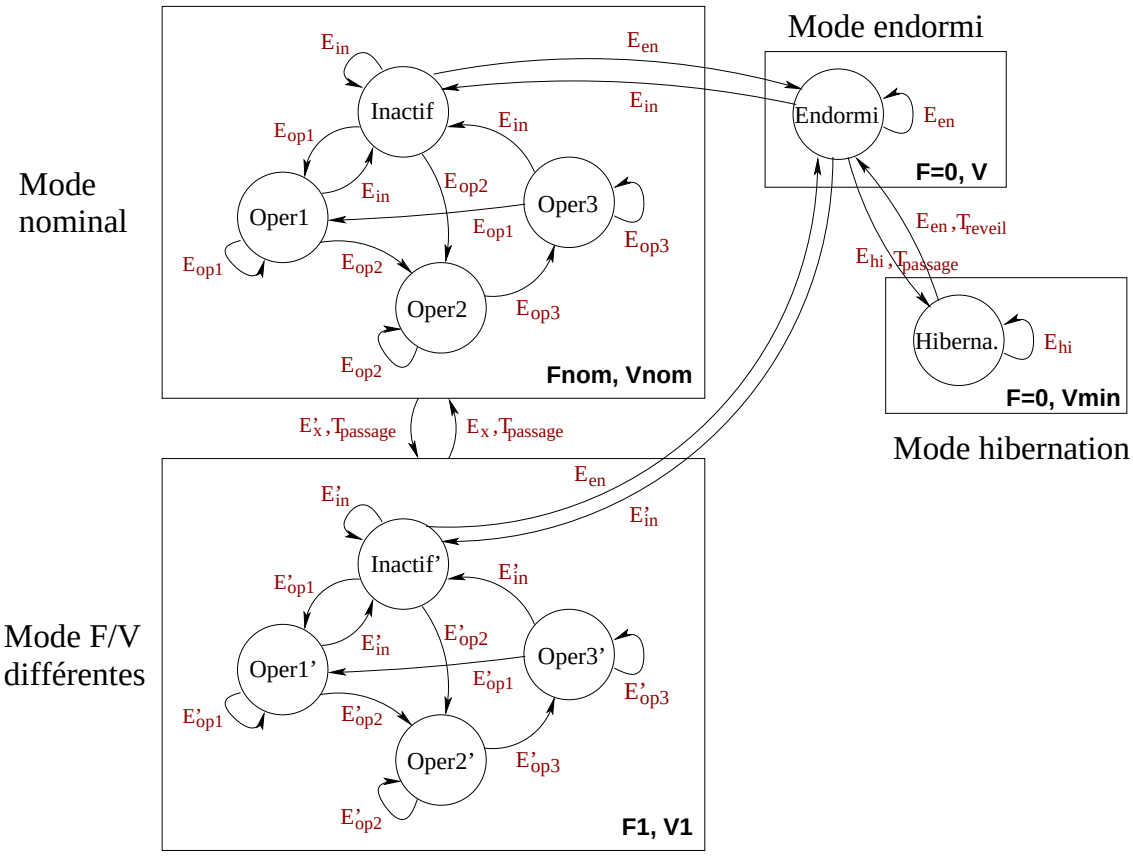

FIG. 5.5 Automate d'état d'un accélérateur matériel

L'entrée dans ces modes est contrôlée extérieurement par un module de gestion de puissance. En analysant l'application et les entrées, il se peut que de longues périodes d'inactivité du composant soient détectées, par exemple, dans le décodage MPEG4, le temps entre le décodage de deux images ou de deux macroblocs de la même image [MPE]. Dans ce cas, le module de gestion peut envoyer un signal au composant, le faisant passer au mode endormi (où les horloges seront gelées) ou au mode

en hibernation (où la tension sera réduite), ou en le faisant fonctionner au ralenti (dans les états actifs à tension et fréquence inférieures). La politique de gestion qui pilotera ces passages devra être définie *a priori* et testée pour chaque application.

La seule utilisation de l'état endormi (*clock gating*) permettra de grandes réductions en consommation. D'un autre coté, l'implantation du mode en hibernation ou des états actifs à tension et fréquence différentes s'avère plus compliquée, puisqu'il faudra non seulement bien définir la politique de gestion, qui prenne en compte la consommation et les temps de passage et de réveil, mais aussi il faudra introduire dans le circuit des convertisseurs de tension et de fréquence, ce qui va ajouter de la surface et, aussi paradoxalement, de la consommation.

Dans les applications temps réel, la simple réduction de la fréquence va permettre une bonne réduction de la consommation, sans avoir besoin de la réduction de la tension et de l'installation de convertisseurs de tension. La réduction de la fréquence implique l'augmentation du temps d'exécution pour le même traitement. Ce prolongement du temps actif implique aussi une réduction du temps passé à l'état inactif sur le temps total disponible qui reste invariable et en conséquence, une réduction du nombre de cycles inactifs et de leur consommation associée.

5.3.2 Les valeurs d'énergie

Les valeurs d'énergie par opération des accélérateurs peuvent être obtenues selon plusieurs méthodes en fonction des informations disponibles au moment de la modélisation.

Si l'on dispose des mesures très bas niveau (physique, layout ou portes), les valeurs d'énergie seront obtenues par **macro-modélisation** (comme pour le processeur et la mémoire). Dans ce cas et si le modèle fonctionnel cycle-précis est bien fait, l'**erreur** de l'estimation d'énergie sera très basse, on l'estime inférieure à **5%** (erreur des mesures).

Si l'on ne dispose pas des mesures de bas niveaux comme c'est normalement le cas pour les nouveaux accélérateurs dont on n'a pas encore de description de bas niveau, les valeurs d'énergie seront calculées par **modélisation rapide** en utilisant les équations générales de calcul de l'énergie (que l'on détaillera dans le paragraphe suivant). Les paramètres technologiques des équations seront déjà connus à ce stade de la conception. Les paramètres structurels et fonctionnels seront estimés par l'analyse de la complexité. Ce type d'analyse est fait couramment dans l'industrie et tire parti de l'expérience des concepteurs pour estimer certains paramètres comme la taille du composant mais on peut estimer également les activités. L'**erreur** du modèle ainsi construit sera d'environ **20-40%**.

Dans le cas où l'estimation des paramètres devienne trop difficile voir impossible, on utilisera l'**analyse de l'entropie**. L'**erreur** de cette estimation sera alors la plus grande d'environ **50%**.

L'**impact de la consommation** des accélérateurs sur le système dépendra du nombre d'accélérateurs et de leur complexité, mais en général dans un système avec processeur et mémoires, leur consommation ne sera plus grande de **10%**. Pour cette raison et dans ce type de système, nous pouvons nous permettre d'avoir une certaine imprécision dans leur modèle.

Nous allons ensuite détaillé comment construire les modèles par modélisation rapide et analyse de l'entropie.

5.3.3 Modélisation rapide

Si l'on utilise la *modélisation rapide*, l'équation globale qui nous donne l'énergie par opération est l'équation 5.7, qui accumule les énergies dynamique E_{dyn} et statique E_{stat} par cycle d'opération.

$$E_{tot} = E_{dyn} + E_{stat} \tag{5.7}$$

L'énergie dynamique

Les équations qui décrivent le comportement dynamique en consommation d'un composant numérique sont dérivées de la formule générale de la puissance dynamique des circuits CMOS 2.7, décrit dans le chapitre 2. On divise le composant en deux parties : l'une combinatoire et l'autre séquentielle, parce qu'elles ont des comportements en consommation différents. La partie combinatoire regroupera toutes les portes et la séquentielle tous les registres. On va appliquer la formule générale de la consommation à ces deux parties. En conséquence, l'énergie dynamique du composant par opération sera l'accumulation des deux, exprimée dans la formule 5.8.

$$E_{dyn} = E_{comb} + E_{seq} \tag{5.8}$$

L'énergie dissipée dans la **partie combinatoire** par opération exécutée E_{comb}, donc par les portes, est calculée par la formule 5.9.

$$E_{comb} = \mu_p \times E_p \times N_p \tag{5.9}$$

où μ_p est l'activité des portes (la proportion de portes qui commutent parmi toutes les portes du circuit), E_p l'énergie moyenne dissipée par une porte pendant la transition et N_p le nombre total de portes équivalentes estimé dans le composant.

L'énergie dissipée dans la **partie séquentielle** par opération exécutée E_{seq}, donc par les registres, est calculée par la formule 5.10.

$$E_{seq} = [\mu_{ff} \times E_{ff} + (1 - \mu_{ff}) \times E_{ffck}] \times N_{ff} \tag{5.10}$$

où μ_{ff} est l'activité des flipflops (la proportion de flipflops dont la sortie a changé dans ce cycle, parmi tous les flipflops du circuit), E_{ff} l'énergie moyenne dissipée par le flipflop quand il y a une transition de la sortie dans le cycle, E_{ffck} l'énergie dissipée par le flipflop quand il n'y a pas de transition en sortie dans le cycle et N_{ff} le nombre total de flipflops équivalents estimé dans le composant.

L'énergie statique

L'énergie statique est composée de plusieurs sources de dissipation de courant de fuites dans le transistor, comme on l'a expliqué dans le chapitre 2. La composante la plus importante dans les technologies actuelles ($< 0.13\mu m$) est celle due aux courants sous le seuil I_{seuil} [Pig04]. Ce courant se produit sur les transistors qui se trouvent bloqués en état OFF ($V_{gs} < V_{th}$). Mais d'autres courants plus ou moins importants existent aussi [RMMM03]. L'ensemble de tous ces courants est le courant total de

fuite I_{fuite} qui normalement est fourni dans les caractéristiques techniques de chaque bibliothèque de cellules.

Dans ce cas, l'énergie statique du composant est calculée à chaque cycle à partir du nombre total de portes et de flipflops équivalents du composant et on considère qu'elle est indépendante de leur activité [PSN05]. Chaque porte et chaque flipflop, à une tension donnée V_{dd} et pour une technologie spécifique, a un courant de fuite moyen associé $I_{fuite,p}$ et $I_{fuite,ff}$. À partir de ce deux courants, on calcule l'énergie statique consommée par cycle d'une durée ΔT, dans un composant de N_p portes et N_{ff} flipflops, grâce à la formule 5.11.

$$E_{stat} = (N_p \times I_{fuite,p} + N_{ff} \times I_{fuite,ff}) \times V_{dd} \times \Delta T \tag{5.11}$$

Le calcul des paramètres des équations

Le modèle de simulation calcule les énergies par opération grâce aux paramètres qu'on lui donne au début de la simulation. Ce sont les paramètres utilisés dans les équations 5.9 et 5.10. Chaque paramètre est d'un type différent (technologique, structurel ou fonctionnel) selon la façon dont il peut être obtenu. Dans le tableau 5.8, on liste tous les paramètres et le type de chacun.

Paramètre	**Définition**	**Type**
E_p	Energie active de la porte équivalente	Technologique
E_{ff}	Energie active du flipflop équivalent	Technologique
E_{ffck}	Energie horloge du flipflop équivalent	Technologique
$I_{fuite,p}$	Courant de fuites de la porte équivalente	Technologique
$I_{fuite,ff}$	Courant de fuites du flipflop équivalent	Technologique
V_{dd}	Tension d'alimentation	Technologique
N_p	Nombre de portes équivalentes	Structurel
N_{ff}	Nombre de flipflops équivalents	Structurel
a_p	Activité moyenne des portes	Fonctionnel
a_{ff}	Activité moyenne des flipflops	Fonctionnel
ΔT	Temps de cycle	Fonctionnel

TAB. 5.8 Les paramètres des équations de l'énergie

Les **paramètres technologiques** dépendent de la technologie choisie pour l'implantation physique du circuit et ils se trouvent dans les caractéristiques techniques de la bibliothèque correspondante. Normalement, ces valeurs sont données par une porte *nand* à deux entrées et un registre *flip-flop* standard. On va les généraliser pour toutes les portes et tous les registres du circuit.

Les **paramètres structurels** sont définis selon la structure du composant. Ils sont estimés en effectuant une analyse de complexité du composant. Cela est un procédé habituel dans l'industrie qui permet d'estimer la taille des circuits avant même d'être conçus. Il consiste à faire une analyse structurelle au plus fin grain possible du composant, pour estimer un nombre de portes et de registres équivalents. Les portes équivalentes sont des portes *nand* à deux entrées et les registres équivalents

sont des registres *flip-flop* standards. C'est pour cela que l'on utilise la consommation de ces deux éléments comme paramètre technologique. Normalement, à partir de ces valeurs, on calcule la taille du circuit, mais dans notre étude on s'en sert pour le calcul de la consommation. Ces valeurs peuvent être aussi calculées par les formules de l'entropie données dans la section suivante *"Le calcul de l'entropie"*.

Les **paramètres fonctionnels** sont définis par le comportement du composant pendant l'exécution de l'application. Ils sont estimés en effectuant une analyse comportementale du composant. Les valeurs d'activité par cycle des portes et des registres seront estimées à partir de la connaissance que l'on a sur l'opération exécutée par le composant dans le cycle actuel en descendant au plus fin grain possible ou par le calcul de l'entropie.

Par exemple, pendant les *cycles d'attente*, on sait que le composant est inactif, les activités sont donc estimées à zéro. Mais cela ne veut pas dire que le composant ne consommera rien pendant ces cycles. Comme on observe dans l'équation 5.10, les registres consomment une énergie dynamique estimée à $E = E_{ffck} \times N_{ff}$ et qui correspond à l'énergie dissipée par les transitions provoquées par coup d'horloge dans l'entrée d'horloge de tous les registres. Cette consommation n'est pas du tout négligeable. Et pendant ces périodes, il faudra aussi prendre en compte la consommation statique.

Pendant les *cycles actifs*, les valeurs d'activité sont normalement comprises entre 5% et 40%. Cela dépend fortement du type d'accélérateur, de son architecture interne (pipelines, glitches) et du style de dessin. Par exemple, pour les accélérateurs du type IDCT avec beaucoup de mouvement et d'opérations sur les données et suivant le style de développement des opérateurs de chez Philips, l'activité moyenne sera d'environ 30%.

Mais le même composant implanté sur une autre architecture et utilisant un autre style de dessin peut avoir une activité très différente, qui pourrait même être supérieure à 100% dans le pire cas. Cela s'expliquerait par le fait d'avoir une énorme quantité des transitions inutiles (*glitches*) sur les opérateurs. Dans les études que nous avons mené sur plusieurs implantations matérielles de l'IDCT utilisant différentes arithmétiques [Dup02] : classique, redondante ou mixte et plusieurs étapes de pipeline, nous avons constaté que l'énergie dissipée varie fortement en fonction de l'arithmétique choisie (40%) mais surtout du nombre de pipelines (90%). Cela s'explique par la différence en nombre de portes de l'implantation de chaque arithmétique, mais surtout de l'activité de ces portes. L'architecture qui n'utilise pas d'étapes de pipeline ou très peu a une consommation très importante car les résultats intermédiaires se propagent tout au long de l'architecture traversant tous les opérateurs sans être arrêtés dans les registres du pipeline. Cela produit une énorme quantité de transitions inutiles qui donnent une activité moyenne importante.

Plus on connaît la structure du composant, mieux on pourra ajuster cette valeur. Et surtout, une fois la description VHDL disponible, on aura les vraies valeurs des paramètres structurels et comportementaux.

Dans le tableau 7.4, un exemple des paramètres et leurs valeurs d'énergie est montré pour l'un des composants du décodeur MPEG4, l'IDCT (*Inverse Discrete Cosinus Transformation*) implanté dans une technologie CMOS12. Les paramètres technologiques proviennent de la bibliothèque des cellules CMOS12 de Philips. Les paramètres structurels et fonctionnels ont été obtenus à partir de la description VHDL du composant développée à Philips.

La valeur d'énergie $E_{operation}$ est calculée à partir des équations 5.8 et 5.11 et représente la consom-

Paramètre	Valeur
E_{porte}	10 fJ
E_{ff}	53 fJ
E_{ffck}	19 fJ
$I_{fuites,p}$	5 nA
$I_{fuites,ff}$	23 nA
V_{dd}	1.2 V
N_{portes}	6180
N_{ff}	440
$a_{p,inactif}$	0
$a_{ff,inactif}$	0
$a_{p,operation}$	36%
$a_{ff,operation}$	36%
ΔT	12 ns
Energie	**Valeur**
$E_{operation}$	36.6 pJ
$E_{inactif}$	9 pJ
E_{sleep}	0.6 pJ

TAB. 5.9 Les paramètres et l'énergie de l'IDCT

mation dynamique d'une opération avec une activité moyenne de portes et flipflops de 36%, plus la consommation statique des toutes les portes et flipflops. La valeur $E_{inactif}$ utilise aussi ces deux équations 5.8 et 5.11 et elle représente la consommation dynamique de l'entrée d'horloge des registres, ainsi que la consommation statique du composant. La valeur E_{sleep} est calculée à partir de l'équation 5.11 et elle représente seulement la consommation statique du composant. La consommation dynamique dans ce mode est nulle car on a coupé l'entrée d'horloge du composant (*clock gating*).

5.3.4 Le calcul de l'entropie

L'entropie (en théorie de l'information) peut être utilisée pour calculer les paramètres structurels et comportementaux, des circuits dont on ne connaît que l'activité des entrés/sorties et le comportement fonctionnel interne, sans avoir aucune information de son implantation physique [CCa02] [NN96]. Malheureusement, la précision obtenue est très approximative.

Le bloc est décrit comme une boite noire fonctionnelle. La simulation fonctionnelle permet d'obtenir l'activité des entrées/sorties par cycle du bloc. Avec ces valeurs, les équations de l'entropie vont donner le nombre de noeuds internes et leur activité, qui pourront être utilisés dans des équations rapides pour estimer la consommation.

L'activité du bloc est calculée par l'équation 7.1, à partir du nombre d'entrées n_b et de sorties m_b et de l'activité des entrées $D_{in,b}$ et des sorties $D_{out,b}$ dans ce cycle par rapport aux valeurs du cycle

précédent.

$$D_b = \frac{2/3}{n_b + m_b}(n_b D_{in,b} + 2m_b D_{in,b}) \tag{5.12}$$

Le nombre de noeuds internes d'un circuit purement combinatoire est estimé à partir de l'information de l'entropie de sortie et il est calculé dans l'équation 5.13, à partir du nombre d'entrés n_b et sorties m_b et de l'entropie moyenne des noeuds de sortie $H_{out,b}$.

$$N_{g,b} = (n_b + m_b) + H_{out,b}(n_b + m_b)\left[log_{10}(n_b) + \frac{0.2}{log_{10}(n_b)}\right] \tag{5.13}$$

Le calcul de l'entropie moyenne des noeuds de sortie $H_{out,b}$ est défini dans l'équation 7.3.

$$H_{out,b} = \frac{1}{m_b}\sum_{i=1}^{m_b}\left(p_i log_2\frac{1}{p_i} + (1-p_i)log_2\frac{1}{(1-p_i)}\right) \tag{5.14}$$

où p_i est la probabilité que le noeud de sortie i vaille 1. Cette probabilité est obtenue en simulation.

La précision de ces estimations dépend du circuit. On peut l'augmenter si l'on connaît l'activité de quelques noeuds internes, qui seront insérés dans les équations comme des noeuds de sortie. Pour les circuits séquentiels, une estimation du nombre de flipflops $N_{FF,b}$ sera introduite dans l'équation 5.13, donnant la nouvelle équation 7.2

$$N_{g,b} = (n_b + m_b + 4N_{FF,b}) + H_{out,b}(n_b + m_b + 4N_{FF,b})\left[log_{10}(n_b) + \frac{0.2}{log_{10}(n_b)}\right] \tag{5.15}$$

Dans [CCa02], l'erreur des estimations de consommation obtenues avec l'analyse de l'entropie par rapport aux mesures sur SPICE est de 58%. En insérant l'activité de quelques noeuds internes dans les équations de l'entropie, l'erreur est réduite à 37%. Cette précision peut suffire pour avoir une première estimation.

5.4 Le modèle d'énergie des interconnexions

Le modèle d'énergie des interconnexions est le seul qui n'est pas précis au cycle près mais au bit près. Cela vient du fait que l'interface dans ce type de modélisation système est précise au bit près et en profitant de cette circonstance, on va modéliser la consommation au bit près aussi pour augmenter la précision du modèle.

La modélisation de la consommation sera la même pour tous les types de transferts existants dans le système. La seule différence viendra des valeurs des paramètres. On a plusieurs types de transferts : à travers le bus système, à travers la carte PCB (*Printed Circuit Board*) vers des composants off-chip comme la mémoire SDRAM (le fait de ne pas pouvoir intégrer tous les composants dans la même puce n'interdit pas leur modélisation système) et finalement, à travers d'un réseau d'interconnexion avec des routeurs. Dans ce cas, la modélisation de la consommation des fils sera modélisée comme celle d'un bus, mais il faudra aussi modéliser la consommation des routeurs.

Le modèle d'énergie calcule l'énergie dissipée par l'ensemble de fils transitant à chaque cycle. Puisque la modélisation de l'interface est précise au bit près, on connaîtra le nombre exact de fils transitant à chaque cycle (activité du bus). Le modèle d'énergie est défini par l'équation 5.16.

$$E_{interc} = \mu_{fils} \times N_{fils} \times C_{fil} \times {V_{dd}}^2 \tag{5.16}$$

où μ_{fils} est l'activité par cycle du bus, N_{fils} le nombre total de fils du bus, C_{fil} la capacité par fil et V_{dd} la tension d'alimentation des portes commandant ces fils.

L'activité μ_{fils} est obtenue en simulation. Le nombre de fils du bus N_{fils} et la tension V_{dd} sont des valeurs connues selon la modélisation du système et la technologie choisie. La capacité par fil C_{fil} est un paramètre à estimer selon le type d'interconnexion (bus, carte PCB, etc.) et la technologie choisie. Pour avoir une bonne précision, le mieux est de copier ce valeur des circuits similaires déjà conçus. Un exemple de valeur de consommation par fil d'un bus $\mathbf{E}_{fil,bus}$ est **1.6 pJ** et par fil d'un accès offchip à traves la carte PCB $\mathbf{E}_{ligne,offchip}$ est **108.9 pJ**.

La précision du modèle dépend de la précision de la valeur de capacité par fil C_{fil} puisque les autres paramètres de l'équation seront connus. Si la valeur de ce paramètre est mesurée sur un circuit physique similaire ou mieux encore, du circuit directement concerné, la précision sera très élevée. Par contre, si l'on estime cette valeur grossièrement, la précision risque d'être mauvaise. En conséquence, on estime que l'**erreur** se trouvera entre **5 et 50%**.

L'**impact de la consommation** des interconnexions sur le système dépendra du type d'interconnexions utilisé. Un bus interne consomme normalement très peu, moins de **1%** (dans un système avec processeur, mémoires et accélérateurs). Par contre, les accès externes offchip à travers la carte PCB peuvent arriver à **5%** de la consommation totale du système. De toutes façon, l'impact n'est pas grand et nous pouvons nous permettre d'avoir une certaine imprécision dans le modèle des interconnexions.

Certaines prédictions pronostiquent une grande augmentation du nombre d'interconnexions et de leur consommation dans les circuits futurs. Si cela se confirme, il faudra faire très attention au calcul des paramètres, en spécial de la capacité par fil C_{fil}. Mais on pourrait même imaginer un autre modèle d'énergie des interconnexions plus haut niveau, qui au lieu d'espionner toutes les transitions des fils par cycle (ce qui prendrait trop de temps dans un tel système), donnerait une estimation de l'activité moyenne pour des parties des interconnexions, en fonction des paramètres comme la quantité des données à transmettre par cycle ou le type de composant accédé, ou le type de données (flux vidéo, instructions, etc.). La construction d'un tel modèle reste dans les perspectives en fonction de l'évolution des interconnexions dans les systèmes futurs.

5.5 Conclusion

Dans ce chapitre, nous avons détaillé les modèles d'énergie des composants typiques d'un système embarqué. Ces modèles regroupent les valeurs d'énergie dissipée par cycle en fonction des opérations réalisées et du mode de fonctionnement du composant. Il y a basiquement deux modes de fonctionnement : un mode normal et un mode endormi. Dans le mode *normal*, le composant exécute les opérations normalement et les valeurs d'énergie dynamique et statique sont associées aux transitions

de la machine d'état. Dans le mode *endormi* ou état de repos, l'horloge se trouve gelée et la machine d'état est donc arrêtée. Dans ce cas, la consommation dynamique est éliminée et il ne reste que celle statique, qui sera associée à la transition de l'horloge en dehors de la machine d'état. Le temps de passage et de réveil vers le mode endormi a été aussi détaillé pour chaque composant.

Avec ces informations, nous avons montré comment modéliser en consommation pour la simulation au niveau cycle, les composants les plus typiques d'un système embarqué. Désormais, d'autres modèles d'énergie pour d'autres composants du système peuvent être créés.

Plusieurs méthodes de calcul des valeurs d'énergie ont été proposées : macro-modélisation, modélisation rapide ou analyse de l'entropie, en fonction des informations disponibles à chaque moment. Cela permet d'avoir le maximum de précision possible par composant. Les modèles peuvent être aussi affinés au fur et à mesure que la conception avance et des information plus précises sont disponibles. L'erreur des valeurs peut donc varier en fonction de la méthode utilisée de 5% à 50%. Les méthodes qui fournissent les valeurs les plus précises sont appliquées aux composants qui ont l'impact le plus grand sur la consommation du système.

CPU (ARM940T)	**Valeur**
$E_{actif,nom}$	850 pJ
$E_{inactif,nom}$	361 pJ
$E_{actif,2}$	320 pJ
$E_{inactif,2}$	136 pJ
$E_{endormi}$	0.2 pJ
SRAM (Samsung 2M x 16)	**Valeur**
$E_{operatif}$	3214 pJ
$E_{endormi}$	5.14 pJ
SDRAM (Micron 64M x 32)	**Valeur**
$E_{inactive}$	1407 pJ
$E_{inactive,endormi}$	47 pJ
E_{actif}	1645 pJ
$E_{actifattente}$	1485 pJ
$E_{actifendormi}$	860 pJ
$E_{lecture}$	4610 pJ
$E_{ecriture}$	3438 pJ
$E_{precharge}$	808
$E_{rafraichissement}$	3828 pJ
Accélérateur (IDCT)	**Valeur**
$E_{operation}$	36.6 pJ
$E_{inactif}$	9 pJ
E_{sleep}	0.6 pJ
Interconnexions (bus, offchip)	**Valeur**
$E_{ligne,bus}$	1.6 pJ
$E_{ligne,offchip}$	108.9 pJ

TAB. 5.10 Exemple des valeurs d'énergie par cycle des composants du système

Un exemple de valeurs d'énergie estimées par cycle et par composant est montré dans le tableau 5.10. Nous observons que les composants qui consomment le plus dans le système sont les mémoires, surtout la SDRAM, puis le processeur et finalement les accélérateurs et les interconnexions. Il faut noter que l'on pourrait avoir encore plus d'accélérateurs, ce qui augmenterait la part de la consommation leur étant empruntée, mais qu'elle resterait toujours inférieure à celle du processeur et des mémoires, puisque les accélérateurs sont de composants très optimisés en performance et consommation pour la fonction à réaliser.

Nous observons aussi des grandes différences de consommation entre le mode inactif et le mode endormi (inactif avec l'horloge inhibée) pour tous les composants. Le mode endormi consomme beaucoup moins. Cela nous montre bien l'intérêt d'utiliser la technique des horloges inhibées (*clock gating*) pour réduire la consommation pendant les périodes de repos. Bien sûr, ces valeurs ne correspondent qu'à l'énergie par cycle, maintenant il faut les insérer dans le simulateur et simuler, pour obtenir les vrais valeurs de consommation de l'application et explorer les différentes possibilités d'optimisation.

Chapitre 6

Estimation de la consommation avec un simulateur cycle-précis

Sommaire

Dans les chapitres précédents, nous avons présenté la méthode d'estimation de la consommation des systèmes embarqués grâce à l'utilisation de la simulation fonctionnelle au niveau cycle. Cette simulation est un niveau nécessaire dans le flot de conception des systèmes embarqués pour la validation des choix de partitionnement, du logiciel embarqué et des communications entre composants. Le fait de l'utiliser aussi pour l'estimation et optimisation de la consommation s'inscrit parfaitement dans le flot sans augmenter notablement le temps de développement du système. Elle demande la modélisation des composants matériels en modèles fonctionnels décrits en automates d'état, dans lesquels on associe une valeur de consommation par opération exécutée dans chaque transition de l'automate. Ces valeurs sont accumulées dans le simulateur à chaque cycle d'exécution.

Il existe plusieurs simulateurs de systèmes, comme CASS [Hom01], TSS [The98] et plus récemment les environnements en SystemC [Sys]. Ce sont des simulateurs pilotés par événement (*event driven*). Nos composants matériels sont modélisés au cycle près donc, dans notre cas, seul l'horloge émet un événement, donnant ainsi des simulations au niveau cycle. La méthode de simulation des composants est différente dans chaque approche, mais le principe de modélisation au niveau cycle près reste toujours le même. En conséquence, la méthode et les modèles d'énergie que nous avons développés et présentés dans les chapitres 4 et 5, peuvent être généralisés à n'importe quel simulateur qui modélise le comportement des composants en automates d'état au niveau cycle.

Dans le cadre de notre étude nous avons utilisé TSS (*Tool for System Simulation*), un simulateur de systèmes développé par Philips et utilisé depuis 1998 pour faciliter le développement des architectures des systèmes embarqués. Nous avons modélisé en TSS un système exemple, auquel nous avons ajouté les modèles d'énergie des composants pour tester notre approche d'estimation de la consommation. Cela nous a permis d'évaluer la consommation de l'architecture du système et de tester les changements architecturaux visant l'optimisation de la consommation. En vue de généraliser notre approche, nous avons aussi modélisé un des composants du système en SystemC. Cela nous a permis de montrer la faisabilité de l'application de l'estimation de la consommation dans ce nouvel environnement de simulation, qui devient de plus en plus populaire dans la modélisation haut niveau de systèmes intégrés.

Dans ce chapitre, nous présentons donc l'application de l'estimation de la consommation dans ces deux environnements de simulation de systèmes : TSS et SystemC.

6.1 L'estimation de la consommation avec le simulateur TSS

TSS est un simulateur cycle-précis, basé en langage C/C++, qui a été développé pour simuler des architectures complexes de systèmes matériel/logiciel, dans les premières phases de conception [The98] [TSS01]. Il permet le développement de l'architecture matérielle du système, en définissant le partitionnement et l'interface entre composants, ainsi que le support du développement des pilotes logiciels et la détection des problèmes au niveau système. Le haut niveau d'abstraction des modèles permet des simulations très rapides de systèmes très complexes.

6.1.1 Les caractéristiques générales de TSS

La plate-forme de simulation de TSS est composée de trois parties :

1. le **noyau** (*kernel*) du simulateur ; qui implante la communication et la synchronisation entre modèles,
2. les **modules standard** du système ; des modèles fournis avec le kernel pour permettre la simulation,
3. et les **modules de l'utilisateur** ; des modèles de composant écrits par le concepteur en ANSI-C/C++.

La figure 6.1 donne une vue d'ensemble de l'organisation logicielle de TSS. Les modules de l'utilisateur en C/C++ sont compilés en fichiers objets par un compilateur ANSI-C et sont reliés par le *linker*, au noyau et aux modules standard du système pour créer un seule exécutable dont l'exécution permettra de simuler le fonctionnement du système.

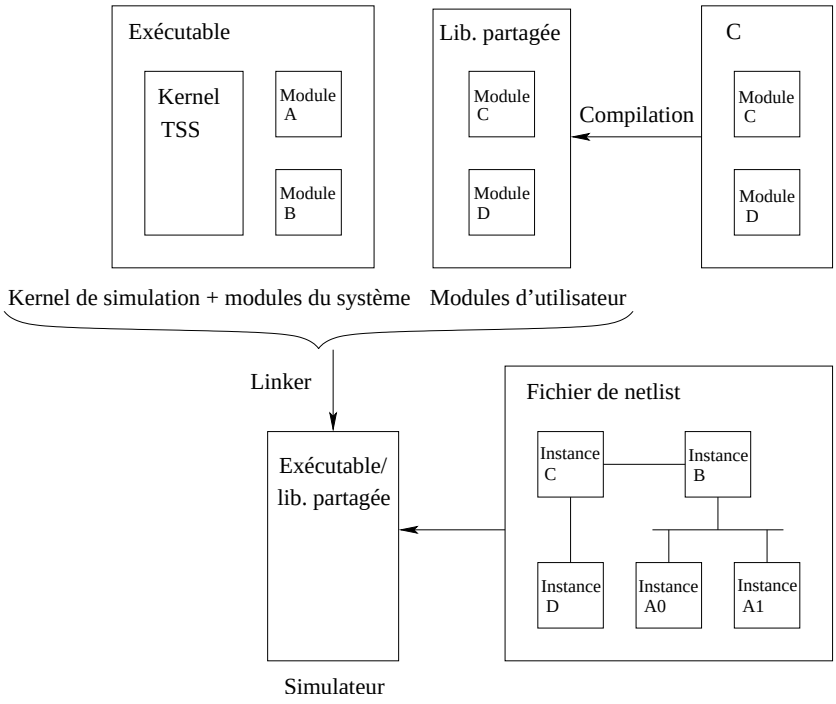

FIG. 6.1 L'organisation logicielle de TSS, source [TSS01]

La figure 6.2 illustre l'architecture logicielle de TSS. Elle nous montre les composants basiques de TSS (le kernel, les modules du système et les modules de l'utilisateur). TSS permet trois types d'interface avec l'extérieur du simulateur, illustrées dans la figure : une première, l'**interface utilisateur**, pour créer et utiliser les commandes du simulateur, qui ont été définis grâce au langage Tcl, *Tool Command Langage* ; une deuxième, l'**interface HDL**, pour permettre la co-simulation avec les simulateurs Leapfrog et Verilog-XL de VHDL et Verilog respectivement ; et une troisième, l'**interface ISS**, pour permettre la co-simulation avec des simulateurs du jeu d'instructions des processeurs (*Instruction Set Simulator, ISS*). Ces interfaces ne font pas partie du kernel donc la simulation peut toujours être faite indépendamment de leur existence. TSS fournit un bon nombre de commandes qui permettent de manipuler la simulation. Des nouvelles commandes peuvent être ajoutées.

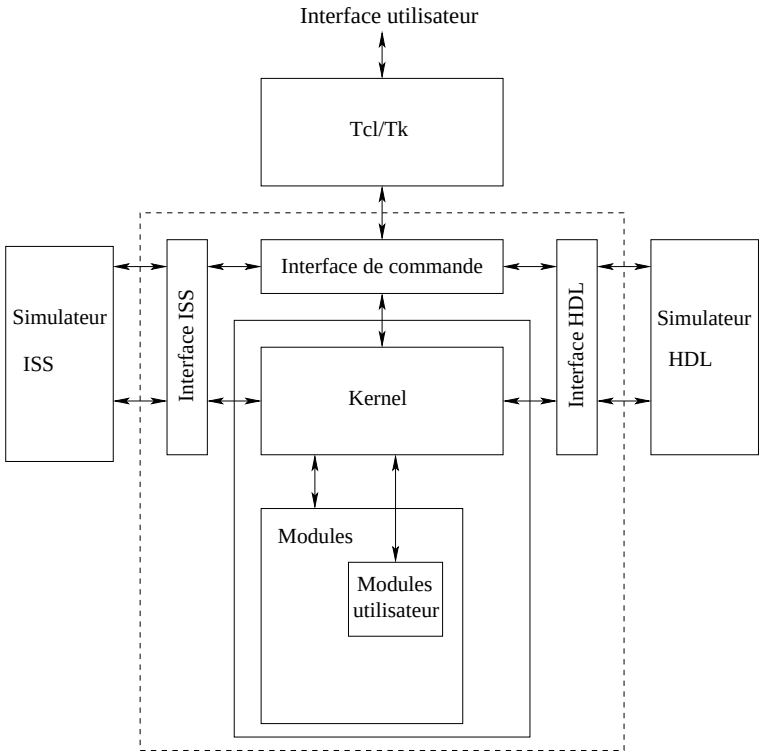

FIG. 6.2 L'architecture logicielle de TSS, source [TSS01]

Il y a deux vues possibles en TSS : la vue de l'utilisateur qui simule, que l'on appelle, *interface utilisateur de TSS* ; et la vue du concepteur modélisant en C les composants matériels, que l'on appelle *interface C de TSS*.

La modélisation consiste à écrire des modèles C/C++ décrivant le comportement au cycle des composants matériels. À partir du code C/C++, la compilation permet de disposer des exemples des modules compilés appelés *instances*. Une *instance* contient un nombre de *processus*, de *ports*, de *viewports* et d'*états*. Les *processus* représentent la fonctionnalité du composant, décrite comme un automate d'*états*. Les *ports* sont utilisés pour les communications entre les instances. Les *viewports* permettent la visualisation et le contrôle depuis l'extérieur des signaux internes et les états de l'automate de l'instance.

Le système à simuler est décrit dans un fichier de description appelé *netlist*. Ce fichier définit les instances dans le système et leurs interconnexions. Il est lu par le simulateur au début de la simulation, ce qui permet d'interconnecter les ports de instances avec des canaux. Un canal peut être connecté à plus d'un port de sortie ou d'entrée, permettant ainsi l'implantation des bus. Les horloges du système sont déclarées dans la netlist. TSS est un simulateur par événement, ce qui signifie que l'activité en TSS est ordonnée grâce aux événements, générés par les assignations aux ports de sortie ou par l'horloge du système. Pour plus d'informations sur le simulateur TSS, on pourra se référer au manuel [TSS01].

La co-simulation matériel-logiciel

TSS permet la co-simulation matériel-logiciel. Pour ce faire, le processeur peut être décrit comme un modèle TSS ou bien il peut être simulé avec un simulateur de jeu d'instructions interfacé avec TSS. Dans notre cas, on a choisi cette dernière option puisque l'on dispose du simulateur de jeu d'instructions du processeur ARM, l'ARMulator [ARM96]. L'interface avec TSS est faite grâce à un modèle TSS qui prend en entrée les requêtes d'accès mémoire faites par l'ARMulator et les transforme en requêtes mémoire TSS [Meu99]. On a modélisé en TSS le bus amba-AHB de l'ARM, donc les requêtes TSS sont faites en utilisant le protocole du bus amba-AHB. L'interface entre les deux simulateurs est illustrée dans la figure 6.3.

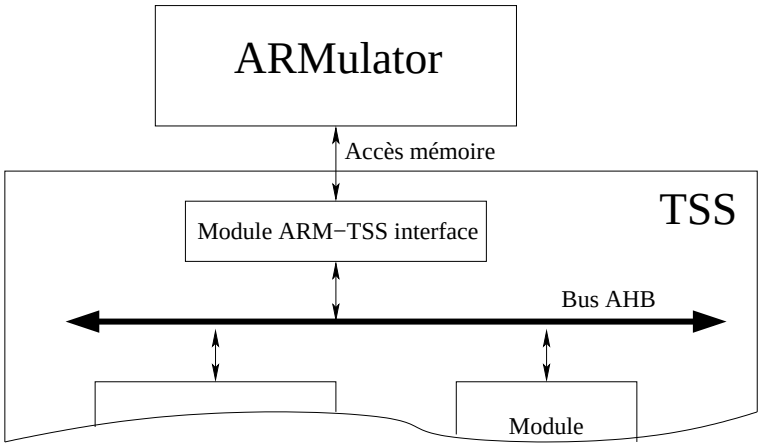

FIG. 6.3 L'interface entre TSS et ARMulator

ARMulator est un ensemble de logiciels qui permettent la simulation du jeu d'instructions et de l'architecture interne de plusieurs processeurs ARM. Il fournit un environnement de développement sur station de travail ou PC, des logiciels qui devront tourner sur ARM. Il est instruction-précis, il modélise donc le jeu d'instructions au niveau cycle sans entrer dans les détails des caractéristiques temporelles du processeur. Il dispose d'une librairie C-ANSI pour permettre la compilation et la simulation complète des programmes écrits en C. Dans notre système, on utilisera ARMulator pour simuler la partie logiciel de notre application exemple. Pour plus d'informations sur le simulateur ARMulator, on pourra se référer au manuel [ARM96].

6.1.2 La simulation TSS pour l'estimation d'énergie

Les modèles TSS sont enrichis avec des modèles de consommation de chaque composant du système. Cela va permettre d'obtenir la consommation par cycle et le consommation totale du système. La séquence typique de simulation de TSS peut être résumée comme suit :

- Le concepteur fournit la description haut niveau de chaque composant matériel du système autre que le processeur et la mémoire.
- Cette description est écrite comme un modèle TSS qui contient son modèle d'énergie.
- La partie logicielle est compilée et chargée dans les adresses spécifiques du modèle TSS de la mémoire et le système matériel complet est compilé.

- Le simulateur est mis en route en exécutant le code chargé en mémoire et en simulant le fonctionnement des modèles TSS de tout le système, avec les paramètres de la consommation de chaque composant.
- La simulation est arrêtée pour manipuler les modèles de façon à démarrer l'estimation de la consommation dans les périodes souhaitées.
- Les valeurs de la consommation sont récupérées pendant la simulation.
- À la fin de la simulation, on extrait les statistiques des valeurs de consommation totale, du nombre de cycles total et les fichiers résultat avec les valeurs évolutives de la consommation sont analysés.

Le concepteur peut ajouter tous les blocs nécessaires. Il suffit de décrire le modèles TSS du bloc avec son modèle d'énergie et de l'interconnecter au système.

6.1.3 Les modèles TSS pour l'estimation d'énergie

L'estimation de l'énergie en TSS est faite en utilisant des modèles fonctionnels des composants, qui sont instrumentés pour permettre la prise en compte des paramètres d'énergie, le calcul de la dissipation de chaque type d'opération et l'accumulation de la consommation totale à chaque cycle.

Pour cela on utilise les paramètres, les variables et les viewports standard des modèles TSS, normalement utilisés pour décrire le comportement fonctionnel du composant, mais dans notre cas, servant à l'estimation de la dissipation d'énergie. Le simulateur TSS a été aussi modifié pour prendre en compte les valeurs d'énergie. Une nouvelle commande a été ajoutée dans l'interface Tcl pour lancer et manipuler l'estimation de l'énergie. Le calcul de l'énergie est fait pendant la simulation fonctionnelle, mais cela ne ralentit pas celle ci puisqu'il s'agit d'une simple accumulation de valeurs d'énergie par cycle.

Pour l'estimation de la consommation avec TSS, on se situe donc dans les deux vues suivantes : l'une dans l'*interface C*, du coté du concepteur qui va créer les modèles et ajouter les valeurs d'énergie et l'autre, dans l'*interface utilisateur*, du coté de l'utilisateur qui va estimer l'énergie dissipée pendant la simulation.

Les paramètres d'énergie

Les **paramètres d'énergie** sont utilisés à chaque simulation pour transmettre des valeurs spécifiques à chaque instance, qui vont permettre le calcul de l'énergie par opération. Ils ont été préalablement calculés pour chaque modèle et pour les modes de fonctionnement que l'on veut tester et se trouvent disponibles dans une librairie. De cette façon, on peut donner à l'instance des valeurs qui vont permettre d'ajuster le modèle d'énergie aux conditions d'usage courantes, sans avoir besoin de recompiler tout le système à chaque fois.

Parfois, la valeur transmise sera la propre énergie par opération, d'autres fois, ce seront des paramètres des équations de l'énergie spécifiques aux modèles (comme l'énergie par porte, le nombre de portes, etc.). Les paramètres sont lus et le calcul de l'énergie par opération est fait, juste au début de la simulation, pendant la phase de création des instances. La façon de créer et d'utiliser les paramètres

est la même que n'importe quel autre paramètre de TSS et les noms utilisés sont propres à chaque modèle. Les paramètres que l'on trouve sont par exemple :

`nb_gates`	(nombre de portes)
`energy_gate`	(énergie dissipée par porte)
`activity_oper1`	(activité dans l'opération 1)
`energy_sleep`	(énergie dissipée dans un cycle endormi)

Les variables d'énergie

Les **variables d'énergie** représentent les valeurs d'énergie par opération du composant. Comme on l'a vu dans le chapitre 4, à chaque transition de l'automate d'état, on exécute une ou plusieurs opérations qui ont chacune une consommation d'énergie associée. Dans le modèle TSS, ces valeurs sont calculées au début de la simulation à partir des paramètres et des équations définies pour le modèle (voir chapitre 5). Pendant la simulation, les opérations réalisées à chaque transition de l'automate sont détectées et au même moment, la valeur d'énergie correspondante est accumulée. Chaque modèle a ses propres variables en fonction de la complexité des opérations et de son modèle d'énergie. On peut trouver par exemple :

`energy_idle`	(énergie dissipée dans les cycles inactifs)
`energy_rw`	(énergie dissipée par l'opération read/write)
`energy_oper1`	(énergie dissipée par l'operation 1)
`energy_oper2`	(énergie dissipée par l'operation 2)

Nous utilisons aussi quelques **variables spéciales** pour l'accumulation et le contrôle du calcul des énergies. L'accumulation de l'énergie dissipée par cycle est faite dans une variable appelée `energy_value`. Elle s'appelle de la même façon dans tous les modèles car elle est aussi utilisée pour le calcul de la consommation d'énergie totale en dehors du module. En plus d'elle, ils existent deux autres variables qui permettront l'activation du calcul de l'énergie dans les périodes définies par l'utilisateur : `energy_command` et `energy_calculate`. `Energy_command` est associée à un viewport pour permettre le passage des commandes pendant la simulation depuis l'extérieur vers l'intérieur du modèle. `Energy_calculate` est modifiée (`true` ou `false`) à l'intérieur du modèle en fonction de la commande passée. Cela permettra de manipuler l'accumulation de l'énergie de trois façons différentes : `reset` pour mettre à zéro la valeur de `energy_value` ; `start` pour commencer l'accumulation ; et `stop` pour l'arrêter. Nous pouvons passer ces commandes à tout moment et dans l'ordre que l'on veut, pour définir les périodes dans lesquelles l'énergie doit être accumulée. Ces trois variables utilisent toujours les mêmes noms, qui sont :

`energy_value`	(accumulation de l'énergie)
`energy_command`	(commande d'énergie)
`energy_calculate`	(modification du calcul de l'énergie)

La façon de déclarer et d'utiliser les paramètres et les variables dans les modèles, est montrée dans le modèle TSS exemple qui se trouve dans l'annexe A.

Les viewports d'énergie

Les **viewports d'énergie** permettent d'accéder à la valeur des variables d'énergie depuis l'extérieur du modèle, pour calculer la somme des énergies de tous les composants et pour visualiser les valeurs d'énergie locale et totale. N'importe quelle interface peut accéder aux viewports pour manipuler les valeurs ou simplement pour les afficher sur un outil de visualisation (par exemple en créant un fichier VCD). Il y a deux types de viewports : un premier avec la valeur accumulée d'énergie ; et un deuxième avec la commande qui permet de manipuler l'opération d'accumulation de l'énergie (`reset, start, stop`). Les viewports ont les mêmes noms dans tous les composants, qui sont : `vp_energy_value`, pour l'accumulation de l'énergie et `vp_energy_command` pour la commande d'énergie.

La façon de déclarer et d'utiliser les viewports est montrée dans le module exemple qui se trouve dans l'annexe A.

La commande `"energy"`

Pour communiquer avec les viewports, une nouvelle commande a été implantée, `"energy"`. Elle permet la manipulation de l'estimation de l'énergie de tout le système ou des composants, depuis l'*interface utilisateur*. On calcule l'énergie accumulée locale de chaque composant et totale de tout le système, grâce aux opérations suivantes :

`Start` : pour commencer à compter l'énergie d'un ou plusieurs modules.
`Stop` : pour arrêter de compter l'énergie d'un ou plusieurs modules.
`Value` : pour afficher la valeur d'énergie d'un ou plusieurs modules.
`Calculate` : pour afficher la somme d'énergie de tous les modules.
`Reset` : pour remettre à zéro la valeur d'énergie d'un ou plusieurs modules.

Le fonctionnement et les paramètres de la commande `"energy"` sont expliqués dans le manuel de l'estimateur [Abr04].

Les résultats

Le calcul de l'énergie peut être fait par périodes, qui seront contrôlés par cycles d'exécution à partir de l'*interface utilisateur* et la commande `"energy"` ou par arrêt logiciel. Ce dernier est fait en introduisant des points de rupture (`breakpoints`) dans le code du logiciel s'exécutant sur ARMulator.

Le simulateur écrit ces valeurs dans un fichier résultat `nom_fichier.trace`, qui pourra être visualisé avec l'outil d'affichage de Philips *Telescope*. Un exemple d'affichage est illustré dans la figure 6.4. Pour éviter la création d'un fichier de dimensions excessives, les valeurs sont prises par périodes de 1000 cycles (mais l'accumulation continue à se faire à tous les cycles). TSS permet aussi de créer un fichier (`nom_fichier.vcd`) avec la valeur des variables internes à tous les cycles et qui pourra être visualisé par les logiciels *Gtkwave* ou *Simwave* de Cadence. Dans les gros systèmes ou dans les longues simulations, le fichier généré risque d'être très volumineux et difficilement manipulable.

Pour cela, nous privilégions l'utilisation de *Telescope* pour la visualisation. Pour voir des exemples de visualisation avec ces outils on pourra se référer au manuel de l'estimateur [Abr04].

FIG. 6.4 L'affichage avec Telescope

6.2 L'estimation de la consommation en simulation SystemC

SystemC est une librairie de classes C++ et une méthode de développement utilisées pour créer des modèles cycle-précis des algorithmes logiciels, des architectures matériels de systèmes et de composants et des interfaces entre composants [Sys]. Pour ce faire, SystemC génére un modèle exécutable du système, grâce à la librairie SystemC et les outils standard de C++, qui en exécution se comporte de la même façon que le système. SystemC est très similaire à TSS et son but est le même, la validation et l'optimisation de systèmes, permettant l'exploration des différents algorithmes et la définition de la spécification du système. Cependant la façon de modéliser et de simuler les systèmes de chacun est un peu différente. Nous allons expliquer les caractéristiques générales de SystemC et les différences par rapport à TSS, surtout pour l'estimation de la consommation. Certains concepts de SystemC comme les interfaces, les canaux et les événements sont inspirés de SpecC [DGG02]. L'initiative SystemC s'est développée dans le cadre de l'OSCI (Open Source systemC Initiative).

6.2.1 Les caractéristiques générales de SystemC

SystemC utilise les mêmes concepts que TSS afin de permettre la modélisation et la description du matériel, avec ses caractéristiques inhérentes comme le parallélisme et l'aspect temporel. Par contre, SystemC les implante par des classes C++ tels que les modules, ports, interfaces, canaux, processus (*threads* et *methods*). Les composants principaux de ce langage de co-design sont :

- **Modules** : ce sont les entités. Différemment qu'en TSS, en SystemC on peut y avoir une hiérarchie, c'est à dire une entité peut contenir d'autres entités.
- **Processus** : il sont utilisés pour décrire la fonctionnalité du module. Ils se trouvent à l'intérieur des modules, comme en TSS.
- **Ports** : Ils sont utilisés pour connecter les modules de la même façon qu'en TSS, sauf qu'en SystemC, au lieu de faire les interconnexions dans le fichier de `netlist`, elles sont faites dans l'entité au niveau hiérarchique le plus élevé où se trouve la fonction `sc_main`.
- **Signals** : Ce sont les lignes qui connectent les ports (comme en TSS).
- **Niveaux de conception** : SystemC permet plusieurs niveaux de conception depuis des modèles fonctionnels haut niveau jusqu'aux modèles RTL cycle-précis très détaillés. En TSS, seulement le niveau cycle-précis est modélisable.
- **Protocoles de communication** : Plusieurs niveaux d'abstraction et de sémantique sont permis en SystemC.
- **Déboguage** : les classes ont une fonction de déboguage qui peut être activée en compilation. Par contre, il n'existe pas encore une interface de simulation comme en TSS, avec des commandes pour le contrôle de la simulation et le déboguage. Mais rien n'empêche qu'elle soit développée.

SystemC permet la modélisation d'un système à partir d'un niveau fonctionnel abstrait, dans lequel les modèles sont affinés en vue du partitionnement. Cet affinement se poursuit progressivement en vue de tenir compte des détails d'implantation et des contraintes de synthèse pour arriver aux modèles RTL cycle-précis synthétisables. De cette façon, on peut facilement détecter les bogues et procéder aux changements nécessaires. Il est ainsi possible de surveiller la cohérence du système tout au long du processus de conception. Les simulations peuvent être réalisées avec le même ensemble de test et le design peut ainsi être vérifié à chaque niveau, ce qui évite de lourdes modifications en phase finale. La fiabilité du design est améliorée en créant une spécification détaillée du système et en permettant le test du hardware et du software par rapport à cette spécification tout au long du cycle de conception. Par ailleurs, le fait de garder le même langage pour tous les niveaux d'abstraction permet d'éviter des traductions propices aux erreurs.

À la différence de TSS, un sous ensemble du langage SystemC est synthétisable. Il est équivalent à celui d'un langage HDL classique (Verilog ou VHDL) et les outils actuels utilisent d'ailleurs le même noyau de compilation pour synthétiser les descriptions. Ce sous ensemble n'est certainement pas figé et pourrait varier en fonction des outils futurs.

L'architecture logicielle de SystemC est illustré dans la figure 6.5. Dans ce diagramme, il y a quelques concepts importants à expliquer :

- Tout en SystemC est écrit en C++.
- Les couches supérieures dans le diagramme sont construites dans SystemC au dessus des couches inférieures.
- Le coeur du langage SystemC fournit un jeu minimal de constructeurs des modèles pour les descriptions structurelles, le parallélisme, la communication et la synchronisation.
- Le coeur implémente aussi un modèle de calcul général qui permet l'exploitation des modèles suivants : réseau de Kahn, processus séquentiels communicants et événements discrets.
- Les types de données théoriquement sont indépendants du coeur du langage et l'utilisateur peut en définir des nouveaux.
- Les mécanismes de communication comme les signaux ou les fifos sont construits au dessus du coeur, ainsi que certains modèles de calcul (*Model Of Computation, MOC*).

– Les couches inférieures du diagramme peuvent fonctionner sans les supérieures.

Une routine générale appelée `sc_main` regroupe le système complet, avec tous les modules et leurs interconnexions. On y trouve aussi la génération de l'horloge et l'activation des traces de sortie. Ce fichier inclut tous les modules du système. Ils sont instantiés et leurs ports sont connectés avec des signaux pour créer le système complet.

Le système est compilée pour créer un fichier exécutable. La simulation se fait en exécutant ce fichier. Elle fait l'instanciation de la netlist et la simulation du comportement du système, en tournant pendant le nombre de cycles spécifié dans le fichier `sc_main`. Quand la simulation est finie, un fichier de trace `VCD` est créé. Il peut être exploité avec des outils de visualisation comme *Gtkwave* ou *Simwave* de Cadence, pour analyser la valeur des signaux et des variables pendant la simulation et détecter si elles sont correctes.

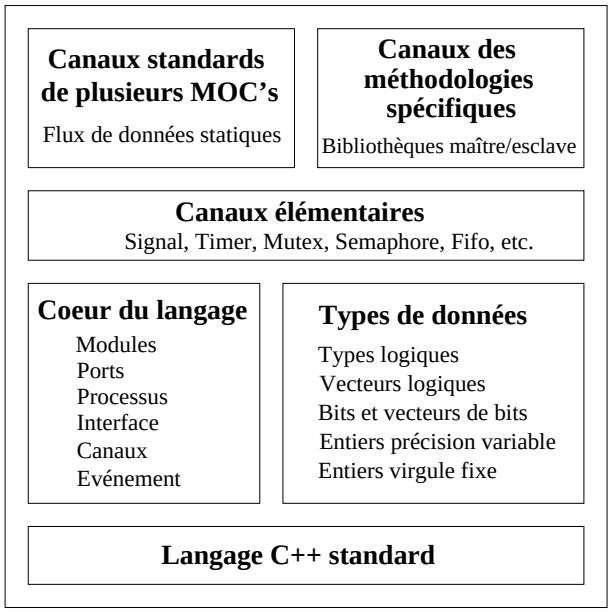

FIG. 6.5 L'architecture logicielle de SystemC

La co-simulation matériel-logiciel

SystemC permet la co-simulation matériel-logiciel à plusieurs niveaux de conception. Aux niveaux fonctionnels les plus élevés, les tâches logicielles et matérielles ne se distinguent pas, puisque le choix de l'architecture du système n'a pas été encore fait, donc toutes les tâches sont décrites au même niveau fonctionnel abstrait. C'est au moment du partitionnement que les tâches seront assignées aux processeurs (tâches logicielles) et aux accélérateurs (tâches matérielles). Dans ce cas, et pour pouvoir simuler les tâches logicielles et matérielles ensemble, un modèle en SystemC du processeur doit être utilisé. Il sera précis au niveau instruction ou au niveau cycle, pour permettre de simuler le jeu d'instructions du processeur. Des interfaces avec d'autres simulateurs de jeu d'instructions comme ARMulator, peuvent être aussi créés, pour permettre son utilisation comme en TSS.

6.2.2 La simulation SystemC

Les modèles SystemC précis au cycle sont enrichis avec des modèles de consommation, de la même façon qu'on a fait en TSS. Cela permet aussi l'obtention de la consommation d'énergie par composant et la consommation totale. La séquence de simulation diffère un peu de celle de TSS. Elle peut être résumée comme suit :

- Le concepteur fournit la description SystemC fonctionnelle haut niveau du système.
- Cette description est partitionnée et décrite en modèles SystemC cycle-précis avec leurs modèles d'énergie.
- L'ensemble est compilé pour créer l'exécutable.
- Le simulateur est mis en route en exécutant le code chargé en mémoire et en simulant le fonctionnement des modèles SystemC des blocs matériels avec leurs paramètres spécifiques.
- La simulation se déroule pendant le nombre de cycles déclaré dans la fonction `sc_main` et les valeurs d'énergie sont accumulées dans les périodes souhaités et récupérés dans un fichier.
- À la fin de la simulation, on extrait les statistiques des valeurs de consommation et du nombre de cycles et on analyse le fichier résultat.

Comme on voit, la simulation SystemC est presque identique de la simulation TSS. Les principales différences sont l'absence d'interface de simulation et d'un débogueur et puis la nécessité de recompiler à chaque fois que l'on fait un changement sur les paramètres des modèles ou sur le déroulement de la simulation. On trouve aussi une différence dans la vitesse de simulation. Selon des mesures faits à Philips, dans le pire cas, SystemC version 2.0 est 3 fois plus lent que TSS.

6.2.3 Les modèles SystemC pour l'estimation d'énergie

L'estimation de l'énergie en SystemC est faite de la même manière qu'en TSS, en ajoutant des valeurs d'énergie aux modèles des composants. On ajoute des nouvelles variables qui vont permettre le calcul de la valeur de consommation par transition de l'automate. Les valeurs numériques sont les mêmes, puisque la modélisation de l'énergie est faite également selon les opérations exécutées par transition. Par contre, la façon de les insérer dans des variables, de manipuler la simulation et d'obtenir les résultats diffère un peu. On va voir les différences en détail.

Les paramètres, les variables d'énergie et les viewports

Les **paramètres** en TSS permettent de passer des valeurs spécifiques aux modules à chaque simulation, sans être obligés de recompiler à chaque fois. En SystemC, nous pouvons passer des paramètres à chaque module durant sa déclaration dans la fonction principale `sc_main`. Une fois le système compilé, nous ne pouvons plus changer les paramètres, pour le faire il faut donc recompiler à chaque fois. Cela peut ralentir les tests des gros systèmes où la compilation peut être longue, s'il faut changer les paramètres à chaque scénario. Une solution est de rendre les modèles paramètrables par la ligne de commande.

Les **variables d'énergie** se comportent de la même façon en SystemC qu'en TSS. Elles prennent la valeur de la consommation des opérations, lesquelles sont accumulées à chaque transition durant la

simulation. Le calcul de la valeur est aussi fait à l'initialisation de la simulation grâce au *constructor*, qui est la fonction de création de l'instance, la liste de sensibilité des horloges et l'on déclare les variables. Dans notre cas, on va aussi calculer l'énergie par opération à partir des paramètres passés au modèle et des équations du modèle.

Les **variables spéciales** d'accumulation et de contrôle du calcul des énergies sont utilisées de la même façon qu'en TSS. Par contre, le contrôle n'est plus fait pendant la simulation à partir d'une commande. Les périodes dans lesquelles la consommation sera calculée sont désormais programmées à l'avance et pour les cycles souhaités dans la fonction principale `sc_main`, en activant et désactivant la variable `energy_calculate`. Il existe aussi la possibilité d'insérer des `breakpoints` dans les fonctions des modèles. Cela peut permettre un certain réglage pour le calcul de la consommation. Par contre, en SystemC, on ne dispose pas d'une *interface utilisateur*, la commande `"energy"` et la variable `energy_command` n'existent donc pas, ce qui nous empêche de manipuler l'estimation d'énergie pendant la simulation.

La façon de déclarer et d'utiliser les paramètres et les variables d'énergie dans un modèle SystemC, est montrée dans le modèle exemple qui se trouve dans l'annexe B.

Le calcul de l'énergie et les résultats

SystemC permet de garder la trace des signaux et des variables sur un fichier au format `VCD`, `WIF` et `ISDB`. Ces fichiers gardent la valeur par transition des signaux et des variables qui ont été incluses dans la fonction de trace. De cette façon, ils seront visualisés avec un outil d'affichage pour ces formats. Pour des grands systèmes ou des longues simulations, le fichier résultat risque d'être trop grand. Dans ce cas, des valeurs d'énergie peuvent être prises chaque n cycles et écrites dans un autre fichier d'un autre format correspondant à l'outil d'affichage de notre choix, par exemple l'outil Telescope de Philips. Dans ce cas, on aura un fichier plus facile à manipuler et à analyser.

6.3 Conclusion

Dans ce chapitre, nous avons présenté les principales caractéristiques de la simulation au niveau cycle avec TSS et avec SystemC, en vue de l'estimation de la consommation. En résumé, la façon de modéliser la consommation en TSS repose sur l'utilisation des variables, des paramètres et des commandes standard du simulateur, qui sont normalement créés pour la description fonctionnelle du module. Cependant, nous les utilisons pour le stockage des valeurs d'énergie par opération, pour l'accumulation de l'énergie pendant la simulation et pour la manipulation de cette accumulation de façon à définir les périodes de comptage de l'énergie. Les paramètres de chaque modèle prennent des valeurs numériques préalablement calculées, spécifiques pour chaque modèle et pour les modes de fonctionnement choisis et qui sont disponibles dans une librairie. Les résultats obtenus peuvent être visualisés de différentes façons, en créant un fichier résultat au format le mieux adapté aux besoins d'analyse.

Dans l'environnement SystemC, la façon de modéliser est pratiquement la même. Sachant que la sémantique des variables et des paramètres est différente, la façon dont on les utilise dans les modèles

reste très similaire. D'ailleurs, le temps de modélisation global reste pratiquement le même.

Comme différences, on trouve qu'en SystemC les paramètres de chaque modèle sont déclarés dans le modèle qui regroupe tout le système et non pas dans une netlist comme en TSS, ce qui oblige à recompiler chaque fois que l'on fait varier un paramètre, La façon de simuler et de définir les périodes de comptage de l'énergie varient aussi puisque en SystemC, on ne dispose pas encore d'interface utilisateur comme TSS. Cela nous oblige à définir les périodes depuis les modèles, au lieu de manipuler la simulation grâce à l'utilisation d'une commande spéciale pour l'énergie. La visualisation des résultats reste pratiquement la même parce que dans tous les deux environnements, on peut sauvegarder les valeurs accumulées d'énergie dans des fichiers au format souhaité.

En conclusion, les différences dans l'estimation de la consommation avec ces deux environnements, TSS et SystemC, ne sont pas significatives. Ce ne sont que des différences sémantiques ou des certaines fonctionnalités qui existent en TSS et pas encore en SystemC, mais rien n'empêche qu'elles soient ajoutées dans le futur. En conséquence, les modèles de consommation définis dans le chapitre 5 sont parfaitement utilisables dans ces deux environnements. De cette façon, on montre que notre approche d'estimation est valable dans n'importe quel simulateur des systèmes au niveau cycle près.

Chapitre 7

Validation sur l'application du décodeur MPEG4

Sommaire

Dans les chapitres précédents, nous avons présenté notre approche de modélisation et d'estimation de la consommation au niveau cycle et les modèles de consommation des principaux composants d'un système embarqué typique. Puis, nous avons montré la façon d'implanter ces modèles sur un simulateur au niveau système cycle-précis comme TSS ou SystemC.

Dans ce chapitre, nous présentons l'application de notre approche sur un exemple de système embarqué approprié pour des études d'estimation et de réduction de la consommation : un circuit décodeur vidéo MPEG4 destiné à la téléphonie mobile.

Nous allons d'abord présenter la norme MPEG4 et le système matériel-logiciel choisi pour l'implantation du décodeur vidéo. Nous décrirons ensuite les modèles d'énergie de chaque composant du système, avec les valeurs retenues des paramètres et des variables d'énergie. Puis nous détaillerons les évaluations de la consommation du système pour le décodage d'une image et d'un macrobloc et la comparaison des valeurs avec les mesures prises au niveau portes logiques. Finalement, plusieurs techniques de réduction de la consommation seront appliquées au système et évaluées en se servant de notre approche d'estimation, de façon à trouver celles dont le gain en énergie est le meilleur, au regard des pertes en performance et de l'augmentation de la complexité de l'implantation.

7.1 Le système décodeur MPEG4

7.1.1 Les généralités de la norme MPEG4

MPEG4 est un standard ISO/IEC développé par le comité d'experts MPEG (Moving Picture Experts Group) [MPE]. L'objectif premier de la norme MPEG4 était de succéder aux normes MPEG1 pour la compression et le transfert audio-vidéo et MPEG2 pour la télévision numérique. Mais lors de l'élaboration de la norme, il a été défini un champ d'applications et de fonctionnalités dépassant largement le cadre d'une simple évolution.

Une grande variété de formats vidéo sont supportés par MPEG4, donnant lieu à une gamme de débits qui s'étend de 64 kbit/s à 38,4 Mbit/s [Mir00]. La compression n'est pas le seul but poursuivi par cette norme. MPEG4 inclut en effet des nouvelles fonctionnalités et définit l'architecture d'un système flexible et ouvert à des futures extensions tant au niveau algorithmique qu'au niveau de la nature de l'information à transmettre.

Les éléments intégrant la spécification de la norme s'appellent des entités MPEG4. L'application envisagée dans le cadre de cette thèse, un décodeur vidéo MPEG4, est définie dans l'entité *MPEG4 visual*, car c'est dans cette partie que les algorithmes de codage/décodage des objets visuels sont spécifiés. Il y a différents types d'objets visuels supportés par la norme : objets vidéo, objets graphiques, objets images fixes, etc. Notre application concerne le décodage des objets vidéo.

7.1.2 Le codage/décodage vidéo MPEG4

Le schéma de codage/décodage utilisé dans les normes MPEG est illustré dans la figure 7.1. On distingue deux types de codage : INTRA et INTER.

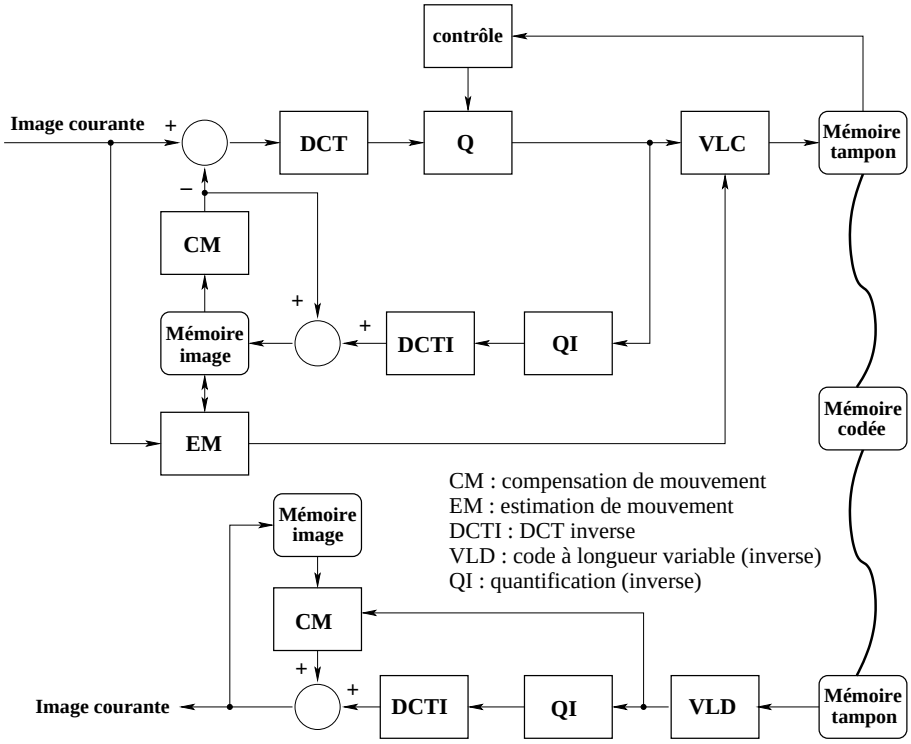

FIG. 7.1 Le schéma de codage des images MPEG4, source [Mir00]

Le codage des images de type **INTRA** n'exploite que la redondance spatiale de l'image, sans faire référence aux images antérieures ou postérieures. Une Transformée en Cosinus Discrète (DCT) suivie d'une Quantification (Q), puis d'un Code à Longueur Variable (VLC) sont appliqués à chaque bloc de (8×8) pixels de luminance et chrominance [Mir00].

Le codage des images **INTER** exploite à la fois les redondances spatiale et temporelles des images de la séquence. Le codage de type INTRA est appliqué non plus directement sur l'image courante, mais sur l'erreur résiduelle entre l'image courante et une prédiction de cette image, construite par estimation et compensation de mouvement. L'estimation de mouvement (EM) est basée sur une image de référence qui est, dans le cas le plus simple, l'image précédente. L'image de référence est stockée tant au codeur qu'au décodeur (mémoire d'image).

Pour chaque macrobloc [1] de l'image courante, l'estimation de mouvement consiste à déterminer quel est le bloc de l'image de référence qui lui "ressemble" le plus. Ce macrobloc sert alors de prédicteur au codage du bloc courant. L'erreur résiduelle entre ces deux blocs est calculée et un codage de type INTRA, décrit précédemment, lui est appliqué. Le vecteur de mouvement (qui définit la position du bloc prédicteur par rapport au bloc courant) est aussi codé et transmis au décodeur. Une mémoire tampon en sortie du codeur permet de réguler le débit du flux codé. Le pas de quantification peut être ajusté de façon à éviter le débordement ou la sous-utilisation du tampon de régulation.

Coté décodeur, le processus inverse a lieu pour la reproduction du bloc de l'image courante. Après le décodage de code à longueur variable (VLD), les opérations de quantification inverse (QI) et de DCT

[1] Un macrobloc (MB) est constitué d'un bloc carré de 16×16 pixels (luminance et chrominance sous-échantillonnée).

inverse (IDCT) sont effectuées. À ce stade, on récupère, soit le bloc original dans le cas d'un codage de type INTRA, soit l'erreur de prédiction, si le codage est basé sur les images précédentes.

Dans MPEG4, quelques modifications ont été néanmoins nécessaires par rapport aux normes précédentes, puisque les algorithmes de codage/décodage sont appliqués à des objets de forme et taille arbitraires et non à des images rectangulaires. Un traitement spécifique est défini pour le codage/décodage de la forme. Des raffinements algorithmiques ont d'autre part été introduits dans un souci d'amélioration des performances.

Pour permettre l'implantation effective du standard, MPEG4 a identifié des sous-ensembles d'outils susceptibles d'être utilisés ensemble dans un certain nombre d'applications. Ces sous-ensembles sont appelés "profils". Ils limitent la complexité d'un décodeur tout en lui donnant la conformité à la norme et la possibilité de communiquer avec d'autres terminaux MPEG4.

L'application visée dans le cadre de cette thèse est un système embarqué codec vidéo MPEG4 pour des applications multimédia portables. Il implante le **profil visuel simple**, qui fournit un codeur, efficace et robuste aux erreurs, d'objets vidéo rectangulaires, adapté pour les applications de réseaux mobiles. Ce profil permet le traitement d'images de taille QCIF [2] et CIF [3]. Nous nous occupons de l'implantation matérielle de la partie décodeur.

7.1.3 La modélisation en TSS

Le décodeur MPEG-4 vidéo développé chez Philips est un sous-système destiné à être intégré dans un terminal portable. L'architecture de son système est hybride, certaines fonctions sont réalisées par du logiciel embarqué s'exécutant sur un microprocesseur, d'autres étant réalisées par de la logique matérielle. De cette façon, les performances sont améliorées tout en réduisant la dissipation d'énergie du système. L'ensemble est relativement complexe. L'algorithme de décodage a été d'abord écrit en C/C++, puis complètement modélisé sous forme d'un réseau de processus parallèles (*Kahn Process Network*). Ce modèle a servi de point de départ au travail de recherche.

À partir de ce réseau de processus, on a fait le partitionnement et le choix de l'architecture matérielle du système. Puis la modélisation au cycle en TSS est utilisée pour permettre la simulation fonctionnelle du système entier afin de valider le partitionnement et les choix architecturaux, définir les spécifications des composants matériels et développer la composante logicielle du système. Dans le cadre de cette thèse, cette modélisation a été utilisée pour l'estimation et l'optimisation de la consommation au niveau système. L'architecture du système retenue est composée d'un processeur, d'une mémoire SDRAM, des mémoires FIFOS, des accélérateurs matériels et des interconnexions. L'architecture complète est illustrée dans la figure 7.2.

Le microprocesseur analyse le flux de données et contrôle et coordonne le fonctionnement de l'ensemble des blocs matériels. Il va accéder souvent à la mémoire principale pour chercher les instructions et les données. C'est une machine RISC, l'**ARM940T**, avec caches d'instructions et de données [Adv98]. Les caches et les buffers d'écriture du processeur améliorent la performance du

[2] QCIF : *Quarter Common Intermediate Format*, dont la résolution vidéo est de 180 pixels/ligne $\times$ 144 lignes à 15 images/s.

[3] CIF : *Common Intermediate Format*, dont la résolution vidéo est de 360 pixels/ligne $\times$ 288 lignes à 30 images/s.

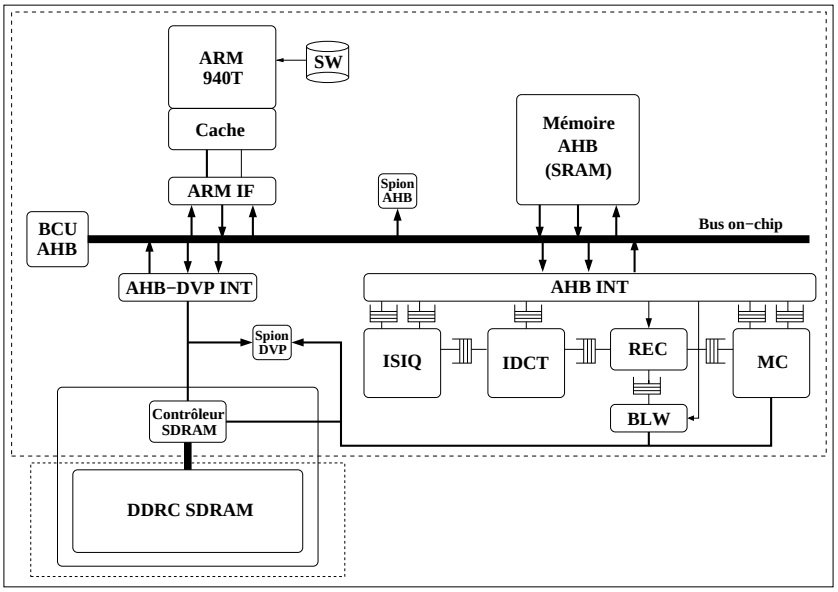

FIG. 7.2 L'architecture du système décodeur MPEG4

système et minimisent les accès au bus et à la mémoire externe, ce qui apporte des réductions notables de la consommation du système.

La simulation du processeur en TSS est faite grâce à l'utilisation du simulateur de l'ARM, l'ARMulator, interfacé avec l'environnement TSS grâce au modèle **ARM IF** [Meu99]. ARMulator est configuré pour simuler le comportement du jeu d'instruction du processeur ARM940T et le modèle des caches. Les accès mémoire du processeur sont pris en compte par l'interface ARM IF qui les transforme en requêtes mémoire de lecture en burst de 4 mots et d'écriture à travers le bus amba-AHB. Le contrôleur de bus est simulé grâce au modèle **BCU AHB** [Meu99].

La mémoire est utilisée principalement pour stocker la partie logicielle du décodeur, le flux vidéo comprimé en entrée (*bitstream*) et les images décodées en sortie. Cette quantité de données est tellement importante que le système peut difficilement se contenter de la taille d'une mémoire SRAM standard. La solution est donc d'utiliser une mémoire de grande taille et beaucoup de capacité électrique comme une SDRAM externe au circuit intégré ou bien une grande mémoire embarquée SDRAM ou SRAM si la technologie le permet. Par exemple, la technologie *system-in-package* (SiP) [Ina], dans laquelle plusieurs circuits sont assemblés dans le même boîtier ; ou bien, les technologies *embedded DRAM* [Inf] ou *SRAM-1T* [Mos], où les grandes mémoires sont intégrées dans le même circuit que l'ASIC. Ces solutions permettront la réduction de la consommation dans les accès mémoire, grâce à l'utilisation des mémoires avec une taille adaptée à l'application et moins de capacité électrique en conséquence et l'élimination de la dissipation d'énergie due à l'activation des broches d'entrée/sortie de l'ASIC et de la mémoire, ainsi que celle due au passage par les pistes de la carte de circuit imprimé.

La première solution retenue a été d'utiliser une mémoire SDRAM externe, car les autres technologies n'étaient pas encore disponibles chez Philips. Pour leur simulation, nous utilisons un modèle TSS très complexe de mémoire SDRAM qui implante le contrôleur mémoire et la plupart des fonctionnalités et des commandes d'une mémoire **DDRC SDRAM** [Gur01]. Il utilise le protocole DVP-DTL (*Device Transaction Level*), un protocole de communication point à point développé par Philips [Phi01b]. Une

interface **AHB-DVP INT** a été insérée pour permettre la communication entre le bus et la mémoire. Nous disposons aussi d'un modèle de mémoire **SRAM** connectée directement au bus amba-AHB. Celui-ci nous permet de tester les autres solutions de mémoire embarquée comme la mémoire *SRAM-1T*.

Dans le système modélisé, on trouve aussi le modèle d'une mémoire SRAM, **Mémoire AHB**, qui nous permettra de tester la faisabilité de l'utilisation de cette mémoire pour le stockage d'autres données en dehors du flux vidéo comprimé et des images (par exemple les instructions, le heap, etc.), ce qui devrait améliorer la performance globale du système et réduire la consommation.

Le reste des composants sont les accélérateurs matériels qui effectuent les fonctionnalités du décodage MPEG4 considérées les mieux adaptées pour sa réalisation en matériel dédié : les opérations choisies sont très coûteuses en termes de temps CPU et de consommation, elles sont suffisamment régulières pour permettre une réalisation matérielle en ASIC simple et bien optimisée en performance, surface de silicium et consommation et finalement elles sont suffisamment générales pour pouvoir être réutilisées dans d'autres types ou versions du décodeur, constituant ainsi des blocs IP (*Intellectual Property*) intégrant la bibliothèque de composants de Philips. Ces opérations sont ISIQ, IDCT, MC, REC et BLW.

ISIQ fait la quantification inverse, **IDCT** la transformée en cosinus discrète inverse, **MC** la compensation de mouvement, **REC** la reconstruction des pixels et **BLW** (*Block Writer*) envoie les pixels décodés à la mémoire SDRAM. Les communications entre accélérateurs sont faites grâce aux mémoires de type **FIFO** et leur accès depuis le bus amba-AHB grâce à l'interface **AHB-INT**.

Dans le cadre du projet décodeur MPEG4 développé à Philips et pour les besoins du projet, ces accélérateurs finalement n'ont pas été développés en TSS sur un système de simulation opérationnel, mais ils ont été directement écrits en VHDL et implantés sur une carte de prototypage avec des FPGA. Cela a permis de montrer la performance du système avant de l'implanter sur des circuits ASIC. Leur utilisation a accéléré le système d'un facteur entre 4 et 5.

Par contre, pour les besoins de ce travail de recherche, j'ai rendu opérationnelle la modélisation TSS du système avec le processeur ARM (simulé en ARMulator et relié à TSS), le bus amba-AHB, une partie du décodeur MPEG4, la mémoire SDRAM avec son interface et la mémoire SRAM. Les modèles TSS du décodeur créés permettent le décodage des images INTRA et ce sont l'IDCT, REC et BLW, avec leurs FIFOS de communication et l'interface avec le bus amba-AHB.

Ce sous-système permet juste de gagner 10-15 % d'accélération par rapport à la solution purement logicielle, ce qui ne justifierait pas une implantation matérielle. Il s'agit donc d'un système irréaliste mais qui a permi de tester cette méthode d'estimation de la consommation au niveau système grâce au décodage des images INTRA. Le modèle de consommation du bloc MC (*Motion Compensation*) a été aussi créé car il nous semblait intéressant de le montrer, puisqu'il est l'accélérateur le plus consommateur du décodeur. Cependant, il n'a pas été simulé car son modèle fonctionnel TSS n'était pas disponible.

7.2 Les modèles d'énergie du décodeur MPEG4

Pour l'évaluation de la consommation, nous avons introduit des modèles d'énergie dans les composants les plus importants du système du point de vue de la consommation. Ce sont le processeur, les mémoires, (SDRAM, SRAM et FIFOS) les accélérateurs matériels (IDCT, REC et BLW) et les interconnexions (onchip et offchip).

Les modèles d'énergie des composants ont été construits suivants les bases théoriques expliquées dans le chapitre 5. Plus précisément, nous avons employé la **macro-modélisation** pour construire les modèles du processeur et des mémoires et la **modélisation rapide** pour les accélérateurs et les interconnexions. Nous avons aussi utilisé l'**analyse de l'entropie** pour estimer certains des paramètres du bloc IDCT de façon à valider la précision de cette méthode par rapport aux mesures prises au niveau portes.

Les macro-modèles ont été créés à partir des valeurs de dissipation d'énergie directement prises des notices techniques. Ce sont les valeurs de l'énergie ou du courant par mode de fonctionnement. La modélisation rapide calcule l'énergie à partir d'équations dont les paramètres proviennent dans notre cas de la description RTL au niveau portes logiques.

Nous allons maintenant illustrer les valeurs et les paramètres spécifiques utilisés pour l'implantation du décodeur MPEG4 dans une technologie 0.12 micron.

7.2.1 Le processeur ARM940T

Le processeur configuré sur ARMulator est un ARM940T avec caches d'instructions et de données. L'automate d'état est illustré dans la figure 5.1 du chapitre 5. Les valeurs de consommation par cycle pour chaque mode de fonctionnement d'une implantation à une technologie CMOS12 de Philips (0.12 micron) sont illustrées dans le tableau 7.1. Les valeurs nominales et en mode endormi proviennent de la bibliothèque des cellules de CMOS12 Philips. Ce sont des mesures physiques sur le composant donc leur précision est très élevée. Les valeurs à tension différente sont calculées avec les équations 5.1 et 5.2 du chapitre 5.

Mode	Fréquence	Tension	Energie/Cycle
Actif nominal	250 MHz	1.3 V	250 pJ
Inactif nominal	250 MHz	1.3 V	110 pJ
Actif V/F inférieures	83 MHz	0.8 V	100 pJ
Inactif V/F inférieures	83 MHz	0.8 V	45 pJ
Endormi	83-185 MHz	1.3-1 V	10 pJ

TAB. 7.1 La consommation du processeur ARM940T plus caches

L'implantation du modèle de consommation est faite dans le modèle TSS de l'interface de communication ARM IF (montré dans la figure 7.2). Deux paramètres sont implantés : `energy_nop` et `energy_active`, qui prennent les valeurs d'énergie directement pour les modes inactif et actif

respectivement (nominal, V/F différentes ou endormi). Cette interface détecte les requêtes que le processeur fait à la mémoire externe et les transforme en accès mémoire à travers le bus Amba-AHB. À chaque cycle, dans ce modèle, on peut savoir s'il y a une requête ou pas et son type : lecture ou écriture.

- Les *lectures* sont la conséquence d'un défaut de cache et donc elles produiront plusieurs cycles d'attente dans le processeur, des cycles inactifs dont l'énergie accumulée sera celle des *modes inactifs*.
- Pendant les cycles *internes* (sans requête mémoire), le processeur travaille à partir des caches. Dans les cycles d'*écriture* en mémoire, il écrit les donnés dans le tampon d'écriture sans arrêter l'exécution. Du point de vue de la consommation, ces deux cas sont des cycles actifs dont l'énergie accumulée est celle correspondant aux *modes actifs*.

Les valeurs à fréquence/tension différentes substitueront les valeurs nominales sur les deux paramètres du modèle, ainsi que la fréquence de travail sera modifiée, quand on voudra tester ces modes de fonctionnement.

Nous n'avons pas implanté sur notre modèle le *mode endormi* car notre système ne produit pas de cycles d'attente sur le processeur à part les défauts de cache. Cependant, ce mode reste toujours modélisable. Il suffit d'ajouter à l'interface ARM IF une variable de détection du passage vers ce mode (contrôlée extérieurement), l'état endormi sur l'automate, le paramètre `energy_endormi` et la modélisation des cycles de réveil. L'utilisation de ce mode demanderait aussi l'implantation d'un module de gestion de la puissance dynamique pour le contrôle du passage.

ARMulator simule le comportement du processeur au niveau cycle avec une précision de presque 100%. Les valeurs d'énergie utilisées sont aussi très précises car elles proviennent des mesures physiques. C'est pourquoi, l'erreur estimée de notre modèle d'énergie est d'environ 5% (erreur des mesures).

Le temps d'obtention du modèle a été rélativement court car on disposait des mesures physiques du processeur concerné. Le développement du modèle fonctionnel TSS de l'interface ARM IF, la mise au point des communications entre ARMulator et le modèle ARM IF et l'introduction du modèle d'énergie m'ont demandé environ trois semaines.

7.2.2 La mémoire SDRAM

Le modèle d'énergie de la mémoire SDRAM utilise les valeurs de consommation d'une mémoire DDRC SDRAM de 64M x 32 bits de Micron [Mic01b]. À partir des valeurs nominales de courant données dans la notice technique et mesurées à une tension de 2.6 V, on calcule l'énergie à une tension de 2.5 V selon les équations expliquées dans le chapitre 5. Les valeurs obtenues sont illustrées dans le tableau 7.2.

L'implantation du modèle de consommation est faite dans un modèle TSS de mémoire SDRAM DDRC développé à Philips. L'automate d'état est celui présenté dans la figure 5.3 du chapitre 5. Le modèle TSS disponible implante tous les modes de fonctionnement, sauf les modes basse consommation (inactif endormi et actif endormi). Il permet les accès en mode raffale (*burst*).

Les paramètres utilisés sont directement les valeurs d'énergie : `energy_idle,`

Mode	Energie/Cycle
Inactif (*precharge standby*)	1407 pJ
Inactif endormi (*precharge power down*)	47 pJ
Actif (*activate*)	1615 pJ
Actif attente (*active standby*)	1485 pJ
Actif endormi (*active power down*)	860 pJ
Lecture (*read*)	4610 pJ
Écriture (*write*)	3438 pJ
Précharge	808 pJ
Rafraîchissement (*refresh*)	3828 pJ

TAB. 7.2 La consommation d'une mémoire DDRC SDRAM Micron 64M x 32 bits

`energy_act_standby`, `energy_activate`, `energy_read`, `energy_write`, `energy_precharge` et `energy_refresh`. Les valeurs données aux paramètres sont accumulées en fonction de la commande générée à chaque transition de l'automate. Cette commande contrôle le fonctionnement de la mémoire à chaque cycle en fonction des entrées et de l'état actuel. Cela provoque un changement d'état ou pas puis l'exécution de l'opération correspondante à l'état : activation, lecture, écriture, précharge, etc.

Même si le modèle TSS n'a pas implanté les modes endormis, leur utilisation peut être évaluée en affectant au paramètre `energy_idle` la valeur de l'énergie *inactive endormi* (au lieu de l'*inactive*) et au paramètre `energy_act_standby` la valeur de l'énergie *active endormi* (au lieu de l'*active attente*). Cela suppose un passage automatique à ces modes (contrôlé extérieurement) à chaque fois que la mémoire n'est pas accédée. Cette modélisation simple n'introduit pas d'erreur car le passage à ces modes n'a pas de surcoût en temps et en énergie sur la mémoire.

Cependant, une correcte modélisation des modes endormis demanderait d'ajouter au modèle TSS une variable de détection du passage vers ce mode (contrôlée extérieurement), les états endormis sur l'automate et les paramètres `energy_inactif_endormi` et `energy_actif_endormi`. L'utilisation de ce mode demanderait aussi l'implantation d'un module de gestion de la puissance dynamique pour le contrôle du passage.

Le modèle TSS de la mémoire SDRAM DDRC est précis au niveau cycle. Les valeurs d'énergie utilisées sont aussi très précises car ils proviennent des mesures physiques fournies par le constructeur. C'est pourquoi, l'erreur estimée de notre modèle d'énergie est d'environ 5% (erreur des mesures).

Le modèle fonctionnel de cette mémoire est assez compliqué et son développement peut demander deux mois à un concepteur expérimenté. On disposait du modèle fonctionnel TSS de cette mémoire et les valeurs de consommation ont été prises des notices techniques [Mic01b]. Par contre, l'introduction des valeurs d'énergie dans le modèle fonctionnel a nécessité environ deux semaines car cela a demandé la totale compréhension du fonctionnement interne du modèle et la modification de certaines fonctionnalités. Aussi, certaines erreurs de fonctionnement on été détectées dans l'interface de la mémoire avec le bus AHB qui ont nécessité deux semaines suplementaires de travail de correction.

7.2.3 Les mémoires SRAM et FIFOS

Les modèles d'énergie des mémoires SRAM et FIFO utilisent les valeurs de consommation des mémoires SRAM de taille optimale pour l'application. Pour la mémoire SRAM, nous avons choisi une *SRAM-1T* [Mos]. Cette technologie utilise des cellules de stockage avec un seul transistor, au lieu des 6 transistors des cellules SRAM standard. Cela permet la réduction de la taille à moitié pour la même capacité de stockage et la réduction du coût de la mémoire de 50-70 % et de la consommation de puissance de 75 %. La taille maximale de ce type de mémoire est de 2Mbits, ce qui permet de stocker 8 images QCIF. La technologie choisie est 0.13μm à une tension de 1.2 V.

Les mémoires FIFO sont implantées dans des mémoires SRAM standard, pour une technologie CMOS12 de Philips à une tension de 1.2 V. La taille des mémoires FIFO est de 8Kbits maximum. Ces mémoires possèdent un mode endormi et leur consommation pendant les cycles inactifs est pratiquement négligeable. On suppose que ce mode est toujours utilisé dans les périodes sans accès à la mémoire (par contrôle extérieur).

Les valeurs de consommation de ces deux types de mémoire sont illustrées dans le tableau 7.3.

Mémoire	**Mode**	**Energie/Cycle**
SRAM-1T 2Mbits	Actif	480 pJ
”	Inactif	180 pJ
FIFO 8Kbits	Actif	20 pJ
”	Endormi	12 fJ

TAB. 7.3 La consommation des mémoires SRAM et FIFO

Nous utilisons un modèle TSS des SRAM et un autre de FIFO. Fonctionnellement, ils sont différents bien que du point de vue de la consommation, le modèle d'énergie reste le même. L'automate d'état du modèle d'énergie a été montré dans la figure 5.2 du chapitre 5. Les paramètres sont directement les valeurs d'énergie : `energy_idle, energy_read` et `energy_write`. Ces deux derniers prennent la valeur du mode actif si dans les notices techniques il n'y a pas de distinction de valeurs de consommation entre les lectures et les écritures. Dans ces modèles, la transition vers un état actif (lecture ou écriture) ou inactif est détectée à chaque cycle et l'énergie correspondante est accumulée.

Le modèle TSS des SRAM est précis au niveau cycle. Les valeurs d'énergie utilisées sont aussi précises car elles proviennent des mesures physiques fournies par le constructeur que l'on suppose bien faites. C'est pourquoi, l'erreur estimée de notre modèle d'énergie est d'environ 5% (erreur des mesures).

Le modèle fonctionnel de cette mémoire est simple et rapide à faire ainsi que l'introduction du modèle d'énergie aussi si l'on dispose des mesures physiques. On disposait du modèle fonctionnel TSS de cette mémoire et les valeurs de consommation ont été prises des notices techniques [Mos]. Leur introduction dans le modèle fonctionnel m'a demandé donc seulement une journée.

7.2.4 Les accélérateurs matériels

Le modèle d'énergie des accélérateurs est construit par modélisation rapide, en utilisant les équations 5.9 et 5.10 présentées dans le chapitre 5. Ces équations nous permettent de calculer la consommation dynamique (combinatoire et séquentielle) et statique à partir des paramètres. Dans notre cas, les paramètres sont calibrés à partir des mesures sur la description en VHDL au niveau portes logiques de chaque accélérateur. La précision ainsi obtenue est celle atteinte au niveau portes, qui donne une erreur de 10-15%.

Les valeurs dont on dispose au niveau portes sont la taille, le nombre de portes, le nombre de flip-flops et la consommation totale du bloc. Cette dernière est mesurée avec l'outil d'estimation DIESEL [Phi01a] pour le décodage d'une image typique. L'activité par opération est calculée à partir de la mesure de consommation totale et insérée dans les équations avec les autres paramètres. De cette façon, on obtient l'énergie par opération de chaque accélérateur avec la même précision qu'au niveau portes.

Les valeurs d'énergie mesurées sur DIESEL pourraient être insérées directement dans le modèle TSS par macro-modélisation, comme on a fait pour le processeur et la mémoire. Cependant, nous avions construit les modèles d'énergie par modélisation rapide avant de disposer de la description VHDL, donc une fois la description disponible et mesurée, on a tout simplement calibré les paramètres des modèles avec les vrais valeurs pour améliorer la précision.

Dans le cas de l'IDCT, nous avons aussi utilisé l'analyse de l'entropie pour estimer les paramètres d'activité et nombre de portes. Ce composant est le plus adapté pour ce type d'analyse car il effectue une opération combinatoire homogène et bien définie ce qui facilite le calcul. Nous avons comparé les valeurs obtenues avec les mesures prises au niveau portes pour estimer l'erreur de cette méthode.

Le temps d'obtention de chaque modèle demandé la modélisation fonctionnelle de l'accélérateur au niveau cycle et de l'estimation des paramètres d'énergie. Cela dépend de la compléxité de l'accélérateur. Dans ce cas, pour modéliser chaque accélérateur et estimer les paramètres à partir des mesures prises sur la description en VHDL au niveau portes, j'ai nécessité en moyenne une semaine.

Chaque accélérateur a un automate d'état propre selon son fonctionnement. Nous allons maintenant illustrer les automates et la structure de chacun, les opérations exécutées par transition et les tableaux avec les valeurs d'énergie.

IDCT

L'IDCT fait la transformée en cosinus discrète inverse. La structure interne du composant est illustrée dans la figure 7.3. Il est composé de deux blocs IDCT-1D, qui exécutent deux fois en série la transformée monodimensionnelle (1D) et des trois buffers pour stocker les coefficients d'entrée, de sortie et intermédiaires. Les 64 coefficients de chaque bloc entrent en série par le buffer d'entrée, puis sont présentés en parallèle à la première IDCT-1D. La sortie est stockée dans le buffer intermédiaire, puis elle est présentée à la deuxième IDCT-1D. Le résultat est stocké dans le buffer de sortie avant d'être écrit en série dans une FIFO extérieur. De cette façon, la transformée est faite sur le premier bloc. Les blocs suivants sont traités de la même façon. On profite du remplissage en série du premier buffer pour vider le dernier, on travaille donc en pipeline avec trois blocs en même temps, chacun dans un

buffer.

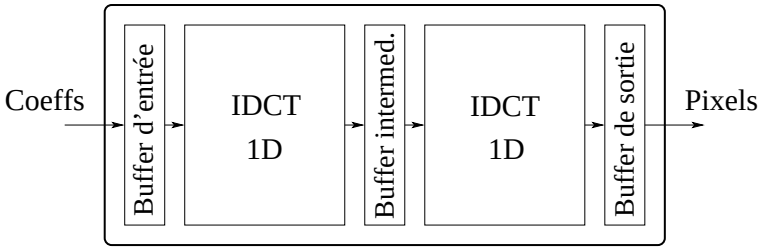

FIG. 7.3 La structure interne de l'IDCT

La machine d'états illustrée dans la figure 7.4, est composée de 8 macro-états, qui exécutent le remplissage, le traitement et la purge du pipeline pour les 6 blocs d'un macrobloc. Les macroblocs sont décodés indépendamment les uns des autres. Le premier macro-état, `R0`, lit les coefficients du premier bloc et remplit le buffer d'entrée. Le deuxième, `R1`, lit le deuxième bloc et traite le premier avec la première IDCT-1D, en écrivant le résultat dans le buffer intermédiaire. Le troisième, `R2-W0`, lit le troisième bloc, traite le deuxième bloc avec la première IDCT-1D et le premier bloc avec la deuxième IDCT-1D et écrit son résultat dans le buffer de sortie. Les macro-états suivants ont le même fonctionnement jusqu'à traiter les 6 blocs du macrobloc et vider le pipeline. Les énergies associées par transition sont les suivantes : `E_idle` ou `E_sleep` dans les transitions correspondantes à l'attente des coefficients de remplissage ou de vidage des buffers ; et `E_idct1D`, dans les autres transitions, parfois multipliée par 2 puisqu'on exécute les deux IDCT-1D dans la même transition sur les pixels de deux blocs différents.

Paramètres	**Valeur**
`nb_gates`	6180
`nb_ff`	440
`energy_gate`	10 fJ
`energy_ff`	53 fJ
`energy_ffck`	19 fJ
μ`_idct1D`	36 %
Energie par opération	**Valeur**
`E_idle`	9 pJ
`E_sleep`	0.6 pJ
`E_idct1D`	36.6 pJ

TAB. 7.4 Les paramètres et la consommation de l'IDCT

Ces énergies par opération sont illustrées dans le tableau 7.4. Elles sont calculées à partir des paramètres donnés au modèle et les équations 5.8 et 5.11 du chapitre 5.

Les paramètres sont obtenus pour l'architecture interne de l'IDCT choisie par Philips. Par contre, pour d'autres architectures internes du composant ces valeurs peuvent changer. Dans ce but, nous avons testé plusieurs implantations matérielles de l'algorithme de Loeffler utilisant différentes

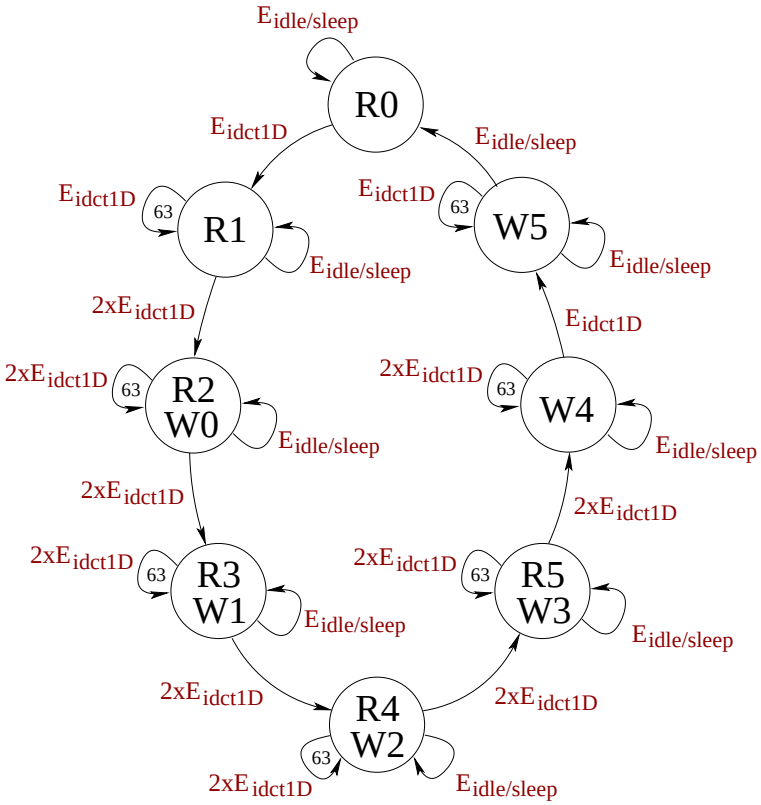

FIG. 7.4 L'automate de l'IDCT

arithmétiques : classique, redondante ou mixte ; et plusieurs étapes de pipeline [Dup02]. Nous avons constaté que la consommation varie fortement en fonction de l'arithmétique choisie mais surtout du nombre de pipelines. La variation maximale de dissipation d'énergie dans l'IDCT pour le décodage d'une image est 40 % en fonction de l'arithmétique et 90 % en fonction du nombre de pipelines. Cela s'explique par la différence en nombre de portes et surtout d'activité des portes pour chaque arithmétique. Un plus grand nombre de pipelines diminue énormément l'activité, puisque cela empêche le passage d'un grand nombre de transitions inutiles (*glitches*).

L'analyse de l'entropie

Dans les nouveaux composants pour lesquels nous n'avons pas encore d'informations structurelles et comportementales de bas niveau, les équations de l'analyse de l'entropie peuvent être utilisées pour calculer le nombre de portes et l'activité interne. Ces deux paramètres seront donc estimés en fonction de l'activité des entrées et des sorties du composant pendant la simulation fonctionnelle du système. On n'aura pas besoin d'aucune indication sur l'architecture interne du composant, ce qui permettra de rester complètement indépendants de l'implantation future.

Pour tester cette approche, nous avons calculé les valeurs des paramètres de l'IDCT. Pour ceci, nous avons simulé en TSS l'opération de l'IDCT sur les 64 coefficients d'entrée de chaque bloc. Les paramètres et le calcul fait sont les suivants :

– L'**activité interne** du bloc par cycle actif est calculée à partir de l'équation 7.1, étant n_b le nombre d'entrées, m_b le nombre de sorties, $D_{in,b}$ la moyenne de l'activité des entrées par cycle et par rapport au cycle précédent et $D_{out,b}$ celle des sorties. Ces activités sont obtenues en simulation.

$$D_b = \frac{2/3}{n_b + m_b}(n_b D_{in,b} + 2m_b D_{in,b}) \tag{7.1}$$

Le tableau 7.5 illustre les paramètres et l'activité résultante par bloc pour le premier macrobloc d'une image exemple.

Macrobloc	**Bloc**	**Nombre bits entrée**	**Nombre bits sortie**	**Activité bits entrée**	**Activité bits sortie**	**Activité IDCT**
0	0	12×64	9×64	5.3 %	29.2 %	**18.7 %**
0	1	12×64	9×64	5.3 %	23.6 %	**15.5 %**
0	2	12×64	9×64	6.5 %	30.6 %	**19.9 %**
0	3	12×64	9×64	6.5 %	33.3 %	**21.5 %**
0	4	12×64	9×64	0.5 %	33.3 %	**19.2 %**
0	5	12×64	9×64	0.5 %	44.4 %	**25.6 %**

TAB. 7.5 Les paramètres et l'activité du premier macrobloc

Le même calcul est fait pour les autres macroblocs de l'image. L'**activité interne** moyenne qui en résulte est **20.03 %**. Avec les résultats DIESEL, la moyenne calculée était *36 %*.

– Le **nombre de portes** d'un composant est calculée à partir des équations 7.2 et 7.3, dans lesquelles n_b est le nombre d'entrées, m_b le nombre de sorties, $H_{out,b}$ l'entropie moyenne des noeuds de sortie et $N_{FF,b}$ une estimation du nombre de flipflops (pour les composants séquentiels).

$$N_{g,b} = (n_b + m_b + 4N_{FF,b}) + H_{out,b}(n_b + m_b + 4N_{FF,b})\left[log_{10}(n_b) + \frac{0.2}{log_{10}(n_b)}\right] \tag{7.2}$$

Le calcul de l'entropie moyenne des noeuds de sortie $H_{out,b}$ est fait à partir de la probabilité p_i de valoir "1" chaque noeud i. Cette probabilité est obtenue en simulation.

$$H_{out,b} = \frac{1}{m_b}\sum_{i=1}^{m_b}\left(p_i log_2\frac{1}{p_i} + (1-p_i)log_2\frac{1}{(1-p_i)}\right) \tag{7.3}$$

Le tableau 7.6 illustre l'entropie et le nombre de portes estimé pour le premier macrobloc d'une image exemple.
Le même calcul est fait pour les autres macroblocs de l'image. Plus on fait de simulations, meilleures sont les valeurs d'entropie obtenues, qui vont se stabiliser doucement. Après la simulation de plusieurs macroblocs, le **nombre de portes** moyen estimé du bloc est **10225**. Avec les résultats DIESEL, ce nombre était *7940* (flipflops inclus).

Avec ces valeurs estimées et à partir des équations présentées dans le chapitre 5 paragraphe *5.3*, nous calculons l'énergie par opération réalisée à chaque cycle actif dans l'IDCT. Nous obtenons une énergie de **21.14 pJ**. Avec la méthode à partir des paramètres DIESEL, l'énergie calculée pour la même opération est de *41.253 pJ*.

Macrobloc	Bloc	Nombre bits entrée	Nombre bits sortie	Nombre flipflops	Entropie moyenne	Nombre de portes
0	0	12×64	9×64	440	0	**3104**
0	1	12×64	9×64	440	0.2361	**5269**
0	2	12×64	9×64	440	0.3428	**6248**
0	3	12×64	9×64	440	0.3808	**6596**
0	4	12×64	9×64	440	0.5902	**8517**
0	5	12×64	9×64	440	0.6426	**8997**

TAB. 7.6 L'entropie et le nombre de portes pour le premier macrobloc

MC

Le travail de MC consiste à fournir au bloc reconstruction REC les blocs de 8x8 pixels qui ont été retrouvés dans l'image de référence grâce au vecteur de mouvement. Deux opérations sont faites :

- Le calcul des nouvelles coordonnées du bloc par rapport à sa position dans le bloc référence, et grâce au vecteur de mouvement.
- La reconstruction du bloc référence final par interpolation avec les blocs référence voisins, si celui-ci se trouve entre blocs (le vecteur de mouvement a une résolution d'un demi bloc).

La structure interne du composant est illustrée dans la figure 7.5. Le traitement est fait par blocs de 8x8 pixels.

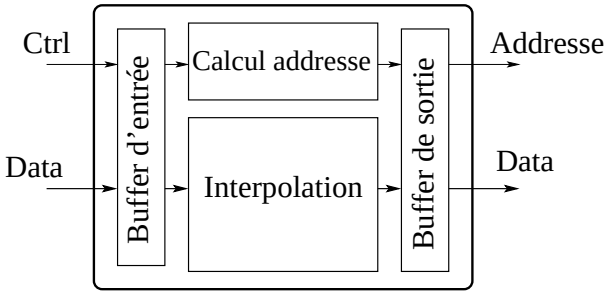

FIG. 7.5 La structure interne du MC

Tout d'abord, on lit les mots de contrôle venant du processeur qui vont définir le type de bloc et le vecteur de mouvement. Avec ces informations, on calcule les adresses mémoire où l'on va chercher le bloc entier. Pour cela, on sollicite la lecture dans la mémoire à l'adresse indiquée et suivant le protocole DTL. Une fois l'autorisation accordée, on procède à la lecture et au stockage du bloc dans le buffer d'entrée. S'il y a l'interpolation, la taille du bloc à lire est plus grande ; 8x9, 9x8 ou 9x9 pixels en fonction des directions d'interpolation. Dans ce cas, on fait l'opération d'interpolation pour reconstruire le bloc de référence. Quand le bloc de référence est disponible, on l'envoie vers le bloc REC à travers la FIFO dès que celle-ci le sollicite.

La machine d'états illustrée dans la figure 7.6, est composée de 10 macro-états, qui exécutent pour chaque bloc, la lecture des mots de contrôle, le calcul des adresses mémoire et le calcul de l'interpolation. Dans la figure, on illustre les énergies par opération accumulées par cycle selon les opérations

exécutées par transition dans chaque macro-état.

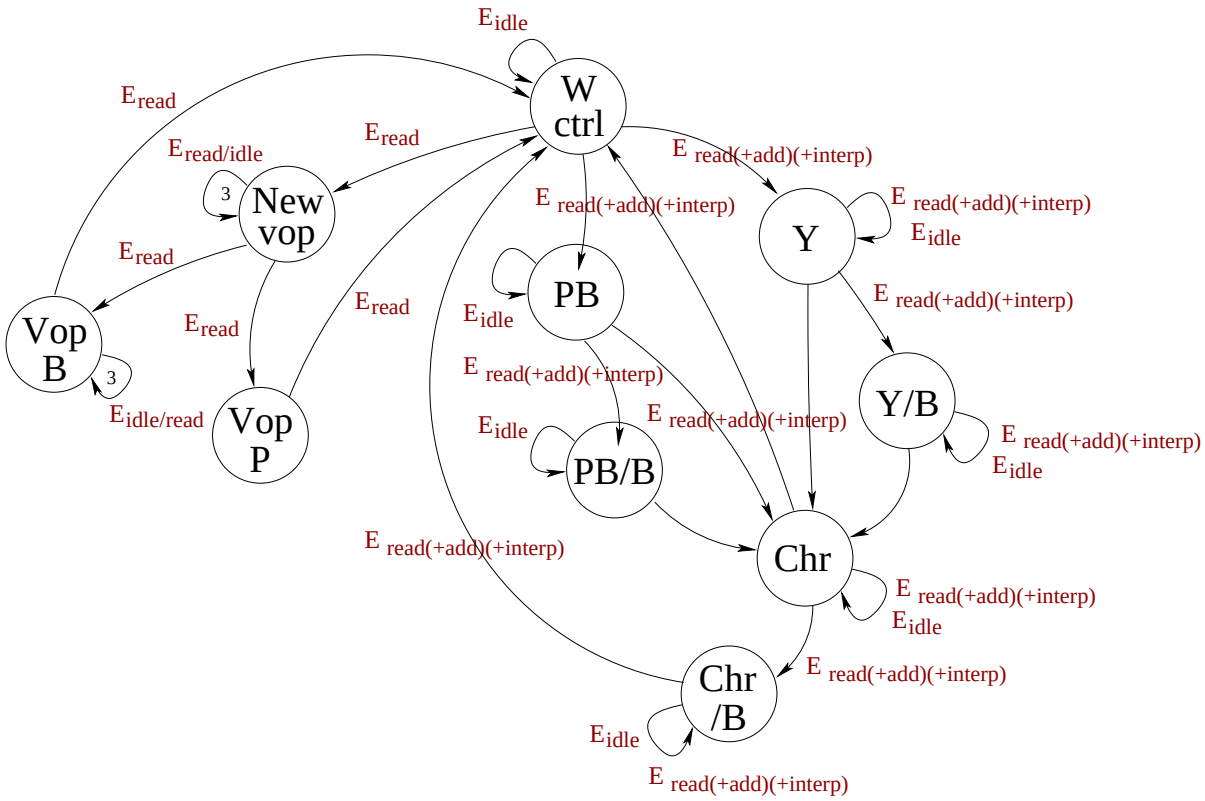

FIG. 7.6 L'automate du MC

Le modèle d'énergie est composé de quatre types d'énergie : `E_idle` ou `E_sleep`, en état d'attente ; `E_read` pour l'opération de lecture de l'extérieur et d'écriture dans les buffers (des mots de contrôle ou des pixels du bloc de référence) ; `E_address`, pour l'opération de calcul des adresses à partir du vecteur de mouvement ; et `E_interpolation`, pour l'opération d'interpolation des pixels. Ces énergies par opération sont calculées à partir des paramètres donnés au modèle et des équations 5.8 et 5.11 du chapitre 5 et sont illustrés dans le tableau 7.7.

Il nous a semblé intéressant de montrer le modèle d'énergie de MC, même s'il n'a pas été implanté car on ne disposait pas du modèle TSS fonctionnel.

REC

REC fait la reconstruction des pixels. La structure interne du composant est illustrée dans la figure 7.7.

Il est composé des deux buffers d'entrée pour stocker les pixels venant de l'IDCT et de MC, d'un bloc de saturation, d'un additionneur, d'un multiplieur et d'un buffer de sortie. Pour les images INTRA, la reconstruction consiste à regrouper les pixels de 8 bits venant de l'IDCT en mots de 32 bits. Pour les images INTER codées, on additionne d'abord les pixels de l'image précédente venant de MC, avec l'erreur de prédiction provenant de l'IDCT, puis on regroupe le résultat en mots de 32 bits.

Paramètres	Valeur
nb_gates	7089
nb_ff	785
energy_gate	10 fJ
energy_ff	53 fJ
energy_ffck	19 fJ
μ_read	10 %
μ_address	35 %
μ_interpolation	25 %
Energie par opération	**Valeur**
E_idle	15.7 pJ
E_sleep	0.8 pJ
E_read	25.4 pJ
E_address	49.8 pJ
E_interpolation	40 pJ

TAB. 7.7 Les paramètres et la consommation du MC

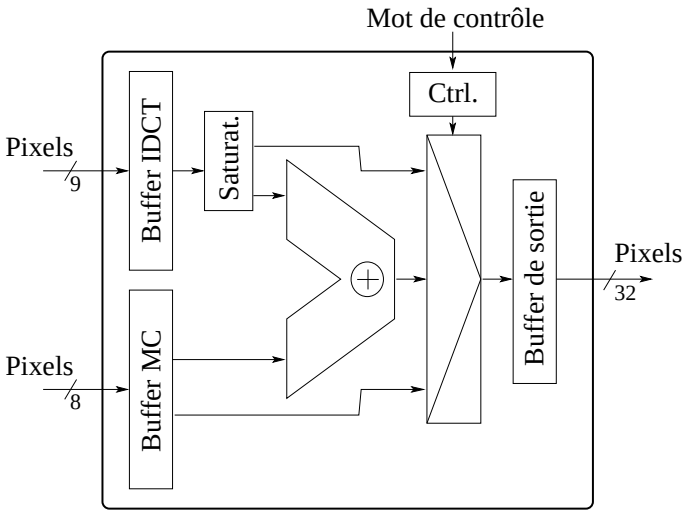

FIG. 7.7 La structure interne du REC

Une opération de saturation est appliquée sur les valeurs issues de l'IDCT qui dépassent l'échelle autorisée.

La machine d'états illustrée dans la figure 7.8, est composée de 8 macro-états, qui exécutent la lecture, la saturation, l'addition et la reconstruction des mots de 32 bits. Les macroblocs sont traités par groupes de 2 blocs (un *token*). Le premier état, W_ctrl attend et décode le mot de contrôle qui définit le type de bloc (inter/intra, bidirectionnel, codé/non codé). Une fois cela fait, on passe à l'état W_I pour atteindre les pixels venant de l'IDCT pour les images INTRA ou à l'état W_P/B pour atteindre les pixels provenant du MC et de IDCT pour les images INTER. Une fois dans ces états et dès que les pixels arrivent, on passe à l'état IL0 (INTRA) ou PL0 (INTER), pour effectuer les opérations

d'addition, reconstruction et éventuellement de saturation des pixels, de deux premiers blocs correspondants à la luminance. Puis, on fait la même chose pour les deux blocs suivants de luminance dans les états `IL1` (INTRA) ou `PL1` (INTER) et finalement pour les deux blocs de chrominance dans les états `IC` (INTRA) ou `PC` (INTER). Du point de vue de la consommation, on y trouve par transition les énergies : `E_idle` et `E_sleep` dans les transitions correspondantes à l'attente du mot de contrôle ou des pixels ; `E_read`, dans les transitions de lecture de mot de contrôle ou de lecture de pixel ; `E_sat`, dans les transitions de lecture d'un pixel en saturation ; et `E_add`, dans les transitions d'addition des pixels (INTER). Ces énergies par opération sont calculées à partir des paramètres donnés au modèle et des équations 5.8 et 5.11 du chapitre 5 et sont illustrés dans le tableau 7.8.

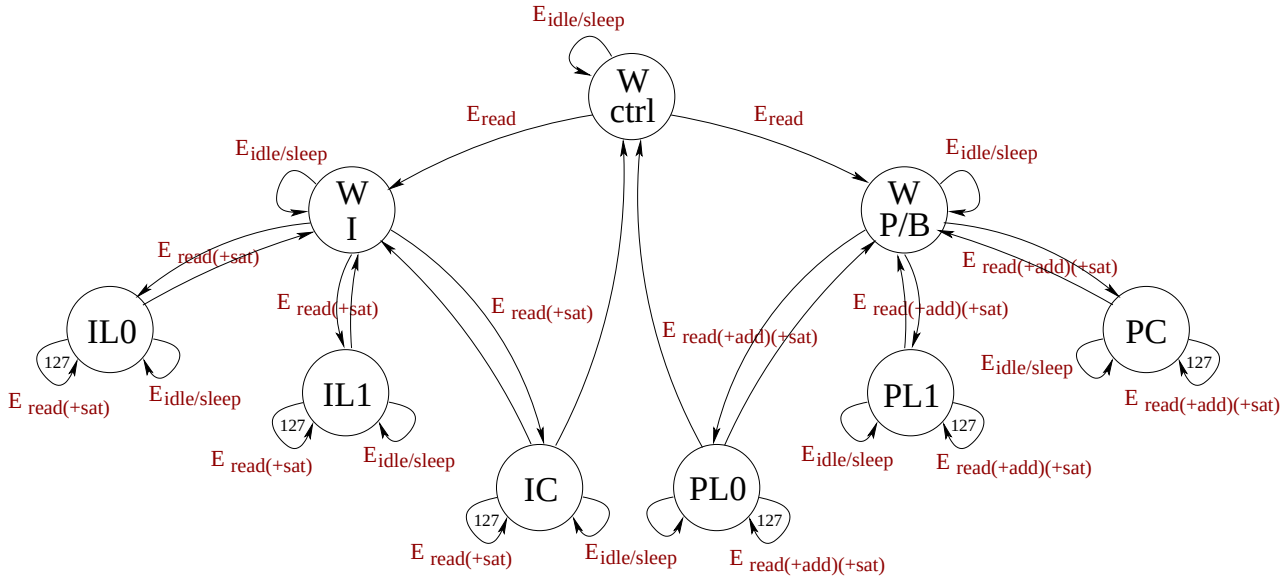

FIG. 7.8 L'automate du REC

BLW

BLW (*BLock Writer*) est un bloc qui envoie les pixels décodés depuis le bloc de reconstruction vers la mémoire SDRAM. BLW lit et décode les quatre mots de contrôle qui arrivent du CPU et calcule grâce à eux l'adresse mémoire de chaque token. Puis, il stocke les données venant du bloc REC, par token de 32 mots de 32 bits chacun. Une fois tout le token chargé, BLW communique avec la mémoire en utilisant le protocole DTL [Phi01b], pour envoyer l'adresse du token et solliciter un transfert de données. Quand la mémoire donne son accord, BLW envoie les pixels de données en paquets de 32 mots. L'automate d'état correspondant à ce comportement est illustré dans la figure 7.9.

En ce qui concerne le modèle d'énergie, on y trouve par transition les énergies : `E_idle` et `E_sleep`, quand le bloc attend les données ou le mot de contrôle ; `E_read`, quand le bloc lit une donnée ou un mot de contrôle ; et `E_oper`, quand le bloc calcule la nouvelle adresse. Ces énergies par opération sont calculées à partir des paramètres donnés au modèle et des équations 5.8 et 5.11 du chapitre 5 et sont illustrés dans le tableau 7.9.

Paramètres	Valeur
nb_gates	1130
nb_ff	114
energy_gate	10 fJ
energy_ff	53 fJ
energy_ffck	19 fJ
μ_read	38 %
μ_add	40 %
μ_sat	5 %
Energie par opération	**Valeur**
E_idle	2.3 pJ
E_sleep	0.1 pJ
E_read	8 pJ
E_add	8.4 pJ
E_sat	3 pJ

TAB. 7.8 Les paramètres et la consommation du REC

Paramètres	Valeur
nb_gates	997
nb_ff	129
energy_gate	10 fJ
energy_ff	53 fJ
energy_ffck	19 fJ
μ_read	8 %
μ_oper	13 %
Energie par opération	**Valeur**
E_idle	2.6 pJ
E_sleep	0.1 pJ
E_read	3.7 pJ
E_oper	4.3 pJ

TAB. 7.9 Les paramètres et la consommation du BLW

Dans l'annexe A se trouve le modèle fonctionnel TSS avec les valeurs d'énergie du bloc BLW et la netlist où l'on passe les valeurs des paramètres d'énergie.

7.2.5 Les interconnexions

Nous avons deux types d'interconnexions dans ce système : une première, dans le bus amba-AHB pour les accès entre composants à l'intérieur du circuit et une deuxième à travers les broches

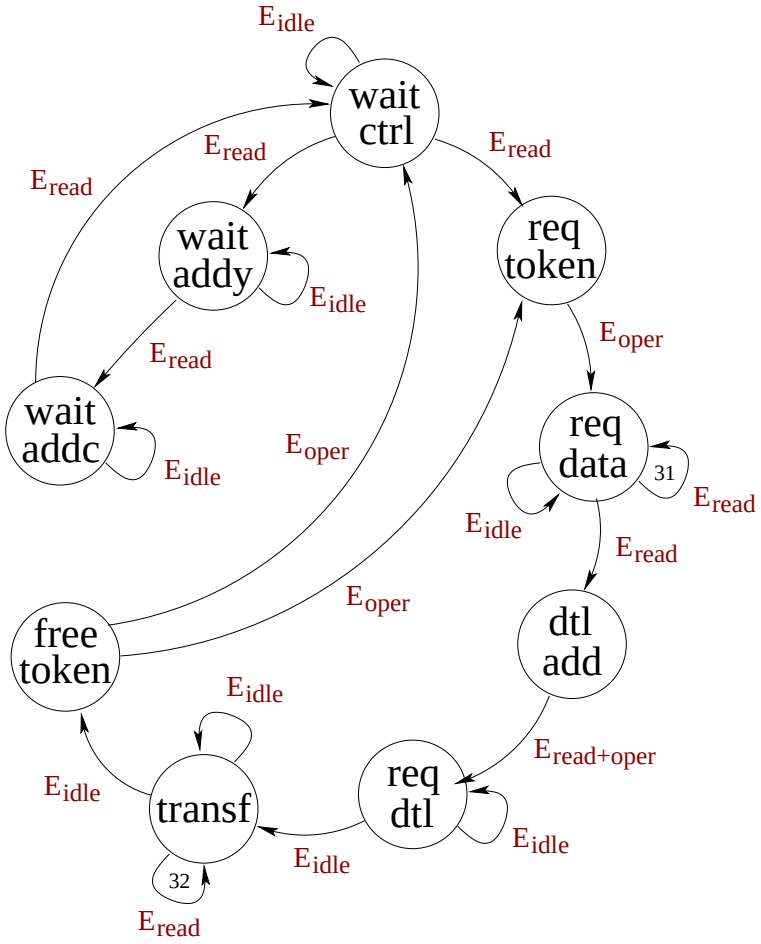

FIG. 7.9 L'automate du BLW

d'entrée/sortie et le circuit imprimé pour les accès vers la mémoire externe DDRC SDRAM. Le modèle d'énergie des interconnexions a été présenté dans le chapitre 5 et il exprime la consommation à partir du nombre de connexions transitant à chaque cycle. Les paramètres utilisés dans notre système sont montrés dans le tableau 7.10

Interconnexion	**Paramètres**	**Valeur**
Bus amba-AHB	`wire_capacity`	1.1 pF
"	`voltage`	1.2 V
PCB + pins	`wire_capacity`	10 pF
"	`voltage`	3.3 V
	Energie par opération	**Valeur**
Bus amba-AHB	`energy_line_bus`	1.6 pJ
PCB + pins	`energy_line_offchip`	108.9 pJ

TAB. 7.10 Les paramètres et la consommation des interconnexions

Dans le système, on a ajouté deux modèles TSS pour calculer l'énergie des interconnexions. Ils sont

illustrés dans la figure 7.2 comme **AHBSPY** et **DVPSPY**. Ce sont des "espions" des interconnexions qui donnent le nombre de transitions par cycle dans les connexions. Avec ces valeurs et les paramètres passés au modèle, on calcule la consommation d'énergie instantanée et cumulée.

La précision de ce modèle dépend de la précision de la valeur de capacité par fil puisque les autres paramètres de l'équation sont connus. Nos valeurs proviennent des mesures sur un autre circuit très similaire donc on estime que l'**erreur** se trouvera entre **5 et 50%**. Le modèle fonctionnel de l'espion est simple et rapide à faire ainsi que l'introduction du modèle d'énergie. J'ai nécessité un temps de modélisation de seulement une journée car je disposais déjà du modèle fonctionnel des interconnexions. Par contre, certaines erreurs de fonctionnement on été détectées dans le modèle de contrôleur de bus qui ont nécessité de deux semaines suplementaires de travail de correction.

7.3 L'évaluation de la consommation du système

Nous venons d'illustrer les modèles d'énergie appliqués au décodeur matériel-logiciel MPEG4 modélisé sur TSS. Nous avons détaillé les paramètres, les valeurs d'énergie par opération et les opérations exécutées par transition dans chaque composant du système. Pour valider notre approche, nous avons décodé des séquences d'images INTRA. Le format des images est QCIF. Chaque image de ce type est composée de 99 macroblocs. On a obtenu une estimation de la consommation par image et par macrobloc.

Nous ne pouvons pas décoder des images INTER car, comme nous avons expliqué, on ne dispose pas du modèle de simulation de l'accélérateur *Motion Compensation*. On aurait pu utiliser des images CIF puisque le décodeur en supporte leur décodage aussi, mais l'étude de leur consommation n'apporterait rien de nouveau car il est équivalent à celui sur QCIF.

7.3.1 Le décodage d'une image et d'un macrobloc

La séquence de décodage d'une succession d'images se déroule comme suit :

- Le processeur lit dans la mémoire principale (DDRC SDRAM) le programme à exécuter sur la partie logicielle du décodeur MPEG4 et le stocke dans le cache d'instructions.
- L'exécution du programme commence dans le processeur avec la lecture des pixels codés de la première image depuis la mémoire principale et l'exécution des premières opérations logicielles de décodage (VLD et ISIQ).
- Les opérations matérielles de décodage sur les accélérateurs commencent avec l'envoi des mots de contrôle du premier macrobloc depuis le processeur vers BLW et REC et les coefficients d'entrée vers l'IDCT.
- IDCT décode les pixels et les envoie en pipeline vers REC et BLW, qui envoi les blocs décodés à la mémoire. Ces opérations se poursuivent macrobloc par macrobloc, jusqu'à compléter le décodage des tous les macroblocs de l'image.
- On procède de la même façon pour les autres images jusqu'à la fin de la séquence vidéo.

Le calcul de l'énergie est fait pendant la simulation en utilisant la commande `energy`. Pour pouvoir obtenir des estimations de la consommation du décodage par macrobloc et par image, nous avons in-

troduit des points de rupture (*breakpoints*) dans le code logiciel. Cela permet d'arrêter la simulation à l'envoi du premier mot de contrôle de chaque macrobloc. Les résultats sont montrés dans les tableaux et les figures des sections suivantes.

La consommation totale d'une image INTRA

Les résultats de l'énergie totale dissipée par le décodage complet d'une image INTRA sont montrés dans le tableau 7.11. Cette consommation comprend tout le traitement de l'image, c'est à dire, dès les premières opérations d'analyse de la séquence d'images et de mise à jour de pointeurs, jusqu'au décodage des 99 macroblocs de l'image par des tâches partagées entre le logiciel et le matériel dédié. La fréquence du système est celle de la mémoire, 83 MHz. À cette fréquence, on satisfait aisément la contrainte de 15 images/s.

Composant	**Energie**
ARM	641 μJ
SDRAM	5725 μJ
IDCT	29 μJ
REC	7 μJ
BLW	8 μJ
INTERC. EXTERNES	366 μJ
INTERC. INTERNES	6 μJ
IDCT FIFO	0.8 μJ
REC FIFO	0.8 μJ
BLW FIFO	0.2 μJ
Energie totale	**6784 μJ**
Nb cycles total	3236532
Fréquence	83 MHz
Puissance totale	**174 mW**

TAB. 7.11 La consommation totale d'une image INTRA

Les accès à la mémoire et le processeur prennent 99 % de la consommation. Pour mieux analyser ces résultats, on va profiter d'une autre caractéristique de notre approche d'estimation : la capacité d'obtenir l'évolution de la consommation d'énergie dans le temps.

L'évolution de la consommation dans le temps d'une image INTRA

Pour analyser cette évolution, nous allons d'abord séparer la consommation d'une image par macroblocs, pour ensuite étudier ce qui se passe à l'intérieur d'un seul, sachant qu'à l'exception du premier, ils ont tous le même comportement. La consommation totale par image illustré dans le tableau 7.11 ne correspond pas seulement à la somme de la consommation de 99 macroblocs, car elle inclut aussi les premières opérations d'analyse de la séquence d'images et l'envoi vers BLW de 3 mots de contrôle

pour la mise à jour des pointeurs des adresses mémoire de l'image. Cela prend beaucoup de cycles qui ajoutent une consommation due à l'attente de tous les éléments.

La consommation moyenne des macroblocs est de **45.8** μJ avec des variations inférieures à 6 %. Elles sont dues aux différences dans le nombre d'accès à la mémoire venant des défauts de caches. Ces différences ne sont pas très significatives. Pour notre étude, nous prenons comme exemple le macrobloc MB8. Le tableau 7.12 illustre le nombre de cycles et la consommation en modes actifs et inactifs de ce macrobloc, ainsi que l'énergie dissipée totale de chaque composant. La figure 7.10 illustre la distribution de la consommation totale et la figure 7.11 l'énergie instantanée de chaque élément. On a groupé la consommation selon les trois groupes principaux de composants : processeur, mémoire et décodeur.

Composant	Cycles actifs	Energie act.	Cycles inactifs	Energie inact.	Energie Totale
ARM	15157	3789.3 nJ	7122	783.4 nJ	4573 nJ
SDRAM	2559	9639.4 nJ	19720	29064.6 nJ	38704 nJ
IDCT	768	31.8 nJ	21511	177.9 nJ	210 nJ
REC	474	4.5 nJ	21805	46.7 nJ	51 nJ
BLW	202	0.8 nJ	22077	53.5 nJ	54 nJ
INTER. EXT.	-	2183 nJ	-	-	2183 nJ
INTER. INT.	-	35.9 nJ	-	-	36 nJ
IDCT FIFO	384	7.7 nJ	21895	0.3 nJ	8 nJ
REC FIFO	384	7.7 nJ	21895	0.3 nJ	8 nJ
BLW FIFO	96	1.9 nJ	22183	0.3 nJ	2 nJ
TOTAL	-	15702 nJ	-	30127 nJ	**45829 nJ**

TAB. 7.12 La consommation du macrobloc MB8

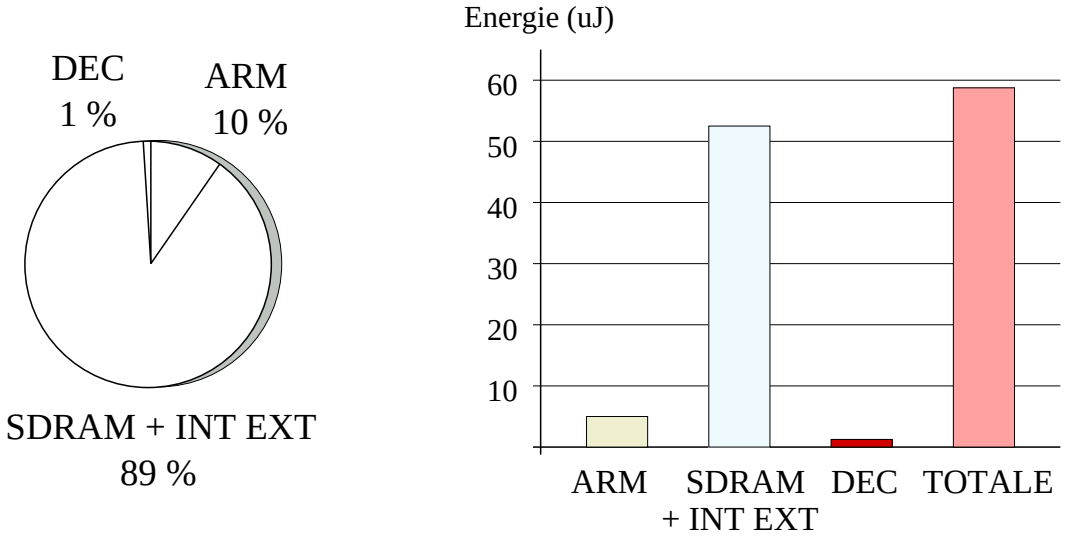

FIG. 7.10 La distribution de la consommation du macrobloc MB8

Au vu de ce tableau et de ces figures, les conclusions que l'on extrait sont les suivants :

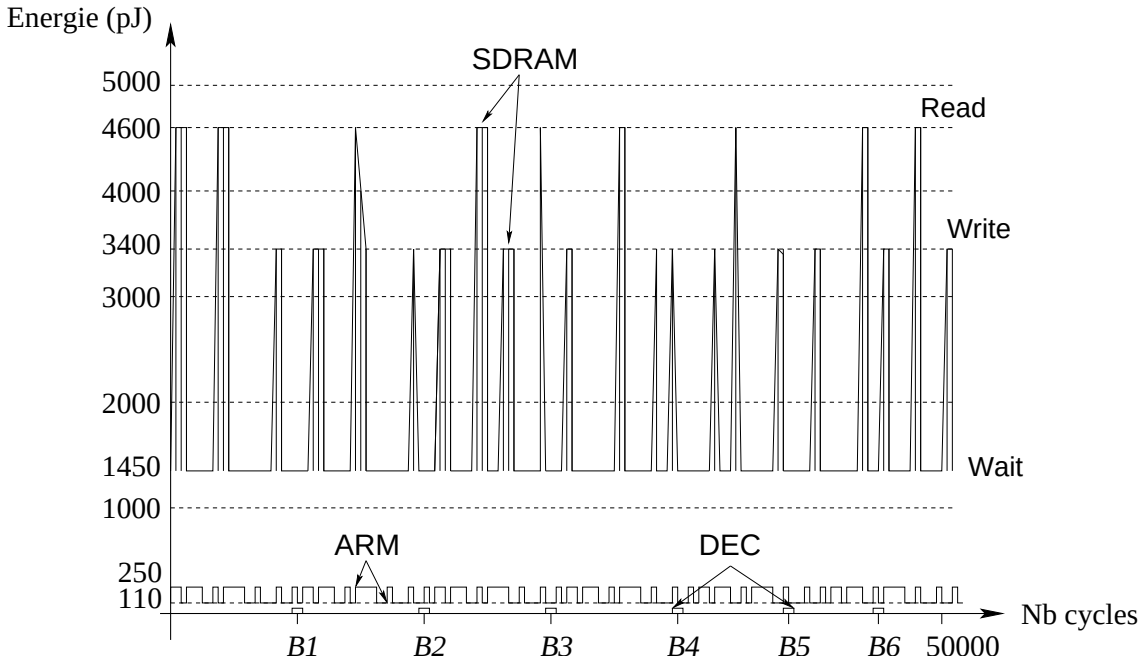

FIG. 7.11 L'énergie instantanée du macrobloc MB8

– Les accès **mémoire** (mémoire plus interconnexions externes) prennent la partie la plus importante de la dissipation d'énergie avec **89 %**. En plus, aussi **89 %** du temps, la mémoire se trouve en état *inactif* ou *active attente* et la consommation pendant ces périodes est très importante.
– Le **processeur** a une consommation non négligeable de **10 %** et il se trouve en état inactif **32 %** de temps en attente de données et d'instructions venant de la mémoire.
– Les **accélérateurs** consomment seulement **1 %** dans le système et ils se trouvent **98 %** du temps en état inactif.

Ce résultat est tout a fait explicable car la mémoire est l'élément qui consomme le plus par cycle dans cette configuration. C'est une très grande mémoire SDRAM avec une capacité électrique élevée. Les interconnexions externes coûtent aussi très cher car les pistes du circuit imprimé et les broches d'entrée/sortie des circuits ont une grande capacité électrique. Le processeur consomme une quantité importante d'énergie car il exécute beaucoup de tâches logicielles. Les accélérateurs consomment très peu par rapport aux autres éléments parce qu'ils ne sont pas nombreux, ils ne sont pas très utilisés, et ils sont très bien optimisés en surface et en consommation.

Nous observons que le système a beaucoup des *cycles d'attente* pour deux raisons principalement :

– La plupart de tâches sont exécutées en logiciel dans le processeur, ce qui fait qu'il travaille longtemps en faisant attendre le reste du système (mémoires et décodeur).
– Les lectures mémoire, même si elles sont faites en rafale (*burst*), sont très lentes. Un accès lecture en burst de 4 mots nécessite 17 cycles minimum, pour la génération et gestion de la requête en burst, le changement de protocole dans l'interface AHB-DTL et le fonctionnement interne de la mémoire SDRAM. Pendant ce temps, le processeur attend la donnée ou l'instruction avant de reprendre son travail, ce qui rajoute 32 % de temps d'attente dans tout le système.

De cette analyse, nous observons en premier que la mémoire consomme énormément et que les accès

mémoire nécessitent beaucoup de temps, ce qui ralentit le système et augmente sa consommation. Puis, on observe que le processeur travaille beaucoup et les accélérateurs ne sont pas du tout suffisants. Leur présence devrait être justifiée pour réduire la charge du processeur de façon à travailler en parallèle, plus rapidement et avec moins de consommation, ce qui n'est pas le cas. Cependant, ce résultat était prévisible car nous savions que le gain en vitesse de notre sous-système devrait atteindre seulement 10-15 % par rapport à la solution logicielle, ce qui est le cas.

D'après ces résultats, nous conseillons une nouvelle implantation du système, avec plus d'accélérateurs et un autre type de mémoire, mais aussi l'application des techniques de réduction de la consommation. Ces analyses sont montrées plus tard.

7.3.2 Evaluation de la précision du système

Une bonne méthode de validation doit être basée sur la comparaison des résultats de consommation TSS du système entier avec des mesures obtenues par un autre outil d'estimation travaillant aussi au niveau système ou bien, à partir des mesures sur une plate-forme physique, car comme on l'a vu, l'estimation basée en simulation système à plus bas niveau n'est pas possible. Malheureusement, un outil de telles caractéristiques n'est pas encore disponible et nous ne disposons pas non plus d'une plate-forme avec le circuit physique en ASIC. Le seul moyen disponible pour évaluer l'erreur de notre méthode consiste donc à comparer nos estimations, composant par composant, avec des mesures provenant d'un outil fonctionnant à plus bas niveau (niveau portes, transistors ou physique).

L'erreur de l'estimation de la consommation dépend de deux facteurs :

- L'erreur des valeurs d'énergie par opération utilisées dans le modèle d'énergie.
- L'imprécision du modèle fonctionnel cycle-précis TSS lui-même et de son interface de communication avec les autres composants du système.

L'erreur totale est la somme de ces deux facteurs :

$$Erreur\_totale = Erreur\_valeurs\_energie + Erreur\_modeleTSS$$

La modélisation de l'interface de communication entre composants dans le système doit être très précise. C'est au niveau système que l'on définit les communications et le système final doit suivre le même comportement que le système modélisé en haut niveau [Hom01]. Le modèle fonctionnel interne cycle-précis TSS devrait être aussi précis que le modèle bas niveau [JKLa04]. Cela veut dire que les états de l'automate et le comportement par cycle des deux modèles devraient être identiques. Cependant, il peut arriver que la modélisation soit *cycle-approximate* au lieu de *cycle-accurate* [Sys], ce qui signifie que, même si le comportement global du composant est identique à celui du modèle bas niveau, le comportement par cycle ne l'est pas, car les états modélisés ne sont pas les mêmes. Par exemple, le modèle TSS pourrait exécuter une opération entièrement en un seul cycle au lieu de n cycles du modèle bas niveau et rester les $(n-1)$ cycles suivants en état d'attente (*wait* cycles) avant de donner le résultat. Dans ce cas, l'erreur introduite dans l'estimation de l'énergie par le modèle fonctionnel peut être corrigée dans le modèle d'énergie si les valeurs d'énergie implantées sont adaptées au nouveau comportement.

Par ailleurs, les valeurs d'énergie par opération utilisées peuvent être aussi très précises. Cette précision dépendra de l'origine même des valeurs : si elles proviennent des mesures directes sur

le matériel, elles seront très précises ; en revanche, si ce sont des estimations faites grâce à l'analyse de l'entropie, la précision sera inférieure.

Dans notre cas, les modèles fonctionnels TSS et leur interface de communication ont la même précision que les descriptions RTL, donc cette composante de l'erreur est nulle. Nos modèles d'énergie utilisent des paramètres qui ont été calibrés à partir des mesures prises en estimation au niveau portes (accélérateurs) et à partir des mesures physiques (mémoires et processeur). L'erreur de l'estimation correspondra donc à l'erreur de ces valeurs de référence.

Dans le cas des **accélérateurs**, les paramètres utilisés ont été calibrés à partir des résultats obtenus avec l'outil d'estimation au niveau portes DIESEL. Et le modèle fonctionnel TSS utilise les mêmes états que le modèle RTL. En conséquence, les estimations TSS sont très près de celles obtenues avec DIESEL, lesquelles ont une erreur inférieure à 15 %. Si l'on calibre les modèles TSS avec des mesures encore plus précises, l'erreur diminuerait encore plus.

Les modèles d'énergie du **processeur** et des **mémoires** sont construits par macro-modélisation à partir des mesures de consommation moyenne données par les constructeurs des composants. On considère que ces mesures sont correctes. Nos modèles fonctionnels TSS suivent le même comportement par cycle des composants physiques. En conclusion, on estime que la précision des modèles est très élevée, d'environ 5 % (erreur des mesures).

Pour les **interconnexions** *off-chip*, une mesure de la capacité électrique réelle des pistes sur la carte de circuit imprimé avec le circuit n'est pas possible car nous ne disposons pas de cette carte. Par contre, nous connaissons la capacité moyenne dans d'autres cartes de circuit imprimé similaires. Pour les interconnexions *on-chip*, nous disposons aussi des capacités moyennes par fil mesurées sur d'autres systèmes similaires. Nous utilisons donc ces valeurs, qui auront une erreur estimée inférieure à 20 %.

Au vu de ces analyses, puisque nous considèrerons que notre système fonctionnel est correctement modélisé par cycle et sachant que les paramètres d'énergie proviennent des mesures de bas niveaux bien calibrées, nous calculons l'erreur de l'estimation de l'énergie du système complet à partir des erreurs des valeurs d'énergie des composants.

Dans ce cas, l'énergie totale du système avec son erreur est $(E_t \pm \delta t)$, étant E_t la somme des énergies par composant et δt l'erreur absolu. L'énergie par composant est $(E_i \pm \delta i)$, étant E_i la valeur d'énergie estimée et δi l'erreur absolue de cette valeur. Selon la théorie de la propagation des erreurs, l'erreur absolue δt d'une somme de n éléments est la somme des erreurs absolues sur chacun des termes :

$$\delta t = \sum_{i=1}^{n} \delta i$$

L'erreur par composant que nous avons calculée correspond à l'erreur relative $\delta i / E_i$. Le calcul de l'erreur relative totale est fait à partir de l'équation :

$$\frac{\delta t}{E_T} = \frac{\sum \delta i}{\sum E_i}$$

Cette erreur prend en compte non seulement l'erreur absolue par composant, mais aussi l'influence du composant sur la consommation totale du système. En appliquant ces équations sur notre système, l'erreur relative totale calculée par rapport aux mesures au niveau physique est de **6 %**.

Validation de l'erreur de l'analyse de l'entropie

Dans le cas des nouveaux accélérateurs, nous pouvons valider les estimations des paramètres d'énergie obtenues grâce à l'analyse de l'entropie en les comparant avec des mesures plus bas niveau, dans notre cas au niveau portes obtenues avec DIESEL. Dans le tableau 7.13, nous illustrons les valeurs de l'IDCT d'activité interne, nombre de portes et énergie de l'opération obtenus avec les deux méthodes et le pourcentage d'erreur trouvé dans les valeurs d'énergie.

	Activité interne	**Nombre de portes**	**Énergie par opération**
Entropie	20 %	10225	21.14 pJ
DIESEL	36 %	7940	41.25 pJ

TAB. 7.13 L'erreur de l'analyse de l'entropie sur l'IDCT

On obtient une précision dans le calcul de l'énergie de 48 %, par rapport aux mesures DIESEL, lesquelles ont une erreur inférieure à 15 %, ce qui donne une erreur totale avec le calcul de l'entropie inférieure à **72 %** par rapport au niveau physique. Ce n'est pas excellent mais cela peut être suffisant pour avoir une première idée de la répartition de la consommation dans le système, surtout puisque la composante de la consommation des accélérateurs dans un tel système (processeur, mémoire) n'est pas grande.

Cette précision demande une estimation du nombre de flipflops de l'accélérateur séquentiel. Nous avons pris la valeur exacte issue de l'implantation RTL. Une erreur de 50 % sur cette valeur produirait une erreur dans le calcul de l'énergie de 93 %.

Notre précision est estimée par rapport à l'implantation VHDL choisie à Philips. Cependant, pour d'autres implantations matérielles, cela peut changer beaucoup, car comme nous l'avons vu dans le paragraphe *7.2.4*, des changements d'arithmétique et du nombre de pipelines dans l'IDCT provoquent des grandes variations de la dissipation d'énergie. Pour augmenter la précision du calcul avec l'entropie, nous pouvons utiliser des informations sur quelques noeuds internes comme s'il s'agissait des noeuds de sortie. Mais cela implique l'adaptation du modèle à une implantation concrète. Nous ne l'avons pas fait, afin de garder l'estimation basée sur l'entropie indépendante de l'implantation.

7.4 Les techniques de réduction de la consommation

Au vu des résultats précédents, la technique de réduction de la consommation qui aurait le plus grand impact sur ce système est évidement un nouveau partitionnement avec un plus grand nombre d'accélérateurs et l'utilisation d'un autre type de mémoire. Dans le système partitionné tel qu'il est, quelques optimisations architecturales sur le système sont aussi envisageables et facilement évaluables avec notre approche d'estimation de la consommation. Elles chercheraient l'utilisation d'une autre mémoire, la diminution du nombre d'accès mémoire et des cycles d'attente des composants et de la consommation pendant les périodes inactives. Cela impliquerait une réduction de la

fréquence de certains composants et l'utilisation des modes de fonctionnement de basse consommation.

Nous avons appliqué des techniques sur chacun des composants, mais surtout sur la mémoire et le processeur. Les changements opérés sur un composant ont une influence directe sur le fonctionnement, la performance et la consommation des autres. Par exemple, le choix de la hiérarchie mémoire produit un effet direct sur la consommation des mémoires, mais aussi sur tous les autres éléments du système car cela modifiera le nombre de cycles d'attente de chacun. C'est pour cela que les optimisations en consommation doivent se faire d'une façon globale et non par composant. Notre approche d'estimation nous permet d'analyser toutes ces interactions entre composants dans le système.

En particulier, nous pouvons tester les techniques suivantes :

- **Optimisations algorithmiques et hiérarchie des mémoires** : Le nombre d'accès à la mémoire peut être réduit en optimisant l'algorithme logiciel et la hiérarchie des mémoires. L'utilisation des mémoires cache réduit déjà énormément le nombre d'accès. On peut ajouter d'autres mémoires, par exemple un second cache de données ou une mémoire ROM de programme. Les données sont analysées pour trouver lesquelles seront gardées et dans quelle mémoire de façon à optimiser la vitesse des accès et la consommation. L'approche de l'IMEC expliqué dans le paragraphe *3.5* permet de faire ce type d'optimisations [Ca00].

- **Mode endormi ou horloge inhibée** (*clock gating*) : Cette technique consiste à couper l'entrée d'horloge d'un composant quand il se trouve en état inactif [Pig04]. Cela évite la propagation des transitions inutiles à travers sa logique et la consommation sur l'entrée d'horloge des registres au bascules.

- **Réduction tension/fréquence** (*voltage/frequency scaling*) : La réduction de la tension d'alimentation Vdd permet une grande diminution de la consommation dynamique. Cependant, elle implique aussi une diminution de la vitesse de commutation des transistors [Pig04]. En conséquence, pour les systèmes synchrones, la stratégie consiste à réduire simultanément la tension Vdd et la fréquence. Cela entraîne une diminution du débit. Pour le maintenir, on applique des techniques comme le parallélisme, le pipeline ou la diminution de la tension de seuil Vth.

  Dans certains systèmes (i.e. systèmes temps réel, comme le notre), la seule réduction de la fréquence peut provoquer une réduction de l'énergie dissipée. Cela est vrai dans le cas où on n'utilise pas de clock gating et les cycles inactifs du composant ont donc une consommation dynamique associée à la ligne d'horloge et à l'entrée d'horloge des flipflops. En conséquence, si la fréquence de travail diminue, la durée des cycles actives du composant augmente (sans varier son nombre ni énergie dynamique), par contre, étant la durée totale de l'application constante, la durée des périodes inactives doit donc diminuer, ce qui implique une réduction du nombre de cycles inactives et par conséquence de leur consommation dynamique associée. La consommation statique reste constante.

- **Partitionnement et choix de l'architecture du système** : Le partitionnement d'un algorithme logiciel visant son exécution sur une architecture de système matériel-logiciel, permet l'augmentation de la vitesse de traitement et la réduction de la consommation par rapport à

l'exécution purement logicielle. Le logiciel de contrôle est exécuté sur un processeur qui commande les accélérateurs matériels. Ils fonctionnent en parallèle avec le processeur, ce qui augmente beaucoup la vitesse. En conséquence, pour un même débit, on peut réduire la fréquence et la tension, ce qui, outre que les tâches sont exécutées sur du matériel dédié bien optimisé, permet de notables réductions de la consommation.

Les techniques comme les horloges inhibées ou la réduction de la tension/fréquence nécessitent de l'utilisation d'un **gestionnaire dynamique de la puissance** (*dynamic power management*). Il sera implanté en logiciel ou en matériel et il permettra de détecter l'état des composants et d'agir sur leurs entrées de façon à optimiser leur comportement du point de vue de la consommation. Cela impliquera des arrêts d'horloges, des changements de fréquence, de tension, du mode de fonctionnement, etc., de chaque composant en fonction de son état et des besoins de l'application à chaque moment. Dans le cas du clock gating, le coût de l'implantation de ce gestionnaire est relativement faible, car il suffit d'installer un contrôleur de l'entrée d'horloge ou *chip enable* des composants et une politique de gestion des passages dans ce mode. Ce passage peut être contrôlé par données, par des signaux venant des autres composants du système ou par un module de gestion qui prend des décisions en fonction de l'états des composants et des entrées au système. La performance du système n'est pas affectée si la politique de gestion est bien faite.

Les sections suivantes expliquent l'effet de toutes ces optimisations sur la consommation sur chacun des composants et sur le système tout entier.

7.4.1 Les techniques appliquées sur la mémoire

La **mémoire SDRAM** est le composant qui dissipe le plus d'énergie dans le système, **84 %**. Les accès par les *interconnexions externes* à travers le circuit imprimé ajoutent **5 %** de dissipation. Cela fait de ce composant le premier candidat aux optimisations en consommation. Les actions possibles à mener pour réduire cette consommation sont :

- changer le type de mémoire et la hiérarchie : par exemple une SRAM-1T embarquée [Mos].
- réduire le nombre d'accès : en optimisant l'algorithme, en utilisant un autre partitionnement et en comprimant les données.
- réduire la consommation en état inactif : en utilisant les modes de basse consommation.

Nous avons testé une nouvelle mémoire, les modes basse consommation et l'utilisation de la compression sur certains données avant de les stocker. Les optimisations algorithmiques requièrent des analyses sur le code qui ne rentrent pas dans le cadre de cette thèse.

Mémoire embarquée SRAM-1T comme mémoire principale

La réduction de la consommation venant des accès à la mémoire externe peut être obtenue en utilisant une mémoire embarquée de grande taille SDRAM ou SRAM. Nous avons utilisé une mémoire SRAM-1T comme exemple [Mos]. Sa capacité maximale de 2 Mbits permet le stockage de 2 images CIF ou de 8 images QCIF, ce qui est suffisant pour le fonctionnement normal du décodeur en QCIF.

Nous disposons d'un modèle TSS de mémoire **SRAM** connectée directement au bus amba-AHB. En

ajoutant le modèle d'énergie de la mémoire SRAM-1T donné dans le paragraphe *7.2.3*, nous avons testé l'utilisation de cette mémoire comme mémoire principale. Nous illustrons dans le tableau 7.14 la comparaison de la consommation par image pour les solutions avec la SDRAM externe et avec la SRAM-1T.

Composant	**Energie SDRAM ext.**	**Energie SRAM-1T**	**Pourcentage de réduction**
ARM	641 μJ	557 μJ	-13 %
Mémoire	5725 μJ	537 μJ	-91 %
IDCT	29 μJ	23 μJ	-20 %
REC	7 μJ	6 μJ	-14 %
BLW	8 μJ	6 μJ	-25 %
INTERC. EXTERNES	366 μJ	0 μJ	-100 %
INTERC. INTERNES	6 μJ	6 μJ	0 %
IDCT FIFO	0.8 μJ	0.8 μJ	0 %
REC FIFO	0.8 μJ	0.8 μJ	0 %
BLW FIFO	0.2 μJ	0.2 μJ	0 %
Energie totale	**6784 μJ**	**1137 μJ**	**-84 %**
Nb cycles total	3236532	2469759	-24 %
Fréquence	83 MHz	83 MHz	
Puissance totale	**174 mW**	**38 mW**	**-78 %**

TAB. 7.14 La consommation totale d'une image INTRA avec la mémoire SRAM-1T

La distribution de l'énergie change complètement avec l'utilisation de la mémoire embarquée. La consommation diminue significativement, surtout dans la partie mémoire avec **-91 %**. La consommation et les délais associés aux connexions externes disparaissent. Le processeur et les accélérateurs ont eux aussi une réduction de la consommation de **-13** à **-25 %** due à la diminution globale des cycles d'attente.

En conséquence, la nouvelle distribution de la consommation est la suivante :

- Les accès **mémoire** prennent **47 %** de la dissipation d'énergie. De plus, dans le traitement d'un macrobloc, elle se trouve en état *inactif* 89 % du temps et la consommation pendant ces périodes est importante.
- Le **processeur** a une consommation de **49 %** et il se trouve en état inactif 15 % de temps en attente de données et d'instructions venant de la mémoire.
- Les **accélérateurs** consomment maintenant **3 %** dans le système et ils se trouvent **96 %** du temps en état inactif.

En résumé, le seul fait d'utiliser une mémoire embarquée au lieu de la SDRAM externe permet une réduction de la dissipation d'énergie dans le système de **-84 %**, avec une augmentation de la vitesse grâce à la réduction du nombre de cycles de **-24 %**. Le seul inconvénient est la capacité de stockage réduite de la mémoire embarquée, qui limite la taille et le nombre d'images stockées. Ce problème peut être résolu en ajoutant d'autres mémoires, par exemple, une ROM d'instructions ou une autre

SRAM de données ou en comprimant les données stockées avec la technique expliquée dans le point suivant.

Compression de données avant stockage

Une façon de réduire le nombre de données à stocker dans la mémoire consiste à les comprimer avant stockage. Dans notre système les données qui occupent le plus d'espace mémoire sont les images décodées. Si l'on réduit la taille des images à stocker, cela permettra de réduire la taille de la mémoire et le nombre d'accès.

Selon les travaux de Li [Li03], l'introduction de la compression embarquée (*embedded compression*, EC) permet la diminution de 50 % du nombre d'accès mémoire depuis les accélérateurs. Pour préserver la qualité des images, le taux de compression n'est pas grand. En plus, les accélérateurs supplémentaires ne vont pas impacter significativement la consommation et la surface silicium.

L'effet de l'introduction de cette compression dans notre système exemple sur le nombre de cycles et l'énergie dissipée est montré dans le tableau 7.15. La taille de la mémoire est réduite de moitié, nous avons donc choisi une mémoire *SRAM-1T* de 1Mbits, dont la consommation par cycle actif est *430 pJ* et par cycle inactif *90 pJ*.

	Nb cycles read	**Nb cycles write**	**Energie active**	**Nb cycles inactifs**	**Energie inactive**	**Nb cycles total**	**Energie totale**
SRAM	284980	23995	148 uJ	2160784	389 uJ	2469759	537 uJ
SRAM + EC	284980	19243	131 uJ	2165536	195 uJ	2469759	326 uJ

TAB. 7.15 La consommation de la SRAM-1T avec EC dans le décodage d'une image

L'élimination de la moitié d'écritures mémoire depuis le décodeur permet une réduction du nombre d'écritures total de -40 %, car le nombre d'écritures depuis le processeur reste invariable. Le nombre de cycles total de l'application ne change pas non plus car il est définit par le processeur. En conséquence, le nombre de cycles inactifs de la mémoire augmente légèrement.

La réduction de l'énergie totale dissipée par la mémoire est de **-40 %**. Cela est dû surtout à l'utilisation d'une mémoire plus petite. La consommation totale du système diminue de -**30 %**.

Dans le système avec le décodeur complet, la majorité des accès mémoire proviennent du décodeur et ce sont des écritures et des lectures de pixels depuis BLW et MC. Dans ce cas, la technique de compression embarquée produira une réduction du nombre d'accès mémoire qui rendra encore beaucoup plus important le gain en consommation obtenu dans le système.

Modes basse consommation sur la mémoire SDRAM

Si l'on ne dispose pas de mémoire embarquée, il y a un moyen de réduire la consommation de la mémoire SDRAM. Cette mémoire dispose de deux modes de basse consommation qui utilisent la technique d'horloge inhibée : l'un dans les périodes inactifs de la précharge, le mode `inactif`

`endormi`; l'autre pendant les périodes d'activation du banc en attente d'un accès, le mode `actif endormi`. Le passage dans ces modes est instantané et est doit contrôlé extérieurement par l'entrée *clock enable, cke*. Comme il n'y a pas de surcoût temporel, l'idéal est de rentrer dans ces modes à tous les cycles sans accès mémoire.

Nous avons implanté ces deux modes sur le modèle TSS de la mémoire, en substituant l'énergie `inactif` par `inactif endormi` et `actif attente` par `actif endormi`, illustrées dans le tableau 7.2. Le résultat du décodage d'une image est montré dans le tableau 7.16.

	Décodage image (mode normal)	**Décodage image (mode endormi)**	**Pourcentage de réduction**
Energie SDRAM	5725 μJ	3663 μJ	**-36 %**
Energie système	6784 μJ	4566 μJ	**-33 %**

TAB. 7.16 La consommation de la mémoire SDRAM avec les modes de basse consommation

La réduction de la consommation dans la mémoire et dans le système est de **- 36 %** et **- 33 %** respectivement, sans besoin de modifier l'algorithme ou l'architecture du système. Par contre, il manque le surcoût du module de gestion de passage entre modes. De toutes façons, l'utilisation de cette technique s'avère un bon moyen de réduction de la consommation dans les systèmes avec mémoire SDRAM.

7.4.2 Les techniques appliquées sur le processeur

Le **processeur** ne dissipe que **9.8 %** de l'énergie du système originaire mais par contre, **49 %** de l'énergie du système avec la SRAM-1T. Il peut donc devenir un consommateur d'énergie prépondérant et dans cette optique, nous montrons ici quelques techniques applicables sur ce dispositif :

- réduction du nombre d'instructions : en optimisant l'algorithme et surtout le partitionnement matériel-logiciel.
- réduction de la consommation en état inactif : en utilisant le mode endormi (horloge inhibée).
- réduction de la consommation en état actif : en réduisant la tension d'alimentation du processeur car la fréquence de travail utilisée est inférieure à la fréquence nominal du processeur.

Nous avons testé seulement la réduction de la tension d'alimentation du processeur car les optimisations algorithmiques ne rentrent pas dans le cadre de cette thèse et le mode endormi ne peut pas être testé car dans notre système le processeur travaille tout le temps et n'attend jamais les accélérateurs (sauf pour le remplissage du cache, pour lequel on ne peut pas appliquer cette technique). Par contre, pour d'autres systèmes, cette technique reste toujours modélisable. Les optimisations algorithmiques ne rentrent pas dans le cadre de cette thèse.

Réduction de la tension dans le processeur

La fréquence de travail utilisée dans le système est inférieure à la fréquence nominal du processeur. Pour cela, la tension d'alimentation du processeur peut être réduite. Les valeurs d'énergie utilisées

correspondent à une fréquence de 83 MHz et une tension de 1.2-1.3 V. Le processeur ARM940T implanté en CMOS13 est conçu pour travailler à une fréquence de 250 MHz avec une tension nominal de 1.3 V.

Les valeurs de tension et fréquences minimales applicables sur un processeur ARM940T fondu à 0.13 μm ne sont pas disponibles. Cependant, observant les valeurs mesurées pour le StrongARM et faisant la règle de trois, on déduit qu'à 83 MHz, la tension peut être réduite jusqu'à 0.8 V. À cette tension, l'énergie dissipée par cycle actif diminue jusqu'à 100 pJ et par cycle inactif à 44 pJ. Dans notre système avec mémoire SRAM-1T fonctionnant à 83 MHz, la consommation mesurée dans le processeur pendant le décodage d'un macrobloc en fonction de la tension et pour la même fréquence est illustrée dans le tableau 7.17.

Tension	Fréquence	Cycles actifs	Energie active	Cycles inactifs	Energie inactive	Energie totale ARM
1.3 V	83 MHz	15145	3786 nJ	2610	287 nJ	**4073 nJ**
0.8 V	”	”	1515 nJ	”	115 nJ	**1629 nJ**

TAB. 7.17 La consommation de l'ARM940T à différentes tensions

La réduction de la consommation est très importante, d'environ **-60 %** dans le processeur et **-30 %** dans le système. Cette technique a un très grand impact sur la consommation du processeur. Son implantation est donc bien justifiée. Elle demande l'implantation des convertisseurs de tension dans le circuit pour communiquer les composants fonctionnant à fréquences différentes.

La fréquence et la tension du processeur peuvent aussi être modifiées dynamiquement par un module de gestion de puissance selon les besoins du système à chaque moment. Dans ce cas, il faudra prendre en compte le surcoût de la mise en oeuvre de la solution et plus particulièrement du module de gestion car dans l'article [PLS00], on montre que les changements fréquents de fréquence/tension sur une processeur n'ajoutent pas forcement un surcoût temporel important à l'application. Dans son exemple, un *video player* MPEG fonctionnant à 30 images par seconde et dont l'ajustement de la vitesse se fera une fois par image, aura un surcoût temporel inférieur à 1%.

7.4.3 Les techniques appliquées aux accélérateurs

Les **accélérateurs matériels** du décodeur ne dissipent que **0.8 %** de l'énergie du système avec SDRAM et **3 %** du système avec SRAM-1T. Cela vient du fait qu'ils sont très optimisés en taille et en consommation et ils exécutent leurs opérations très rapidement et avec peu d'énergie. En plus dans notre sous-système, ils se trouvent en mode d'attente la plupart du temps car, comme on l'a vu, beaucoup des tâches sont exécutées en logiciel et puis ils ont été dimensionnés pour décoder des images CIF. Même si dans ce cas, il ne vaut pas la peine de réduire encore plus leur consommation, dans d'autres systèmes plus équilibrés avec plus d'accélérateurs, plusieurs techniques peuvent être utilisées pour réduire cette consommation. Elles sont les suivantes :

- réduction de la consommation des cycles inactifs : en utilisant le mode endormi.
- réduction de la consommation des cycles actifs : en réduisant la fréquence et la tension de fonctionnement.

Nous avons testé ces deux techniques sur le système avec la mémoire SRAM-1T pour montrer l'intérêt de leur application.

Mode endormi sur les accélérateurs ou clock gating

Le mode endormi des accélérateurs est celui où les horloges sont inhibées de façon à diminuer la consommation dynamique du composant en état inactif (associée à la ligne d'horloge et à l'entrée d'horloge des flipflops). Les horloges peuvent être coupées facilement et rapidement dès que le composant n'est pas utilisé. Le passage dans ce mode est donc instantané et demande un simple contrôle extérieur, qui peut être fait par donnée ou par adresse : on active l'horloge dès que la donnée ou l'adresse arrive et on le désactive deès que le traitement a fini.

L'utilisation sur TSS a été testée en substituant la consommation en état inactive ou d'attente de chacun des accélérateurs `E_idle`, par celle du mode endormi `E_sleep` (illustrés toutes dans les tableaux 7.4, 7.8 et 7.9). Le résultat du décodage d'une image est montrée dans le tableau 7.18.

	Décodage image (mode normal)	**Décodage image (mode endormi)**	**Pourcentage de réduction**
Energie IDCT	22.94 μJ	4.65 μJ	**-80 %**
Energie REC	5.64 μJ	1.17 μJ	**-79 %**
Energie BLW	6.02 μJ	0.82 μJ	**-86 %**
Energie système	1136.45 μJ	1065.08 μJ	**-6 %**

TAB. 7.18 La consommation avec le mode endormi des accélérateurs

Le pourcentage de réduction dans les accélérateurs est d'environ **-80 %**. Il est très grand dans notre exemple, mais évidemment ce taux dépend de l'architecture du système, de la longueur des périodes inactives des composants, ainsi que du surcoût de la mise en oeuvre du contrôle extérieur. Dans notre cas, on estime que le surcoût n'est pas significatif.

Un système bien dimensionné avec les composants travaillant en parallèle, ne doit pas avoir de longues périodes d'inactivité. Pour synchroniser leur fonctionnement, différentes fréquences de travail peuvent être envisagées, ce qui réduira la consommation en diminuant le nombre de cycles inactifs. Si l'on diminue en même temps la tension, les réductions seront encore beaucoup plus importantes.

Réduction de la fréquence et de la tension sur les accélérateurs

Les accélérateurs de notre système passent **98 %** du temps en état inactif. En conséquence, une façon de diminuer leur consommation dynamique est de rallonger la durée des cycles actifs en réduisant la tension d'alimentation et la fréquence de travail. Cette technique peut donner des gains très importants [Pig04].

En analysant le chronogramme de travail de chaque composant par macrobloc, on observe que l'IDCT travaille par périodes de 64 cycles (1 bloc) et ces périodes sont séparées par des intervalles de temps de

durée variable entre 2000 et 5000 cycles. Pendant ce temps, le processeur effectue de tâches logicielles et le décodeur reste en état d'attente. Les autres accélérateurs, REC et BLW travaillent par périodes de 128 cycles, espacées par encore plus de cycles (>5000 cycles). La fréquence de travail du décodeur peut être donc réduite de façon à travailler beaucoup plus lentement et de produire beaucoup moins de cycles d'attente.

Comme test, nous avons réduit la fréquence de travail de nos accélérateurs 10 fois, de 83 MHz à 8.3 MHz. Cela n'influence pas le fonctionnement des autres composants du système car le processeur travaille en parallèle une période encore plus longue que les accélérateurs (même avec leur réduction de fréquence). À cette nouvelle fréquence, on va réduire la tension de 1.2 V à 0.5 V. Dans le tableau 7.19, on illustre les valeurs de nombre de cycles et d'énergie obtenues dans le décodage d'un macrobloc, à différentes fréquences et tensions. Les modes endormis sont utilisés pendant les cycles inactifs. On considère que l'énergie statique dissipée (inactif) reste la même, même si en réalité les courants de fuites vont augmenter avec la réduction de la tension de seuil qui accompagne la diminution de la tension d'alimentation.

	Fréq. (MHz)	**Tension (V)**	**Nb cycles actifs**	**Energie active**	**Nb cycles inactifs**	**Energie inactive**	**Energie totale**	**Pourcentage de réduction**
IDCT	83	1.2	768	31.78 nJ	16987	10.70 nJ	42.48 nJ	Réf.
”	8.3	0.5	768	5.52 nJ	1007	10.70 nJ	16.22 nJ	**-62 %**
REC	83	1.2	474	4.46 nJ	17281	5.18 nJ	9.64 nJ	Réf.
”	8.3	0.5	474	0.77 nJ	1301	5.18 nJ	5.96 nJ	**-40 %**
BLW	83	1.2	205	0.81 nJ	17550	5.27 nJ	6.07 nJ	Réf.
”	8.3	0.5	205	0.14 nJ	1570	5.27 nJ	5.41 nJ	**-11 %**

TAB. 7.19 La consommation des accélérateurs du MB8 à différentes fréquences et tensions

Les réductions de consommation sont grandes, d'entre **-11 %** et **-62 %** selon le composant (en réalité un peu moins dû à l'augmentation des courants de fuites). Il manque aussi de prendre en compte le surcoût de la mise en oeuvre du contrôle extérieur (module de gestion).

Ces valeurs dépendent de l'utilisation du composant dans le système, de la fréquence choisie et de l'application. Dans cet exemple, on montre que ces techniques permettent un gain très significatif. Dans un système plus complexe, plusieurs tensions et fréquences peuvent être appliquées par groupes d'accélérateurs optimisant encore plus la consommation. De même, des changements dynamiques de tension et fréquence peuvent être envisagés en fonction de la charge instantanée du système (par exemple, par type d'image à décoder). Au niveau de la modélisation, cela demanderait de modéliser tous ces modes et leur gestion, ainsi que de prendre en compte le surcoût de la mise en oeuvre.

En conclusion, la réduction de la fréquence de travail et de la tension peut être utilisée effectivement comme technique de réduction de la consommation dans les systèmes temps réel, car elle permet des diminutions significatives de la consommation dynamique des accélérrateurs.

7.4.4 Les techniques appliquées sur les interconnexions

Les **interconnexions externes** (*offchip*) d'accès à la mémoire SDRAM consomment 5% de la consommation totale du système. Les **interconnexions internes** à travers le bus amba-AHB consomment moins de 1%.

Ces dissipations peuvent être réduites utilisant différentes techniques comme la réduction de la tension dans les fils [Ath99] ou l'augmentation de la densité d'information par fil grâce au codage des données avant l'envoi (illustré en paragraphe *7.4.1*) [Li03] ou à l'utilisation des logiques multi-valuées [Kel97].

Une façon d'éliminer complètement la dissipation des interconnexions externes est l'utilisation d'une mémoire embarquée, comme on a montré dans le paragraphe *7.4.1*. Pour les interconnexions internes aucune optimisation n'est envisagée car leur consommation est négligeable. Cependant, vu que le nombre d'interconnexions dans les circuits devient de plus en plus important surtout avec l'utilisation des réseaux d'interconnexions, il faudra en tenir compte dans le futur.

7.4.5 Récapitulatif

Les premières estimations de consommation faites sur notre système décodeur MPEG4 ont donné la répartition de la dissipation suivante :

- Les accès **mémoire** prennent **89 %** de la dissipation d'énergie et **89 %** du temps, la mémoire se trouve en état inactif avec une consommation très importante.
- Le **processeur** consomme **10 %** dans le système et il se trouve en état inactif **32 %** de temps en attente de données et d'instructions venant de la mémoire.
- Les **accélérateurs** consomment seulement **1 %** dans le système et ils se trouvent **98 %** du temps en état inactif.

Nous avons constaté que la réduction de la consommation sur le système dépend de la technique appliquée sur les composants, mais aussi des composants choisis pour les optimisations. Certains composants ont une influence considérable sur la consommation totale et sur la performance globale du système. Pour ceci, nous avons d'abord cherché l'application des techniques de réduction sur les composants les plus critiques du point de vue de la consommation, puis sur les autres. Toutes ces techniques ont été largement discutées dans le chapitre. Un récapitulatif des gains atteints est illustré dans le tableau 7.20 et la figure 7.12.

Au vu des résultats, on peut tirer les conclusions suivantes :

⇒ La substitution de la mémoire SDRAM par une **SRAM-1T** est un excellent moyen de réduction de la consommation. Si l'on utilise en plus la **compression embarquée** pour réduire la taille des images stockées et pouvoir ainsi utiliser une mémoire plus petite, la consommation du décodeur diminue encore plus, d'environ 90 % au total.

⇒ Le **mode endormi** (*clock gating*) est très intéressant pour tous les composants, surtout ceux qui ont des longues périodes inactives. L'utilisation de cette technique est spécialement recommandée pour la mémoire SDRAM (si elle est utilisée) car ce composant prend une très grande partie de la consommation du système. Sur les accélérateurs, cette technique a

Techniques de réduction de la consommation	Composants directement affectés	Réduction de l'énergie par composant	Réduction de l'énergie dans système
SRAM-1T embarquée	SDRAM Intercon. offchip Processeur Accélérateurs	**- 91 %** **- 100 %** **- 13 %** **- 20 %**	**- 84 %**
Compression embarquée	SRAM	**- 40 %**	**- 30 %**
Modes basse consommation SDRAM	SDRAM	**- 36 %**	**- 33 %**
Réduction tension processeur	Processeur	**- 60 %**	**- 30 %**
Mode endormi accélérateurs	Accélérateurs	**- 80 %**	**- 6 %**
Réduction fréquence-tension accélérateurs	Accélérateurs	**- 11 % / - 62 %**	**- 6 %**

TAB. 7.20 Les techniques de réduction de la consommation appliquées au décodeur MPEG4

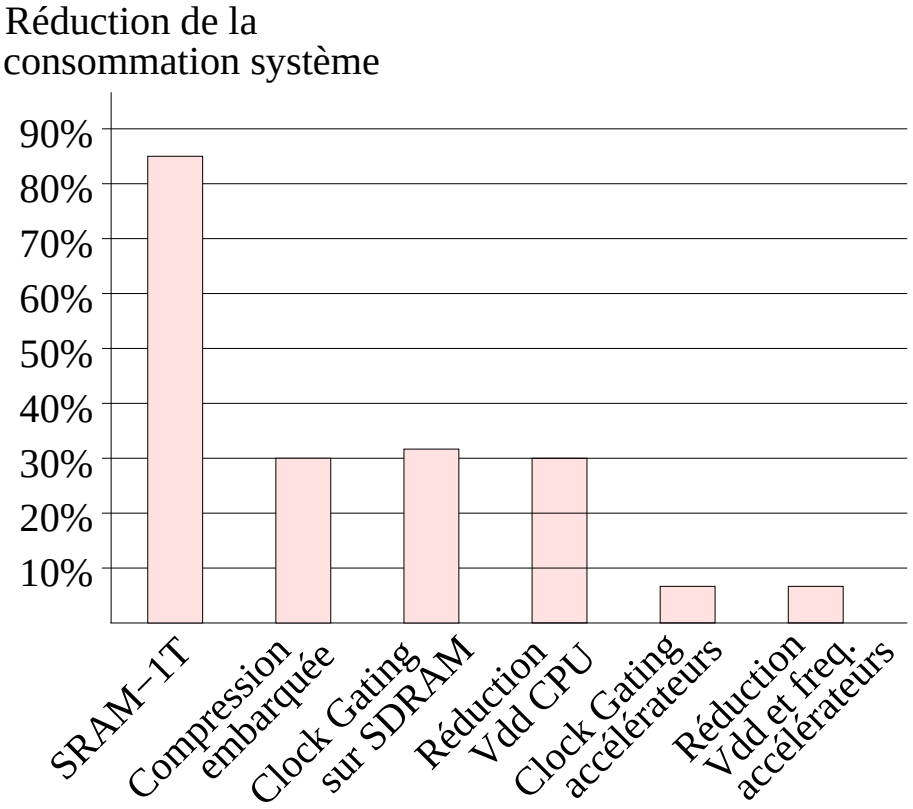

FIG. 7.12 Le pourcentage de réduction par technique

aussi un impact très significatif. Dans les systèmes ou le processeur pourrait rester bloqué en attendant des données d'un autre composant (en dehors des attentes pour cache-miss mémoire), l'utilisation de cette technique permettra aussi de grandes réductions de la consommation dans le processeur. Mais, il faudrait toujours ajouter le surcoût de la mise en oeuvre du contrôle extérieur de passage en mode endormi.

⇒ La **réduction de la fréquence et de la tension** sur les accélérateurs donne des gains très importants sur les accélérateurs (pas sur notre système). La réduction de la tension sur le processeur (quand il tourne à fréquence inférieure à la fréquence nominale) réduit aussi beaucoup sa consommation. Par contre, cette technique implique l'implantation de plusieurs tensions dans le même circuit et de convertisseurs de tension pour connecter les composants fonctionnant à des tensions différentes. Cela peut être délicat à mettre en place.

Ces techniques peuvent être combinées pour obtenir la meilleure optimisation possible. Dans ce cas, on utiliserait un système avec une mémoire SRAM-1T, la compression embarquée des pixels décodés stockés en mémoire, le processeur alimenté à une tension réduite de 0.8 V et les accélérateurs fonctionnant à une fréquence de 8.3 MHz et une tension de 0.5 V et qui en plus utilisent le mode endormi pendant les périodes inactives. Nous avons modélisé toutes ces techniques dans notre système et la réduction de la consommation du système ainsi obtenue est d'environ **93 %**. Il en manque le surcoût du contrôle extérieur de passage en mode endormi des accélérateurs quand ils ne sont pas accédés, mais on estime que dans notre cas ce surcoût ne sera pas significatif.

Finalement, un nouveau partitionnement avec plus d'accélérateurs permettrait de réduire encore plus la consommation en éliminant une grande partie de la charge du processeur. Les accélérateurs travailleraient plus et en parallèle avec le processeur. La fréquence et la tension de l'ensemble pourraient être réduites et les accès mémoire seraient aussi moins fréquents. En conséquence, le système serait plus équilibré du point de vue de l'activité des composants dans le temps, de leur performance et leur consommation.

Une autre étude possible avec notre approche serait l'analyse des pics de courant, c'est à dire la détection des périodes de puissance instantanée maximale dans le système et leur durée. Cette étude permettrait de trouver les périodes "chaudes" qui risquent d'endommager le circuit et d'analyser la meilleure façon de réduire cette dissipation de chaleur, soit avec des techniques de réduction au niveau système, soit avec l'introduction de systèmes de refroidissement.

Dans notre système, les cycles qui produisent plus de dissipation instantanée correspondent aux moments où la mémoire, le processeur et le décodeur travaillent tous ensemble. L'énergie par cycle résultante est d'environ 800 pJ. Si ces périodes durent suffisamment, la densité de chaleur par mm^2 générée peut devenir importante et des techniques appliquées dans ces périodes seraient indispensables. Une étude de la durée de ces périodes serait intéressante à faire et envisageable dans le futur.

Avec toutes ces analyses, on a montré l'intérêt de l'estimation de la consommation combinée avec la simulation au niveau système. L'importance de cette estimation vient du fait qu'elle permet l'obtention de la consommation totale d'un système, avec les interactions entre composants et les évolutions temporelles de leurs activités. Cela permet de tester l'efficacité des optimisations dynamiques sur les composants et d'analyser les effets sur chacun d'eux et sur le système entier.

7.5 Conclusion

Dans ce chapitre, nous avons présenté le système décodeur MPEG4 matériel-logiciel utilisé comme exemple d'application de notre approche d'estimation de la consommation. Nous avons montré les modèles d'énergie de chaque composant du système, avec les valeurs retenues des paramètres d'énergie et nous avons estimé la dissipation d'énergie du système grâce à la simulation TSS du décodage d'images.

Nous avons constaté que la précision des estimations de la consommation dépend de la précision de la modélisation fonctionnelle TSS et de la précision des valeurs des paramètres d'énergie. L'erreur de l'estimation de consommation sur notre système utilisant des valeurs d'énergie mesurées au niveaux portes et physique est considérée inférieure à **6 %**, ce qui est un résultat très satisfaisant. L'estimation de l'entropie peut donner les valeurs d'énergie des nouveaux accélérateurs avec une erreur pour cette partie d'environ 70 %, ce qui peut être suffisant pour une première étude de consommation, sachant que l'influence des accélérateurs sur la consommation totale du système est très faible. Plus on connaît l'implantation matérielle, plus précis seront les paramètres et meilleure sera l'estimation.

Nous nous sommes servis du décodage d'une séquence d'images INTRA pour tester l'approche et nous avons illustré la consommation d'une image et d'un macrobloc typique. Les constats que l'on peut faire sur notre système du point de vue de la consommation sont les suivants :

- ✓ Les accès **mémoire** coûtent énormément en consommation et en temps d'accès.
- ✓ Le **processeur** travaille longtemps et consomme une partie importante de la consommation.
- ✓ Les **accélérateurs** consomment très peu, surtout parce qu'ils se trouvent inactifs la plupart du temps.

Au vu de ces constats, plusieurs techniques de réduction de la consommation ont été proposées et testées. Les plus intéressantes sont les suivantes :

- ✓ L'utilisation d'une mémoire SRAM embarquée, d'une taille inférieure à celle d'une SDRAM mais avec moins de consommation.
- ✓ La compression embarquée des données avant leur stockage en mémoire.
- ✓ Le mode endormi (*clock gating*) sur tous les composants en fonction de leur activité instantanée.
- ✓ La réduction de la tension sur le processeur.

Dans un système de décodage réel avec plus d'accélérateurs, la charge entre le processeur principal et les accélérateurs serait plus répartie. Par exemple, les accélérateurs consommeraient entre 10-20 % de la dissipation d'énergie, le processeur 30-40 % et la mémoire 40-60 %. Dans ce cas, en plus des techniques déjà citées, il serait intéressant d'en appliquer d'autres sur les accélérateurs (mode endormi, réduction de la fréquence-tension, etc.) pour réduire leur partie de la consommation.

L'application conjointe de toutes ces techniques sur notre système exemple permet une réduction de **93 %** de la dissipation d'énergie, ce qui est très significatif. D'autres architectures de système et d'autres techniques peuvent être aussi testées pour optimiser davantage la consommation et la performance. Tout cela montre le formidable intérêt de cette nouvelle approche d'estimation et d'optimisation de la consommation des systèmes intégrés complexes, très précoce et rapide à faire si l'on dispose d'une bibliothèque de valeurs de consommation.

Chapitre 8

Conclusion

L'objectif de ce travail de thèse était de proposer une méthode générale d'évaluation de la consommation électrique des systèmes embarqués numériques décrits au niveau architecture de système cycle-précis. Cette estimation de haut niveau cherche à fournir des valeurs de la consommation suffisamment précises pour permettre l'exploration de l'architecture matérielle du système et l'évaluation de l'impact des techniques de réduction de la consommation existant au niveau système.

Nous avons choisi un niveau d'abstraction très élevé pour permettre l'estimation de la consommation très tôt dans le flot de conception et très rapide si l'on dispose d'une bibliothèque de valeurs d'énergie bien modélisée. Cela rend possible des optimisations en consommation précoces et d'importance considérable grâce à l'évaluation de différents scénarios. Nous pouvons agir sur l'architecture du système, sur la partie logique, sur les tâches à réaliser, etc. Désormais, nous pouvons concevoir un système embarqué complexe en vue de sa consommation depuis le début.

La méthode utilise la simulation fonctionnelle au niveau système. Elle est basée sur l'enrichissement du modèle fonctionnel avec des informations sur la consommation, ce qui évite de refaire tout un modèle juste pour y introduire la consommation. La simulation au niveau système modélise le comportement de tous les composants d'un système et de toutes leurs intercommunications, cycle par cycle, avec un haut degré de précision qui peut être similaire à celui atteint au niveau portes, mais avec des vitesses de simulation beaucoup plus importantes.

Des systèmes très hétérogènes avec des composants très différents peuvent être modélisés. L'ensemble est simulé avec les composants fonctionnant simultanément, ce qui est très difficile à réaliser aux niveaux plus avancés de la conception. Ainsi, la simulation réaliste des communications entre composants donne les activités réelles de chacun pendant l'exécution de l'application, paramètres très importants pour la consommation et qui peuvent difficilement être obtenus autrement. Cela contribue notablement à la précision des estimations de consommation de cette méthode.

Dans cet état d'esprit et pour généraliser au maximum notre approche, nous avons modélisé les composants les plus typiques d'un système embarqué : le processeur, les mémoires, les nouveaux accélérateurs matériels et les interconnexions. Pour chaque type de composant, nous avons défini un modèle d'énergie, qui a été introduit dans le modèle fonctionnel et qui comporte l'ensemble des valeurs d'énergie dissipées par chaque opération exécutée dans la transition de l'automate d'état du composant à chaque coup d'horloge.

La valeur de l'énergie par opération est calculée à partir des paramètres qui prennent en compte les fonctionnalités du composant et les modes de fonctionnement (normal et basse consommation). Cela comprend l'énergie dynamique et statique. Un système embarqué étant très hétérogène, chaque composant suit une approche différente pour le calcul numérique des paramètres d'énergie. Du point de vue fonctionnel, ces valeurs sont, soit fournies directement au modèle à partir d'une bibliothèque, soit calculées à chaque simulation en fonction des paramètres. Ces valeurs sont ensuite stockées dans des variables spéciales qui sont consultées à chaque cycle pour l'accumulation de l'énergie correspondante aux opérations courantes. De cette façon, on obtient la dissipation instantanée et totale, par composant et pour tout le système, ce qui est très intéressant pour analyser les influences entre composants et l'évolution de la consommation dans le temps.

Précision

La précision des estimations de consommation ainsi obtenues peut être très élevée, car elle dépend de la précision de la modélisation fonctionnelle cycle-précis elle-même et de la précision des valeurs d'énergie utilisées. La précision sera aussi fine que possible en fonction de la connaissance disponible sur la future implantation matérielle, car plus on aura des informations sur sa structure, plus les modèles fonctionnels haut niveau et les valeurs d'énergie seront fidèles à la réalité. Si l'on ne dispose d'aucune information, dans le cas des nouveaux coprocesseurs par exemple, l'analyse de l'entropie permettra le calcul des valeurs d'énergie avec une précision d'au moins 70 % par rapport aux mesures au niveau physique, ce qui peut être suffisant pour une première évaluation architecturale du système, sachant que la contribution en consommation des coprocesseurs dans des systèmes comme le nôtre n'est pas la plus importante.

Nous avons validé notre approche sur un exemple réel de système embarqué, dans le cadre d'une application industrielle conçue pour le marché portable. Il s'agit d'un circuit décodeur vidéo MPEG4 destiné à la téléphonie mobile. L'outil de simulation utilisé est le simulateur de systèmes TSS. Toutefois, l'approche proposée est suffisamment générale pour être utilisée dans une large gamme d'applications embarquées ainsi que dans d'autres environnements de simulation au niveau cycle-précis, par exemple, ceux basés sur SystemC. Nous avons décodé des séquences d'images et obtenu des résultats en consommation par image et par macrobloc. Les modèles fonctionnels TSS créés suivent le même comportement par cycle que les modèles RTL et les valeurs d'énergie ont été calibrées à partir des mesures aux niveaux portes et physique. Pour ces raisons, l'erreur relative moyenne de nos estimations par rapport aux mesures au niveau physique est estimée inférieure à 6 %. Cependant, il faudrait comparer nos estimations aux mesures obtenues sur une plate-forme physique avec le circuit matériel pour avoir une meilleure précision sur cette erreur.

Techniques de basse consommation

L'analyse de la dissipation sur notre système a permis de trouver des solutions architecturales qui réduisent considérablement la consommation. D'abord, nous avons trouvé que la mémoire SDRAM consomme excessivement (90 %) et que ses accès sont trop lents. Pour améliorer cela, nous l'avons remplacée par une mémoire embarqué SRAM et nous avons appliqué une technique de compression embarquée des données. Cela nous a permis un gain en consommation de 90 %. Puis, nous avons

observé que la charge sur le processeur contribue notablement à la consommation. Une façon d'y remédier est d'ajouter encore plus d'accélérateurs matériels pour répartir mieux la charge. Cependant, et sans changer le partitionnement, nous pouvons réduire cette consommation en diminuant la tension appliquée sur le processeur car il travaille à une fréquence inférieure à la fréquence nominale. Cela apporte un gain en consommation sur le processeur de 60 %. Finalement, nous avons constaté que les accélérateurs consomment très peu car ils sont inactifs la plupart du temps. Toutefois, et comme dans d'autres systèmes plus équilibrés, ils consommeront plus, nous avons appliqué sur eux plusieurs techniques de réduction. Les plus intéressantes sont l'utilisation du mode endormi (*clock gating*) ainsi que la réduction de la fréquence et de la tension. Elles permettent des gains très significatifs pouvant atteindre 90 % sur la contribution des accélérateurs.

L'application combinée de toutes ces techniques diminue la dissipation du système de 93 %. Ce résultat est très encourageant. Il montre que notre approche permet l'obtention de réductions importantes de consommation à un stade précoce de l'étude, en facilitant le choix des composants d'une architecture de système en fonction de leur consommation et de leur influence sur le système et avec une précision suffisante, que l'on peut affiner au fur et à mesure que la conception avance.

Perspectives

Plusieurs possibilités s'ouvrent en perspective comme suite à nos travaux. D'abord, une gestion dynamique de la puissance pourrait être implantée dans le système (en logiciel ou en matériel) de façon à contrôler les changements de fréquence, tension et mode de fonctionnement des composants. Il faudrait analyser le surcoût de la mise en oeuvre de cette gestion. Le contrôle de passage entre modes pourrait être fait en fonction de l'état du matériel ou des entrées, par exemple le type d'image. L'estimation de la consommation au niveau architecture de système permettrait de développer cette gestion parallèlement à la conception du système.

Par ailleurs, notre méthode pourrait être exploitée sur l'environnement de modélisation en SystemC, ce qui permettrait l'insertion de l'estimation de la consommation d'une architecture de système, dans un flot de conception qui permet la synthèse haut niveau des composants.

De plus, une bibliothèque de consommation des IP pourrait être créee, prenant en compte les paramètres technologiques, structurels, etc. de chaque composant. D'autres composants seraient aussi modélisés et ajoutés à la bibliothèque, comme les processeurs DSP, autres architectures VLIW, autres processeurs RISC, différents types de mémoires, des nouveaux systèmes d'interconnexion (reseau avec routeurs, etc.), des composants analogiques ou optoélectroniques, des cellules reprogrammables type FPGA, etc. Pour ceci, il suffirait de modéliser le comportement et l'énergie par cycle des nouveaux composants pour l'environnement de simulation choisi. On pourrait utiliser aussi la co-simulation avec d'autres simulateurs cycle-précis. Dans ce cas, l'énergie par cycle des composants serait ajoutée dans le nouveau simulateur ou dans l'interface de communication.

Une aussi vaste bibliothèque ouvrirait la possibilité d'explorer beaucoup d'autres techniques de réduction de la consommation sur ces nouveaux composants et systèmes. Par exemple, sur les nouvelles interconnexions, on pourrait tester l'utilisation des logiques multivaluées, la réduction de tension ou le codage des données. Sur les différents processeurs ou cellules reprogrammables, on étudierait les optimisations sur le code et sur la gestion de ces composants. Finalement, pour les

différents types de mémoires, on pourrait analyser des optimisations de la hiérarchie de mémoires, etc.

En plus, des nouvelles méthodes d'estimation des paramètres d'énergie inconnus peuvent être explorées pour améliorer la précision des estimations pour les composants de nouvelle conception. Pour la même raison, des études plus poussées sur les possibilités de l'analyse de l'entropie sur les composants fortement séquentiels peuvent être menées. Le modèle fonctionnel haut niveau peut être aussi mieux exploité, en y ajoutant des paramètres du modèle RTL qui améliorent la précision des estimations d'énergie. Par exemple, si l'on utilise des valeurs d'énergie calculées par l'entropie, on pourrait enrichir le modèle avec des informations fonctionnelles paramétrables en fonction de la future implantation RTL (par exemple le nombre ainsi que le type de portes et des bascules).

On pourrait aussi faire l'analyse de la précision en fonction de l'origine des valeurs d'énergie et du type de conception choisi. Dans cette optique, il faudrait établir des fourchettes d'erreur en fonction de la provenance des valeurs : calcul de l'entropie, estimations à partir d'un outil au niveau portes, au niveau transistors, mesures sur une plate-forme matérielle, etc.

Annexe A

Un modèle TSS : BLW

BLW (*BLock Writer*) est un bloc qui envoie les pixels décodés depuis le bloc de reconstruction jusqu'à la mémoire SDRAM. BLW calcule l'addresse mémoire et l'envoie à la mémoire en utilisant le protocole DTL (*Device Transaction Level*), un protocole de communication point à point développé par Philips [Phi01b].

BLW lit et décode les quatre mots de contrôle qui arrivent de la CPU, utilisés pour calculer les addresses de chaque *token*. Puis, il stocke les données venant du bloc de reconstruction, par tokens de 32 mots de 32 bits chacun. Une fois tout le token chargé, BLW envoie l'addresse à la mémoire et solicite un transfert de données. Quand la mémoire donne son accord, BLW envoie les pixels de données en paquets de 32 mots, en se servant des lignes *stride* et *line* du protocole DTL.

En relation à la dissipation d'énergie, ce bloc exécute trois types d'opérations : lecture (de données ou de mots de contrôle), calcul de l'addresse et écriture (de données). L'énergie de chacune de ces opérations constitue le modèle d'énergie du bloc. Ces énergies sont les suivantes :

- **Eidle** (**energy_idle** dans le modèle), quand le bloc attend les données ou le mot de contrôle.
- **Erw** (**energy_rw** dans le modèle), quand on lit une donnée ou un mot de contrôle.
- **Eoper** (**energy_oper** dans le modèle), quand on calcule la nouvelle addresse.

L'automate d'état avec les énergies correspondantes à chaque transition est illustré dans la figure 7.9 du chapitre 7. Les valeurs d'énergie ont été calculés avec les paramètres suivants :

- **nb_gates**, nombre estimé de portes dans le modèle.
- **nb_ff**, nombre estimé de flipflops dans le modèle.
- **energy_gate**, énergie active dissipée par une porte (nand2, à la technologie visée).
- **energy_ff**, énergie active dissipée par un flipflop (df1, à la technologie visée).
- **energy_ffck**, énergie de l'horloge dissipée par un flipflop (df1, à la technologie visée).
- **u_gate_rw**, activité estimée des portes (nb de portes commutant entre toutes les portes) nécessaire pour exécuter l'opération de lecture/écriture *rw*.
- **u_ff_rw**, activité estimée des flipflops (nb de flipflops commutant entre tous les flipflops) nécessaire pour exécuter l'opération de lecture/écriture *rw* (normalement égal à l'activité des portes).
- **u_gate_oper**, activité estimée des portes pour exécuter l'opération *oper* (calcul de l'address).
- **u_ff_oper**, activité estimée des flipflops pour exécuter l'opération *oper*.

Le nombre de portes, de flipflops et les activités associées à chaque opération sont des valeurs estimées et calibrées à partir des résultats obtenus en simulation VHDL niveau portes logiques. Ils sont montrés dans le tableau 7.9.

Ci-dessus se trouvent quelques parties du modèle TSS, plus concrètement, la déclaration des variables, des viewports et des paramètres d'énergie, la mise à jour des variables avec les valeurs passées en paramètre et grâce au calcul de l'énergie par opération, l'accumulation par cycle de l'énergie dans la machine d'états du modèle selon l'état et finalement l'affichage des valeurs d'énergie. On montre aussi la netlist où l'on passe les valeurs des paramètres d'énergie du modèle.

Description TSS de BLW

```
#include <tss.h>
#include <tssvp.h>
#include "blw.h"
.........

/* Viewports declaration */
typedef enum {
  vp_energy_value,
  vp_energy_command,/* Start/stop */
  vp_need_word,
  vp_cycle_count,
  N_VIEWPORTS
} vld_vpT;

/* Variables declaration */
typedef struct
{
  ................
  tss_float64T energy_value;
  tss_float64T energy_vld_idle;
  tss_float64T energy_vld_dc;
  tss_float64T energy_vld_dc_w;
  tss_float64T energy_vld_ac;
  tss_float64T energy_vld_ac_w;
  tss_float64T energy_vld_end_w;
  tss_strT     energy_command;
  tss_booleanT energy_calculate;  /* start=1, stop=0 */

}vld_stateT;

/* Parameters declaration */
tss_error_typeT blw_session(tss_module_handleT module,
tss_session_reasonT reason)
{
  switch (reason) {
    case TSS_SESSION_INIT:
      printf("TSS_SESSION_INIT blw\n");
        tss_declare_parameter(module,"nb_gates",
          TSS_PARAM_INTEGER,0);
        tss_declare_parameter(module,"nb_ff",
          TSS_PARAM_INTEGER,0);
        tss_declare_parameter(module,"energy_gate",
          TSS_PARAM_INTEGER,0);
        tss_declare_parameter(module,"energy_ff",
          TSS_PARAM_INTEGER, 0);
        tss_declare_parameter(module,"energy_ffck",
          TSS_PARAM_INTEGER, 0);
        tss_declare_parameter(module,"u_gate_oper",
          TSS_PARAM_INTEGER, 0);
        tss_declare_parameter(module,"u_ff_oper",
          TSS_PARAM_INTEGER, 0);
        tss_declare_parameter(module,"u_gate_rw",
          TSS_PARAM_INTEGER, 0);
        tss_declare_parameter(module,"u_ff_rw",
          TSS_PARAM_INTEGER, 0);
        break;
  .........
  }
  return TSS_OK;
}

/* Final energy value displaing */
tss_error_typeT blw_instance( tss_instance_handleT id,
tss_instance_reasonT reason)
{
  .........
    case TSS_INSTANCE_QUIT :
      printf("TSS_INSTANCE_QUIT %s\n",tss_get_instance_name(id));
      state->energy_value = state->energy_value / 1000000000000;
      tss_printf ("\t%s_total_energy = %.10f J\n",
      tss_get_instance_name(id), state->energy_value);
      break;
}

/* Parametres getting values */
static tss_error_typeT blw_instance_create
(tss_instance_handleT id)
{
  tss_int32T  nb_gates, nb_ff;
  tss_int32T  energy_gate, energy_ff, energy_ffck;
  tss_int32T  u_gate_oper, u_gate_rw;

  state->energy_value = 0.000;
  state->energy_command = "stop";
  state->energy_calculate = TSS_FALSE;
  printf("Energy_total_blw = %f \n", state->energy_value);

/* Parameters   */
/* Energy en pJ */
  nb_gates = (tss_float32T)((tss_uint32T)
    tss_get_instance_parameter(id, "nb_gates"));
  nb_ff = (tss_float32T)((tss_uint32T)
    tss_get_instance_parameter(id, "nb_ff"));
  energy_gate= (tss_float32T)((tss_uint32T)
    tss_get_instance_parameter(id, "energy_gate"));
  energy_ff = (tss_float32T)((tss_uint32T)
    tss_get_instance_parameter(id, "energy_ff"));
  energy_ffck = (tss_float32T)((tss_uint32T)
    tss_get_instance_parameter(id, "energy_ffck"));
  u_gate_oper = (tss_float32T)((tss_uint32T)
    tss_get_instance_parameter(id,"u_gate_oper"));
  u_ff_oper = (tss_float32T)((tss_uint32T)
    tss_get_instance_parameter(id, "u_ff_oper"));
  u_gate_rw = (tss_float32T)((tss_uint32T)
    tss_get_instance_parameter(id,"u_gate_rw"));
  u_ff_rw = (tss_float32T)((tss_uint32T)
    tss_get_instance_parameter(id, "u_ff_rw"));

/* Energy per operation calculation */
  state->energy_idle = (nb_ff*energy_ffck)/1000;
  state->energy_oper = (((energy_ffck+2*(energy_ff-energy_ffck)
    *u_ff_oper)*nb_ff) + (u_gate_oper*energy_gate*nb_gates))/1000;
  state->energy_rw = (((energy_ffck+2*(energy_ff-energy_ffck)
    *u_ff_rw)*nb_ff) + (u_gate_rw*energy_gate*nb_gates))/1000;

  printf("\t%s_energy_idle = %f pJ\n",tss_get_instance_name(id),
    state->energy_idle);
  printf("\t%s_energy_oper = %f pJ\n",tss_get_instance_name(id),
    state->energy_oper);
  printf("\t%s_energy_rw = %f pJ\n",tss_get_instance_name(id),
    state->energy_rw);
.........
}

/* Viewports creation */
static void blw_make_viewports (tss_instance_handleT id)
{
        blw_stateT *state = tss_get_instance_state(id);
        state->vph[vp_energy_value] = tss_vp_create (
                        id,
                        tss_value_typev(TSS_FLOAT64,0),
                        "energy_value",
                        "energy_value",
                        blw_vp_get, blw_vp_set, blw_vp_set_mon,
                        TSS_VP_ACC_R | TSS_VP_ACC_W ,
                        TSS_VP_EVENT_W,
                        vp_energy_value);
        state->vph[vp_energy_command] = tss_vp_create (
                        id,
```

```
                        tss_value_typev(TSS_STR,0),
                        "energy_command",
                        "energy_command",
                        blw_vp_get, blw_vp_set, NULL,
                        TSS_VP_ACC_R | TSS_VP_ACC_W,
                        TSS_VP_EVENT_NON,
                        vp_energy_command);
        state->vph[vp_cycle_type] = tss_vp_create (
                        id,
                        tss_value_typev(TSS_UINT32,0),
                        "cycle_type",
                        "cycle_type",
                        blw_vp_get, blw_vp_set, blw_vp_set_mon,
                        TSS_VP_ACC_R | TSS_VP_ACC_W ,
                        TSS_VP_EVENT_W,
                        vp_cycle_type);
 }

static size_t blw_vp_get(int vp_index, const tss_vpixT index[],
  tss_portT *ports, tss_raw_dataT env, tss_raw_dataT dest,
  size_t nitems)
{
 .........
        switch(vp_index) {
          case vp_energy_value :
            if (n==1) {
              *((tss_float64T*)dest) = state->energy_value;
            } else  n=0;
            break;
          case vp_energy_command :
            if (n==1) {
              *((tss_strT*)dest) = state->energy_command;
            } else  n=0;
            break;
          case vp_cycle_type :
            if (n==1) {
              *((tss_uint32T*)dest) = state->cycle_type;
            } else  n=0;
            break;
          default:
            n=0;
        } return n;
}

static size_t blw_vp_set (int vp_index, const tss_vpixT index[],
  tss_portT *ports, tss_raw_dataT env, tss_raw_dataT src,
  size_t nitems)
{
  .........
        switch(vp_index) {
          case vp_energy_value :
            if (n==1) {
               state->energy_value = *((tss_float64T*)src);
            } else  n=0;
            break;
          case vp_energy_command :
            if (n==1) {
              if (strcmp(*((tss_strT*)src),"start")==0) {
                state->energy_command =
                  malloc(strlen(*((tss_strT*)src)));
                strcpy(state->energy_command,*((tss_strT*)src));
                state->energy_calculate = TSS_TRUE;
      } else {
                if (strcmp(*((tss_strT*)src),"stop")==0) {
                  state->energy_command = malloc(strlen
                    (*((tss_strT*)src)));
                  strcpy(state->energy_command,*((tss_strT*)src));
                  state->energy_calculate = TSS_FALSE;
                } else  n=0;
              }
            } else  n=0;
            break;
          case vp_cycle_type :
            if (n==1) {
             state->cycle_type = *((tss_uint32T*)src);
            } else n=0;
            break;

          default:
            n=0;
          }
        return n;
}

/*************************************************************
 * Synchronous function
 *************************************************************/

/* Energy accumulation per cycle  */
static void fct_clock(tss_portT *ports, blw_stateT *state)
{
  .........
  switch (state->state_machine) {

  case sm_wait_ctrl :
    .........
    if (state->energy_calculate) {
      if (IPORTV(blw_ctrl_ack)) { state->cycle_rw++;
        state->energy_value = state->energy_value+state->energy_rw;}
      else { state->cycle_idle++;
        state->energy_value = state->energy_value+state->energy_idle;}
      state->cycle_type=sm_wait_ctrl;
    }
  break;

  case sm_wait_ctrl_addr_y :
    .........
    if (state->energy_calculate) {
      if (IPORTV(blw_ctrl_ack)) { state->cycle_rw++;
        state->energy_value = state->energy_value+state->energy_rw;}
      else { state->cycle_idle++;
        state->energy_value = state->energy_value+state->energy_idle;}
      state->cycle_type=sm_wait_ctrl_addr_y;
    }
  break;

  case sm_wait_ctrl_addr_c :
  .........
    if (state->energy_calculate) {
    if (IPORTV(blw_ctrl_ack)) { state->cycle_rw++;
      state->energy_value = state->energy_value+state->energy_rw;}
    else { state->cycle_idle++;
      state->energy_value = state->energy_value+state->energy_idle;}
    state->cycle_type=sm_wait_ctrl_addr_c;
  }
  break;

  case sm_req_token :
  .........
  if (state->energy_calculate) {
    state->energy_value = state->energy_value+state->energy_oper;
    state->cycle_oper++; }
  state->cycle_type=sm_req_token;
  break;

  case sm_req_data :
    .........
    if (state->energy_calculate) {
      if (IPORTV(blw_rec_ack)) { state->cycle_rw++;
        state->energy_value = state->energy_value+state->energy_rw;}
      else { state->cycle_idle++;
        state->energy_value = state->energy_value+state->energy_idle;}}
  break;

  case sm_setup_dtl_addr :
  .........
    if (state->energy_calculate) { state->cycle_oper++;
      state->cycle_rw++;
      state->energy_value = state->energy_value+
      state->energy_oper+state->energy_rw;}
  break;

  case sm_req_dtl:
    .........
  if (state->energy_calculate) { state->cycle_idle++;
    state->energy_value = state->energy_value+state->energy_idle;}
  break;

  case sm_transfer:
    .........
    if (state->energy_calculate) {
      if (IPORTV(dtlo_wr_accept)) { state->cycle_rw++;
      state->energy_value = state->energy_value+state->energy_rw;}
    else { state->cycle_idle++;
      state->energy_value = state->energy_value+
      state->energy_idle; }
      state->cycle_type=sm_transfer;
     }
  break;

  case sm_free_token:
  .........
    if (state->energy_calculate) { state->cycle_oper++;
      state->energy_value = state->energy_value+state->energy_oper;}
      state->cycle_type=sm_free_token;
  break;

  } /* End of switch (state->state_machine) */
.........
return;
}
```

Netlist de BLW

```
CLOCK (HCLK 2 0)

# BLW block
blw : BLW
[
/* Giving values to parametres on netlist */
    nb_gates    =  997,
    nb_ff       =  129,
    energy_gate =  10,  # nJ
    energy_ff   =  53,  # nJ
    energy_ffck =  19,  # nJ
    u_gate_rw   =  10,  # %
    u_ff_rw     =  10,  # %
    u_gate_oper =  15,  # %
    u_ff_oper   =  15   # %
]
(
/* Netlist interconnexions */
    HCLK            =  HCLK,
    RESET           =  reset,
    BLW_CTRL        =  blw_ctrl,
    BLW_CTRL_ACK    =  blw_ctrl_ack,
    BLW_CTRL_REQ    =  blw_ctrl_req,
    BLW_REC         =  blw_rec,
    BLW_REC_ACK     =  blw_rec_ack,
    BLW_REC_REQ     =  blw_rec_req,
    DTLO_CMD_VALID  =  BLW_REQ,
    DTLO_CMD_ACCEPT =  BLW_ACK,
    DTLO_CMD_BLOCK_SIZE = BLW_REQ_SIZE,
    DTLO_CMD_ADDR   =  BLW_ADDR,
    DTLO_CMD_READ   =  BLW_READ,
    DTLO_CMD_LINE   =  BLW_MREQ_LINE,
    DTLO_CMD_STRIDE =  BLW_MSTRIDE,
    DTLO_WR_ACCEPT  =  BLW_WVALID,
    DTLO_WR_DATA    =  BLW_WDATA,
    DTLO_WR_MASK    =  BLW_WMASK
)
```

Annexe B

Un modèle SystemC : BLW

Le modèle SystemC du bloc BLW et le fichier principal de simulation du système sont décrits ici. Dans le modèle, les variables et les paramètres d'énergie sont appelés de la même façon qu'en TSS. Les paramètres sont passés au modèle dans la déclaration de l'instance du fichier principal.

En SystemC, on ne dispose pas de commandes donc le contrôle de la simulation nécessaire pour l'estimation de la consommation est faite depuis le fichier principal à travers la variable `energy_calculate`. Dans ce fichier, on genère les coups d'horloge de la simulation. La modification de cette variable est faite selon un comptage des cycles d'horloge, ce qui permet de définir les périodes d'estimation de l'énergie. Pour plus d'information sur la façon de modéliser en SystemC se référer au manuel [Sys].

Description SystemC de BLW

```
#include <systemc.h>

////////////////////////////////////////////////////////////
// Structure definition
////////////////////////////////////////////////////////////
struct BLW : sc_module
{
.........
//  Registers (variables) declaration
  sc_signal<uint> energy_idle;
  sc_signal<uint> energy_rw;
  sc_signal<uint> energy_oper;

  sc_signal<uint> energy_value;
  sc_signal<bool> energy_calculate; //start=1, stop=0

  sc_signal<uint> cycle_idle;
  sc_signal<uint> cycle_rw;
  sc_signal<uint> cycle_oper;
  sc_signal<uint> cycle_type;
.........

////////////////////////////////////////////////////////////
// Constructor
////////////////////////////////////////////////////////////
  SC_HAS_PROCESS(BLW);

//  Energy parameters
  BLW (sc_module_name     insname,
     uint               nb_gates,
     uint               nb_ff,
     uint               energy_gate,
     uint               energy_ff,
     uint               energy_ffck,
     uint               u_gate_rw,
     uint               u_ff_rw,
     uint               u_gate_oper,
     uint               u_ff_oper)
  {
  SC_METHOD(transition);
  sensitive_pos << CLK;
  SC_METHOD(output);   // Moore
  sensitive_neg << CLK;

  NAME  = (const char*) insname;

//  Energy per operation calculation
  energy_idle=(nb_ff*energy_ffck)/1000;
  energy_rw=(((energy_ffck+2*(energy_ff-energy_ffck)*u_ff_rw/100)*nb_ff)
    +(energy_gate*nb_gates*u_gate_rw/100))/1000;
  energy_oper=(((energy_ffck+2*(energy_ff-energy_ffck)*u_ff_oper/100)*nb_ff)
    +(energy_gate*nb_gates*u_gate_oper/100))/1000;
}

////////////////////////////////////////////////////////////
// New state
////////////////////////////////////////////////////////////
void transition()
{
  .........
    switch(TARGET_FSM)
    {
      case sm_wait_ctrl :
        .........
        if (energy_calculate) {
          if (BLW_CTRL_ACK.read() == 1) {
            cycle_rw++;
            energy_value = energy_value + energy_rw; }
          else {
            cycle_idle++;
            energy_value = energy_value + energy_idle; }
          cycle_type = sm_wait_ctrl;
        }
        break;

      case sm_wait_ctrl_addr_y :
        .........
        if (energy_calculate) {
```

```
          if (BLW_CTRL_ACK.read() == 1) {
            cycle_rw++;
            energy_value = energy_value + energy_rw; }
          else {
            cycle_idle++;
            energy_value = energy_value + energy_idle; }
          cycle_type = sm_wait_ctrl_addr_y;
        }
        break;

      case sm_wait_ctrl_addr_c :
        .........
        if (energy_calculate) {
          if (BLW_CTRL_ACK.read() == 1) {
            cycle_rw++;
            energy_value = energy_value + energy_rw; }
          else {
            cycle_idle++;
            energy_value = energy_value + energy_idle; }
          cycle_type = sm_wait_ctrl_addr_c;
        }
        break;

case sm_req_token :
          .........
  if (energy_calculate) {
    energy_value = energy_value + energy_oper;
    cycle_oper++;
    cycle_type=sm_req_token;
  }
  break;

case sm_req_data :
          .........
          if (energy_calculate) {
            if (BLW_REC_REQ == 1 && BLW_REC_ACK.read() == 1) {
              cycle_rw++;
              energy_value = energy_value + energy_rw; }
            else {
              cycle_idle++;
              energy_value = energy_value + energy_idle; }
            cycle_type=sm_req_data; }
          break;

case sm_setup_dtl_addr :
          .........
  if (energy_calculate) { cycle_oper++; cycle_rw++;
    energy_value = energy_value + energy_oper + energy_rw;
    cycle_type = sm_setup_dtl_addr; }
  break;

case sm_req_dtl :
          .........
  if (energy_calculate) { cycle_idle++;
    energy_value = energy_value + energy_idle;
    cycle_type = sm_req_dtl; }
  break;

case sm_transfer:
          .........
  if (energy_calculate) {
    if (DTLO_WR_ACCEPT.read() == 1) { cycle_rw++;
      energy_value = energy_value + energy_rw; }
    else { cycle_idle++;
      energy_value = energy_value + energy_idle; }
    cycle_type=sm_transfer;
  }
  break;

case sm_free_token:
         .........
  if (energy_calculate) {
    cycle_oper++;
    energy_value = energy_value + energy_oper;
    cycle_type = sm_free_token; }
  break;
    }
  }
}

}; // End sc_module
```

Fichier principal de simulation

```
#include <systemc.h>
#include "blw_systemc.h"

////////////////////////////////////////
//  System  Architecture
////////////////////////////////////////
int sc_main (int argc, char *argv[])
{
  // Signals declaration
  sc_signal<int> signal_energy_total("signal_energy_total");

  // Giving values to parametres on netlist
  BLW blw ("blw,
           997,   // nb_gates
           129,   // nb_ff
           10,    // energy_gate (nJ)
           53,    // energy_ff (nJ)
           19,    // energyffck (nJ)
           5,     // Activity u_gate_rw (%)
           5,     // Activity u_ff_rw (%)
           15,    // Activity u_gate_oper (%)
           15");  // Activity u_ff_oper (%)


  // Open trace file
  sc_trace_file *my_trace_file;
  my_trace_file = sc_create_vcd_trace_file ("simulation_trace");

  // Chronogrammes signaux
  sc_trace(my_trace_file, blw.energy_value,   "ENERGY_VALUE");
  sc_trace(my_trace_file, signal_energy_total,"ENERGY_TOTAL");

  // Simulation
  sc_initialize ();
  signal_reset = true;
  blw.energy_calculate = false;
  next_cycle (signal_clk);
  signal_reset = false;

  for (int i = 0; i < 30; i++) next_cycle (signal_clk);

  blw.energy_calculate = true;   // Beginning of the energy count

  for (;;) {
    next_cycle (signal_clk);  // clock runs indefinitely
    signal_energy_total = blw.energy_value
    // + x.energy_value + y.energy_value + ...  All the components
  }

  // Close trace file
  sc_close_vcd_trace_file (my_trace_file);

  return EXIT_SUCCESS;
}
```

Glossaire

AMBA AHB : (Advanced High-performance Bus) bus d'interconnexion des systèmes d'ARM.

ASIC : (Application Specific Integrated Circuit) circuits dédiés à une application.

BLW : (Block Writer) bloc du décodeur MPEG4 qui envoie les pixels décodés vers la mémoire.

BCU : (Bus Control Unit) unité de gestion des requêtes sur le bus.

Burst mode : mode de transmission rapide des données vers la mémoire.

CAO : (Computer Aided Oriented) outils informatiques d'aide a la conception VLSI.

CASS : (Cycle Accurate System Simulator) outil de simulation cycle précis.

CIF : (Common Intermediate Format) format video dont la résolution est de 360 pixels/ligne x 288 lignes à 30 images/s.

Clock Gating : technique basse consommation qui consiste à couper l'horloge des parties inactives du circuit.

Cycle précis : niveau de description où l'affectation des cycles est faite.

DPD : (Deep Power Down) mode de fonctionnement basse consommation des mémoires DRAM qui consiste à desactiver tout le rafraîchissement de la mémoire quand elle n'est pas utilisée.

DPM : (Dynamic Power Management) technique de basse consommation qui consiste à contrôler de façon dynamique les modes de fonctionnement des composants (fréquence, tension, etc).

Drowsy : Mode inactif d'hibernation d'un circuit dont la tension est réduite pour diminuer les courants de fuites.

DSP : (Digital Signal Processor) processeur dédié au traitement du signal.

DVP/DTL : (Device Transaction Level) protocole de communication point à point.

DVS : (Dynamic Voltage Scaling) technique de basse consommation dont la tension et la fréquence sont contrôlées de façon dynamique.

EC : (Embedded Compression) technique de basse consommation dont les données sont codées avant d'être stockées en mémoire ce qui permet de réduire la taille mémoire.

FPGA : (Field Programmable Gate Array) circuit programmable.

IDCT : (Inverse Discret Cosinus Transform) opération du décodage MPEG4 qui fait la transformée discrète inverse du cosinus.

Inter/Intra : types de codage des images : Inter exploite à la fois les rédondances spatiales et temporelles et Intra n'exploite que la rédondance spatiale.

IPc : (Intellectual Property Core) composant électronique qui peut être implanté sur FPGA ou ASIC pour faire un produit.

ISIQ : (Inverse Quantization) opération du décodage MPEG4 de quantification inverse.

MB : (Macro Block) dans le codage MPEG, groupe de pixels constitué d'un bloc carré de 16 x 16 pixels (luminance et chrominance sous-échantillonnée).

MC : (Motion Compensation) opération du décodage MPEG4 de compensation de mouvement.

MPEG4 : (Moving Picture Experts Group, standard 4) standard ISO/IEC de compression d'images qui supporte une grande variété de formats vidéo.

PAR : (Partial Array Refresh) mode de fonctionnement basse consommation des mémoires DRAM qui consiste à rafraîchir seulement une zone de la mémoire.

PCB : (Printed Card Board) carte de circuit imprimé.

QCIF : (Quarter Common Intermediate Format) format video dont la résolution est de 180 pixels/ligne x 144 lignes à 15 images/s.

REC : (Reconstruction) opération du décodage MPEG4 qui fait la reconstruction des pixels une fois décodés.

RTL : (Register Transfer Level) niveau de description où l'affectation des registres est faite.

RISC : (Reduced Instruction Set Component) jeu d'instructions simples simplifiant les optimisations de la vitesse d'exécution.

Simulation Event-Driven : simulation pilotée par événement. Si celui-ci est l'horloge la simulation est cycle précise.

DDRC SDRAM : (Double Data Rate Synchronous Dynamic RAM memory) type de mémoire SDRAM activée sur les deux fronts de l'horloge donc fonctionnant à une fréquence double.

SRAM-1T : (Static RAM 1 transistor) type de mémoire SRAM ont la cellule de stockage utilise un seul transistor et une seule capacité.

SoC : (System on Chip) système intégré sur puce.

SOI : (Silicon On Insulator) technologie qui offre une parfaite isolation diélectrique entre la couche active des circuits et le substrat de silicium massif, ce qui permet la fabrication de composants à basse consommation et à haute fréquence.

SystemC : langage de description de matériel qui a pour objectif de modéliser des systèmes numériques matériels et logiciels à l'aide de C++.

TSS : (Tool for System Simulation) outil de simulation des systèmes cycle précis.

VHDL : (Very high speed integrated circuit Hardware Description Language) langage de modélisation de circuit.

VLD : (Variable Lenght Decoder) opération du décodage MPEG4 de décodage à longueur variable.

Annexe C

Liste des publications

Conférence	The 5th IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems, DDECS 2002, Brno, Czech Republic.
Date	April 2002.
Auteurs	Ana Abril, Jean Gobert, Thomas Dombek, Habib Mehrez and Frédéric Pétrot.
Titre	Energy Estimations in High Level Cycle-Accurate Descriptions of Embedded Systems.
Conférence	The 9th IEEE International Conference on Electronics Circuits and Systems, ICECS 2002, Dubrovnik, Croatia.
Date	September 2002.
Auteurs	Ana Abril, Jean Gobert, Thomas Dombek, Habib Mehrez and Frédéric Pétrot.
Titre	Cycle-Accurate Energy Estimation in System Level Descriptions of Embedded Systems.
Magazine	Philips Research Newsletter.
Date	December 2002.
Auteurs	Ana Abril.
Titre	Early estimation of power-consumption in systems-on-chip.
Magazine	Philips Semiconductors Word News, volume 12, number 2.
Date	April-Mai 2003.
Auteurs	Ana Abril.
Titre	Early estimation of power-consumption in systems-on-chip.
Conférence	The 8th IEEE Symposium on Low-Power and High-Speed Chips, COOLChips 2005, Yokohama, Japan.
Date	April 2005.
Auteurs	Ana Abril, Habib Mehrez, Frédéric Pétrot, Jean Gobert and Carolina Miro.
Titre	A High Level SoC Energy Analysis Method with good Accuracy using Cycle Accurate Simulation.

Conférence	SPIE Conference : Microtechnologies for the New Millennium 2005 ; VLSI Circuits and Systems, Sevilla, Spain.
Date	May 2005.
Auteurs	Ana Abril, Habib Mehrez, Frédéric Pétrot, Jean Gobert and Carolina Miro.
Titre	Architectural Energy Estimation of Embedded Systems using Cycle Accurate Simulation.
Conférence	Le 5ème journées d'études Faible Tension Faible Consommation, FTFC 2005, Paris, France.
Date	May 2005.
Auteurs	Ana Abril, Habib Mehrez, Frédéric Pétrot, Jean Gobert and Carolina Miro.
Titre	Energy Estimation and Optimisation of Embedded Systems using Cycle Accurate Simulation.

Bibliographie

[ABR01] A. Acquaviva, L. Benini, and B. Ricco. An adaptive algorithm for low-power streaming multimedia processing. In *Proceedings of the IEEE DATE, Design Automation and Test in Europe conference*, 2001.

[Abr04] A. Abril. *TSS/SystemC Power Estimation User Manual*. Philips Digital Systems Labs - Paris, version 1.0 edition, janvier 2004.

[Adv98] Advanced RISC Machines Ltd. *ARM940T Technical Reference Manual, Document number ARM DDI0144A*, 1998.

[ARC] ARC International, http ://www.arccores.com.

[ARM96] Arm software development toolkit version 2.11. Technical report, ARM, Advanced RISC Machines Ltd., 1996.

[Ath99] W. Athas. Low-power vlsi techniques for applications in embedded computing. In *Proceedings of the IEEE Alessandro Volta Memorial Workshop on Low-Power Design*, pages 14–22, 1999.

[Ato] Atomium project, http ://www.imec.be/atomium/Welcome.html. *IMEC, Interuniversity MicroElectronics Center*.

[BBB+98] M. Barocci, L. Benini, A. Bogliolo, B. Ricco, and G. De Micheli. Lookup table power macromodels for behavioral library components. In *Proceedings of the IEEE DATE, Design Automation and Test in Europe conference*, 1998.

[BBM98] L. Benini, A. Bogliolo, and G. De Micheli. Dynamic power management of electronic systems. 1998.

[BDIM03] L. Benini, M. Donno, A. Ivaldi, and E. Macii. Clock-tree power optimization based on rtl clock-gating. In *Proceedings of the IEEE 40th DAC, Design automation conference*, pages 622–627, 2003.

[BE95] A. Bellaouar and M.I. Elmasry. *Low-Power Digital VLSI Design ; circuits and systems*. Kluwer Academic Publishers, 1995.

[Ber91] G. Berry. A hardware implementation of pure esterel. In *Proceedings of the International Workshop on Formal Methods in VLSI desing*, 1991.

[BFSS02] C. Brandolese, W. Fornaciari, F. Salice, and D. Sciuto. The impact of source code transformations, software power and energy consumption. *Journal of Circuits, Systems and Computers*, 11(5) :477–502, 2002.

[BMP03] L. Benini, A. Macii, and M. Poncino. Energy-aware design of embedded memories : A survey of technologies, architectures and optimization techniques. *Transactions on Embedded Computing Systems*, 2(1) :5–32, 2003.

[Bou99] M. Bouzidi. *PhD, Estimation de la consommation de circuits CMOS numériques*. Ecole Nationale Supérieure des Télécommunications, Paris, France, 1999.

[Bul] BullDast, http ://www.bulldast.com. *PowerChecker : An integrated environment for RTL power estimation and optimization*, version 4.0 edition.

[Ca00] F. Catthoor and al. *Unified low-power design flow for data-dominated multi-media and telecom applications*. Kluwer Academic Publishers, 2000.

[CB95] A.P. Chandrakasan and R.W. Brodersen. *Low Power Digital CMOS Design*, chapter 3. Kluwer Academic Publishers, 1995.

[CCa02] M Caldari, M. Conti, and al. Instruction based power consumption estimation methodology. In *Proceedings of the 9th IEEE ICECS, International Conference on Electronics, Circuits and Systems*, pages 721–724, 2002.

[CCCa02] M. Caldari, M. Conti, P. Crippa, and al. Dynamic power management in an amba-based battery-powered system. In *Proceedings of the 6th IEEE ICECS, International Conference on Electronics, Circuits and Systems*, pages 525–528, 2002.

[Chi] ChipVision, http ://www.chipvision.com. *Orinoco : A high-level power estimation and optimization tool suite*.

[Cor00] Intel Corporation. Intel strongarm 1110 microprocessor. Brief datasheet nb. 278241-005, April 2000.

[DGG02] Rainer Dömer, Andreas Gerstlauer, and Daniel Gajski. *SpecC Language Reference Manual*. SpecC Technology Open Consortium, www.specc.org, version 2.0 edition, décembre 2002.

[dKa00] E.A. de Kock and al. Yapi : Application modeling for signal processing systems. In *Proceedings of the IEEE 37th DAC, Design automation conference*, 2000.

[Dup02] Y. Dupret. Modélisation de la consommation dans des descriptions précises au cycle. Technical report, Rapport de stage, DEA ASIME, Université Paris 6, 2002.

[Fa98] F.Arakawa and al. Sh-4 risc multimedia microprocessor. In *Proceedings of the IEEE MICRO Chips, Systems, Software, and Application*, pages 3–4, 1998.

[Gur01] K. Gururaj. Ddrc sdram controller, user manual. Technical report, RTG/EPD Group, Philips Semiconductors, 2001.

[GVH01] T. D. Givargis, F. Vahid, and J. Henkel. Trace-driven system-level power evaluation of system-on-a-chip peripherical cores. In *Proceedings of the IEEE DATE, Design Automation and Test in Europe conference*, pages 306–311, 2001.

[Hen99] J. Henkel. A low power hardware/software partitioning approach for core-based embedded systems. In *Proceedings of the IEEE 36th DAC, Design automation conference*, pages 122–127, 1999.

[HKSW03] J. Haid, G. Kaefer, Ch. Steger, and R. Weiss. Run-time energy estimation in system-on-a-chip design. In *Proceedings of the IEEE 40th ASP-DAC, Asian South Pacific-Design Automation Conference*, pages 595–599, 2003.

[HL02] J. Henkel and Y. Li. Avalanche : an environment for design space exploration and optimization of low-power embedded systems. *Transactions on VLSI Systems*, 10(4) :454–468, 2002.

[Hom01] D. Hommais. *PhD, Une méthode d'évaluation et de synthèse des communications dans les systèmes intégrés matériel-logiciel*. Université Pierre et Marie Curie, Paris, France, 2001.

[Ina] Inapac Technology, Inc., http ://www.inapac.com/index.asp. *DRAM for system-in-package (SiP) applications*.

[Inf] Infineon Technologies AG, http ://www.infineon.com/edram/. *Embedded DRAM*.

[Int03] Intel. 3rd generation itanium : power budget. In *Talk at the IEEE 340th DAC, Design automation conference*, 2003.

[JKLa04] H. Jang, M. Kang, M. Lee, and al. High-level system modeling and architecture exploration with systemc on a network soc : S3c2510 case study. In *Proceedings of the IEEE DATE, Design Automation and Test in Europe conference*, 2004.

[JLSM] N. Julien, J. Laurent, E. Senn, and E. Martin. Power consumption modeling and characterization of the ti c6201. *IEEE Micro Septembre/Octobre 2003*, 23(5).

[Kel97] G. Kelemen. *PhD, Conception des circuits intégrés pour la base consommation : méthodes comparées*. Ecole Nationale Supérieure des Télécommunications, Paris, France, 1997.

[KFBM04] N. Kim, K. Flautner, D. Blaauw, and T. Mudge. Circuit and microarchitectural techniques for reducing cache leakage power. In *IEEE Trans. VLSI, vol. 12, no. 2*, pages 167–184, Feb. 2004.

[Lau03] J. Laurent. *SofExplorer : A Power/Energy Software Consumption Estimation tool for DSP*. LESTER, Laboratoire d'Electronique des Système TEmps Réel, http ://lester.univ-ubs.fr :8080, version 1.0 edition, july 2003.

[LH98] Y. Li and J. Henkel. A framework for estimating and minimizing energy dissipation of embedded hw/sw systems. In *Proceedings of the IEEE 35th DAC, Design Automation Conference*, pages 1–6, 1998.

[Li03] Y. Li. A low-power architecture for an h264 video decoder. Technical report, Rapport de stage, DESS CISAN, Université Paris 6, Philips Digital System Laboratories Paris, 2003.

[LJSM04] J. Laurent, N. Julien, E. Senn, and E. Martin. Functionnal level power analysis : An efficient approach for modeling the power consumption of complex processors. In *Proceedings of the IEEE DATE, Design Automation and Test in Europe conference*, 2004.

[LP01] X. Liu and M. C. Papaefthymiou. A static power estimation methodologie for ip-based design. In *Proceedings of the IEEE DATE, Design Automation and Test in Europe conference*, 2001.

[LRDL00] M. Lajolo, A Raghunathan, S. Dey, and L. Lavagno. Efficient power co-estimation techniques for system-on-chip design. In *Proceedings of the IEEE DATE, Design Automation and Test in Europe conference*, pages 27–34, 2000.

[MCJM96] M. Miranda, F. Catthoor, M. Janssen, and H. De Man. Adopt : Efficient hardware address generation in distributed memory architectures. In *Proceedings of the 9th IEEE Int. Symposium on System Level Synthesis*, pages 20–25, 1996.

[Men04] Mentor Graphics, http ://www.mentor.com/lsmpoweranalyst/datasheet.html. *Lsim Power Analyst : Transistor-Level Simulation*, 2004.

[Meu99] P. Meuwissen. Tss modeling of the amba asb bus and the arm7tdmi system controller. Technical report, Nat.Lab. Technical Note 031/99, Philips Research, 1999.

[Mic01a] Micron Technical Note TN-46-03, http ://www.micron.com/products/dram/syscalc.html. *Calculating DDR memory system power*, 2001.

[Mic01b] Micron Technology Inc., http ://www.micron.com/products/dram/. *Double Data Rate (DDR) SDRAM Datasheet*, 2001.

[Mic04] Micron Technology Inc., http ://download.micron.com/pdf/datasheets/psram/ AsyncCellularRAM_16_32.pdf. *Asynchronous CellularRAM DataSheet*, 2004.

[Mir00] C. Miro. *PhD, Architecture d'un accélérateur matériel pour la composition d'objets vidéo MPEG-4*. Ecole Nationale Supérieure des Télécommunications, Paris, France, 2000.

[Mos] Mosys, http ://www.mosys.com/products/1t_sram/standard_macro_list.html. *1T-SRAM Standard Macros*.

[MPE] Technical report, Motion Picture Experts Group standard 4, http ://www.mpeg4.net.

[NN96] M. Nemani and F. N. Najm. Towards a high-level power estimation capability. In *IEEE Transactions on Computer-Aided Design, vol. 15, no. 6*, pages 588–598, June 1996.

[NS00] K. Nose and T. Sakurai. Analysis and future trend of short-circuit power. *Transactions on Computer-Aided design of integrated circuits and systems*, 19(9) :1023–1030, September 2000.

[OFF02] OFFIS Institut de recherche (Alemagne), http ://poet.offis.de. *Projet POET : Power Optimisation for Embedded sysTems*, 2002.

[PBB00] T. Pering, T. Burd, and R. Brodersen. Voltage scheduling in the lparm microprocessor system. In *Proceedings of the ISLPED, International Symposium on Low Power Electronics Design*, pages 96–101, 2000.

[Phi01a] Philips Electronic Design & Tools Group, Philips Research. *DIESEL User Manual*, version 2.5 edition, June 2001.

[Phi01b] Philips Semiconductors. *Device Transaction Level (DTL) Protocol Specification*, 2001.

[Pig04] C. Piguet. *Low-Power Electronics Design*. CRC Press, 2004.

[PLG98] R. Peset-Llopis and K. Goossens. The petrol approach to high-level power estimation. In *Proceedings of the ISLPED, International Symposium on Low Power Electronics Design*, pages 130–132, 1998.

[PLS00] J. Pouwelse, K. Langendoen, and H. Sips. Dynamic voltage scaling on a low-power microprocessor. In *Proceedings of the MMSA*, 2000.

[PRO01] C. Piguet, M. Renaudin, and T. Omnes. Low-power systems on chips (socs). In *Proceedings of the IEEE DATE, Design Automation and Test in Europe conference*, 2001.

[PSN05] C. Piguet, C. Schuster, and J. Nagel. Réduction des consommations statique et dynamique par sélection des architectures. In *Proceedings of the FTFC, 5emes journées d'études Faible Tension Faible Consommation*, 2005.

[PSZ04] M. Poncino, P. Sithambaram, and R. Zafalon. Rtl power optimisation : Concepts, tools and design experiences. In *Master Course Notes at IEEE DATE, Design Automation and Test in Europe conference*, 2004.

[RH92] F. Rocheteau and N. Halbwachs. Implementing reactives programs on circuits : A hardware implementation of lustre. In *Proceedings of the REX Workshop, Real-time, Theory in Practice*, volume 600 of LNCS, pages 195–208, 1992.

[RMMM03] K. Roy, S. Mukhopadhyay, and H. Mahmoodi-Meimand. Leakage current mechanisms and leakage reduction techniques in deep-submicrometer cmos circuits. *Proceedings of the IEEE*, 91(2) :305–327, February 2003.

[Roa03] Process integration, devices, and structures. Technical report, International Technology Roadmap for Semiconductors ITRS, 2003.

[Sak03] T. Sakurai. *Perspective of Power-Aware Electronics*. Presentation of the Center for Collaborative Research, University of Tokyo (Japon), 2003.

[SBM99] T. Simunic, L. Benini, and G. De Micheli. Cycle-accurate simulation of energy consumption in embedded systems. In *Proceedings of the IEEE 36th DAC, Design Automation Conference*, pages 867–872, 1999.

[SBT00] A. Sama, M. Balakrishnan, and F. Theewen. Speeding up power estimation of embedded software. In *Proceedings of the ISLPED, International Symposium on Low Power Electronics Design*, pages 191–196, 2000.

[SC01] A. Sinha and A.P. Chandrakasan. Jouletrack - a web based tool for software energy profiling. In *Proceedings of the IEEE 38th DAC, Design Automation Conference*, pages 220–225, 2001.

[Sim01] T. Simunic. *PhD, Energy Efficient System Design and Utilization*. Stanford University, USA, 2001.

[SSN99] C.V. Schimpfle, S. Simon, and J.A. Nossek. High-level circuit modeling for power estimation. In *Proceedings of the 6th IEEE ICECS, International Conference on Electronics, Circuits and Systems*, pages 807–810, 1999.

[Syna] Synopsys, http ://www.synopsys.com/products/mixedsignal/nanosim/nanosim_ds.pdf. *NanoSim Datasheet (TimeMill & PowerMill) : Memory and Mixed signal verification*.

[Synb] Synopsys, http ://www.synopsys.com/news/pubs/rsvp/fall97/rsvp_fall97_5.html. *PowerGate(TM) : a dynamic, simulation-based, gate-level power analysis tool*.

[Sys] SystemC Open Source, http ://www.systemc.org. *SystemC Version 2.0.1 User's Guide*.

[The98] F. Theeuwen. Tss : System simulation at philips research. In *Talk at the MEDEA Workshop on System Simulation*, 1998.

[TMW94] V. Tiwari, S. Malik, and A. Wolfe. Power analysis of embedded software : A first step towards software power minimization. *Transactions on VLSI Systems*, 2(4) :437–445, 1994.

[TSS01] Tss 3.2 user manual. Technical report, Philips Electronic Design and Tools Group, Philips Research, 2001.

[Tur00] A. Turier. *PhD, Etude, conception et caractérisation des mémoires CMOS, faible consommation, faible tension en technologies submicroniques*. Université Pierre et Marie Curie, Paris, France, 2000.

[Uni] University of Berkeley (USA), http ://bwrc.eecs.berkeley.edu/Classes/IcBook/SPICE/. *Spice Manual*, version spice3f edition.

[Usa98] K. Usami. Design methodology of ultra low-power mpeg-4 codec core exploting voltage scaling techniques. In *Proceedings of the IEEE 35th DAC, Design Automation Conference*, 1998.

[vB00] K. van Berkel. *Low Power by Design*. Philips Centre for Tecnical Training cours, 2000.

[VS94] S. R. Vemuru and N. Steinberg. Short circuit power dissipation estimation for cmos logic gates. *IEEE Transactions on Circuits and Systems*, 41 :762–765, November 1994.

[YVKI00] W. Ye, N. Vijaykrishnan, K. Kandemir, and M. J. Irwin. The design and use of simplepower : A cycle-accurate energy estimation tool. In *Proceedings of the IEEE 37th DAC, Design Automation Conference*, pages 340–345, 2000.