Chapter 1

Estimation de la consommation d'énergie dans les MPSoC

1.1 Introduction

Dans le chapitre précédent, nous avons montré l'intérêt de la modélisation au niveau transactionnel (PVT) pour l'accélération de la simulation des MPSoC. Dans ce dernier chapitre, nous nous sommes basés uniquement sur le critère de performance pour évaluer les différentes alternatives de l'espace de solutions architecturales. Du fait de l'augmentation de la densité d'intégration des circuits dans les SoC, il est indispensable de prendre en compte la consommation comme critère de conception d'un système au même titre que la vitesse. En effet, avec des fréquences d'horloge de plus en plus élevées, une quantité de ressources matérielles de plus en plus grande et un degré d'intégration de plus en plus important, les circuits se rapprochent des limites physiques et thermiques supportables. Par conséquent, l'exploration d'architectures dans le flot de conception des systèmes sur puce nécessite des outils capables d'évaluer la consommation d'énergie à différents niveaux d'abstraction. Ceci permet de prendre des décisions à chaque niveau pour réduire l'espace des solutions architecturales. Dans ce chapitre, l'objectif que nous nous sommes fixé est de concevoir des outils permettant une estimation de la consommation statique et dynamique de l'énergie. Ces outils sont destinés pour être utilisables dans les environnements de conception de MPSoC aux niveaux CABA et PVT.

Comme, toutes les unités de la plateforme (processeur, hiérarchie mémoire, et réseau d'interconnexion) contribuent dans cette consommation, il devient important dans un souci de précision d'estimer la contribution de chaque unité. L'un des objectifs de notre travail est donc, de développer une méthodologie permettant la conception de modèles de consommation pour ces composants. Dans cette méthodologie nous tenterons de regrouper à la fois les critères de précision et de flexibilité.

Le mot "consommation" est souvent utilisé pour désigner à la fois la dissipation de puissance et la dissipation de l'énergie. 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 d'un système, il est nécessaire de connaître avec précision la puissance dissipée par ce système. A l'opposé, 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 (P) par l'application mais également au temps d'exécution (T)

de l'application. Ainsi, l'énergie est calculée par ($E = P\ x\ T$) [8]. Dans ce document, nous utilisons le terme consommation dans le contexte des systèmes embarqués pour indiquer la dissipation d'énergie.

Ce chapitre est structuré comme suit. La deuxième et la troisième sections traitent l'estimation de la consommation d'énergie respectivement aux niveaux CABA et PVT. Notre méthodologie pour le développement des modèles de consommation à différents niveaux d'abstraction est présentée dans la section 4. La section 5 de ce chapitre détaille la mise en œuvre de notre méthodologie. Dans cette section, plusieurs modèles de consommation pour les composants, tels que le processeur, les mémoires, le réseau d'interconnexion crossbar, et l'accélérateur matériel TCD-2D, sont présentés. La section 6 présente les résultats de simulation et une comparaison de la précision de l'estimation de la consommation aux niveaux PVT et CABA est réalisée. La réutilisation de l'environnement pour réaliser une exploration architecturale est aussi présentée dans cette section. En fin, un aperçu sur l'intégration des modèles de consommation dans notre environnement Gaspard est donné dans la section 7.

1.2 Estimation de la consommation d'énergie au niveau CABA

L'évaluation des systèmes MPSoC nécessite des outils permettant une estimation précise de la consommation au niveau de la micro-architecture. Ces outils seront utiles pour réaliser la comparaison des différentes solutions dans une exploration architecturale. Ils doivent être flexibles et modulaires pour découpler la partie traitement de la partie communication et pour permettre une exploration locale au niveau d'un composant. Dans ce dernier cas, il s'agit de trouver les paramètres les plus adéquats de la micro-architecture d'un composant. L'estimation à ce niveau doit permettre, par ailleurs, d'analyser un système tout entier dans un temps raisonnable.

Dans ce paragraphe nous détaillons la méthodologie d'estimation de la consommation au niveau architectural en se basant sur une simulation au cycle précis dans les systèmes multiprocesseurs. La consommation d'énergie d'une application exécutée sur une plate-forme est égale à la somme des consommations dans les différents composants. Dans le chapitre précédent, nous avons détaillé la description (sous forme de FSM) des composants au niveau CABA. Dans ces FSM, à chaque cycle une transition est réalisée, même s'il n'y a pas de changement d'état. De ces transitions résulte une consommation due à l'exécution des opérations associées au nouvel état. Ces opérations peuvent être de typesă: lecture, écriture, attente, calcul, etc. Une estimation suffisamment précise de la consommation revient par conséquent à identifier l'énergie dissipée par transition dans chaque FSM. La figure 1.1 illustre la méthodologie d'estimation de la consommation au niveau CABA pour un composant donné. Notre composant x est décrit avec une FSM à 4 états. A chaque transition vers un de ces états, nous associons un coût énergétique qui représente la consommation du composant à un cycle donné. Une autre approche consiste à accumuler les consommations de chaque cycle durant la simulation afin de calculer l'énergie totale.

Comme nous nous intéressons à la consommation dans chaque composant du MPSoC, l'objectif est de développer les modèles de consommation correspondants aux unités et les intégrer dans un simulateur qui prend en compte les paramètres architecturaux et les paramètres technologiques. Cette approche offre plusieurs avantages :

- Localiser l'unité fonctionnelle qui consomme le plus d'énergie. Ce qui permet au concepteur de proposer une nouvelle architecture de cette unité pour réduire ainsi

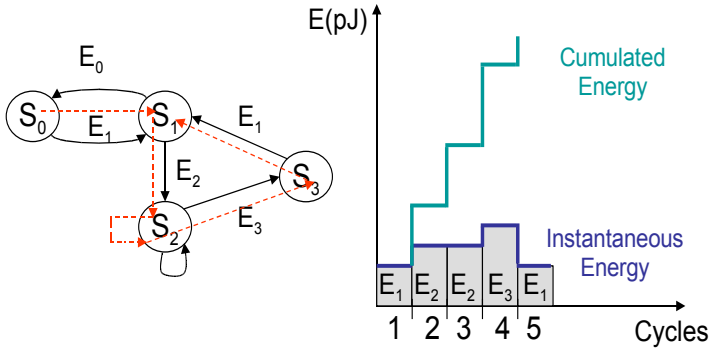


Figure 1.1: *Automate d'états et estimation de la consommation d'un composant x*

l'énergie consommée.

- Evaluer le coût en consommation d'énergie de plusieurs alternatives architecturales afin d'implémenter au niveau physique celle la plus prometteuse.

Deux facteurs essentiels interviennent dans la détermination de la précision d'estimation d'énergie au niveau CABA. Le premier facteur est lié à la description du composant. En effet, il faut réussir à identifier les opérations qui sont exécutées au cours de chaque transition des FSM décrivant le composant et pouvoir les comptabiliser. L'exécution de ces opérations est toujours accompagnée par un changement dans l'état interne du composant, ce que nous appelons une activité. Chaque activité est considérée comme une source de consommation d'énergie à laquelle un compteur est associé. Plusieurs activités peuvent s'additionner pour définir la consommation totale d'un état de la FSM. Le deuxième facteur essentiel pour garantir la précision de l'estimation d'énergie est lié à l'évaluation du coût énergétique de chaque activité. Cette phase sera détaillée dans la section 1.4.

Dans notre travail, nous avons identifié les activités pertinentes dans chaque composant et un compteur a été attribué à chaque activité du composant. Au cours de la simulation, le compteur est incrémenté si l'état correspondant est valide. A la fin de la simulation, par le biais de ces compteurs, le nombre d'occurrences de chaque activité est obtenu. En plus des compteurs, un coût énergétique, calculé à partir des modèles que nous allons développer, est attribué à chaque activité. Cette phase sera détaillée pour chaque composant dans la section 1.5.

Ainsi, la consommation totale de l'application est déterminée à partir de l'équation 1.1 suivante:

$$E = \sum_{i} N_i \times C_i \tag{1.1}$$

N_i: Nombre de fois où l'activité i est réalisée
C_i: Coût d'une unité de l'activité i

La figure 1.2 détaille notre stratégie d'estimation. Les paramètres de l'architecture (nombre de processeurs, taille du cache, etc.) sont spécifiés avant la simulation. Au cours de l'exécution de l'application, les valeurs des compteurs sont transmises au simulateur de la consommation. Ces valeurs permettent de calculer la dissipation d'énergie par cycle ou la dissipation totale à la fin de la simulation. Le simulateur de consommation contient un modèle d'énergie pour chaque composant qui dépend des paramètres de l'architecture et des

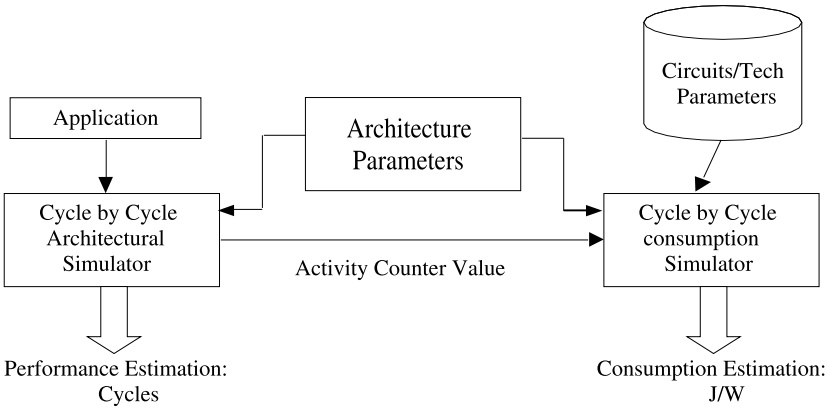


Figure 1.2: *Estimation de la consommation au niveau CABA*

paramètres technologiques. Cette approche sera appliquée sur notre plateforme au niveau CABA **??**.

1.3 Estimation de la consommation d'énergie au niveau PVT

Au niveau PVT, plusieurs obstacles rendent difficile l'opération d'estimation de la consommation d'énergie. Le premier obstacle concerne les détails de la micro-architecture qui sont absents dans ce niveau de la modélisation, ce qui rend impossible la récupération de certains types d'activités. Le deuxième obstacle est dû à l'absence de l'horloge dans la description au niveau PVT. En effet, la notion de cycle n'existe plus et les activités s'exécutent simultanément de façon atomique comme les transactions. Face à ces défis, nous proposons une stratégie d'estimation de la consommation qui se base sur la récupération des informations de bas niveau afin de les combiner et de produire une estimation dans le niveau supérieur, représenté ici par le niveau transactionnel. D'une façon similaire au niveau CABA, l'estimation de la consommation au niveau PVT, revient à estimer la dissipation dans chaque composant de notre MPSoC. La formule de l'équation 1.1 est aussi valide au niveau PVT.

Néanmoins, la granularité des activités pertinentes est plus grossière que celle au niveau CABA. En général, pour pouvoir estimer la consommation d'énergie à un niveau de description donné, il est nécessaire d'utiliser les activités présentes dans ce niveau. A titre d'exemple, au niveau CABA, il est envisageable de récupérer les occurrences des activités dont la granularité correspond à la micro-architecture. A l'opposé, pour estimer la consommation au niveau RTL, la connaissance des valeurs des compteurs de ces activités n'est pas pas suffisante puisque la granularité se situe cette fois-ci au niveau des des éléments séquentiels (bascules, registre, etc.).

Dans notre stratégie nous nous sommes basés sur les modèles de consommation du niveau CABA pour déduire ceux du niveau PVT et obtenir ainsi le maximum de précision. Les différents modèles de consommation pour les composants du MPSoC au niveau PVT ont été intégrés dans un simulateur de consommation qui interagira avec le simulateur d'architecture. La figure 1.3 détaille notre stratégie d'estimation au niveau PVT. Nous pouvons remarquer que l'approche présentée ici est assez similaire à ce qui a été proposé

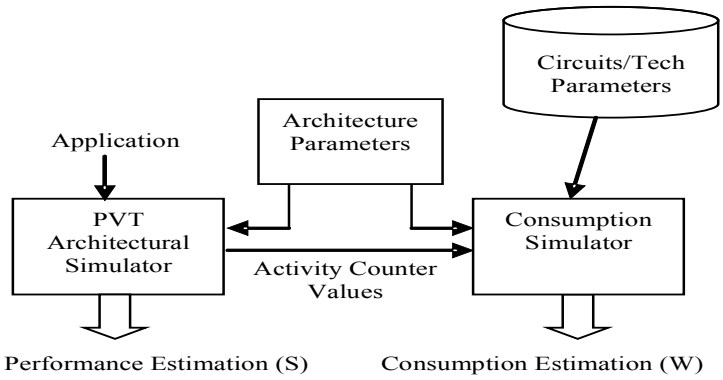


Figure 1.3: *Estimation de la consommation au niveau PVT*

au niveau CABA. Les deux points de différence concernent la granularité des activités pertinentes et l'intervalle de temps entre deux évaluations possibles de la consommation. En effet, au niveau CABA la consommation peut être estimée à chaque cycle, au niveau PVT ceci n'est pas possible. Les activités s'exécutent de façon atomique et nécessitent donc plusieurs cycles avant d'être terminées. Ce qui ne permet de connaître la consommation qu'à la fin des actions de gros-grain.

1.4 Méthodologie pour le développement des modèles de consommation

Dans cette section nous décrivons la méthodologie de développement des modèles de consommation pour les différents composants permettant de construire un système multiprocesseur aux niveaux CABA et PVT. Comme ceci a été précisé dans l'introduction de ce chapitre, notre méthodologie pour l'évaluation de la consommation est réalisée en deux étapes:

1. Localiser les activités pertinentes en terme de consommation d'énergie et associer à chaque activité un compteur afin de connaître le nombre d'occurrences correspondant lors de l'exécution de l'application.

2. Evaluer le coût énergétique élémentaire de chaque activité pertinente.

L'évaluation des coûts énergétiques des activités pertinentes dépend de l'architecture du composant et des paramètres technologiques. La consommation dans chaque composant est liée à 2 sources: la consommation dynamique, qui correspond aux changements de l'état interne du circuit (activités) et la consommation statique, qui est due principalement aux courants de fuite dans les transistors. Dans le passé, la dissipation dynamique était considérée plus importante que la dissipation statique. Avec les nouvelles technologies submicronique, la situation a changé et les deux sources de dissipation ont le même ordre de grandeur. La précision des modèles de consommation analytiques existants dans la littérature et qui ne prenant pas en compte les sources de dissipations statiques, est de ce fait remise en cause.

Pour cela, la solution qui offre le maximum de précision pour estimer la consommation est d'avoir physiquement le composant, de faire des tests et de réaliser des mesures mesures directes sur le circuit. Cette approche nécessite néanmoins du temps et des outils de conception. En effet avec cette approche, il faut manipuler des outils de modélisation de bas niveau (RTL ou transistor).

Dans notre travail pour satisfaire au critère de précision, nous avons utilisé des simulations de bas niveau lorsque les outils pour ces simulations sont disponibles et lorsque le composant s'apprête facilement à ces mesures. L'évaluation des coûts énergétiques dans les niveaux les plus bas (RTL pour nous) et leur utilisation pour réaliser des estimations dans les niveaux supérieures (CABA et PVT pour nous) a comme résultat une estimation hybride qui peut offrir un rapport précision/délai intéressant.

Néanmoins, cette évaluation avec des outils de CAO de bas niveau n'est pas réalisable pour tous les composants. En effet, cette approche n'est par exemple pas réalisable pour le réseau d'interconnexion crossbar. Ce dernier n'est pas un composant préexistant comme la mémoire ou le processeur. Rappelons, qu'un crossbar sert à connecter plusieurs initiateurs à différentes cibles, de ce fait la longueur de connexion dépend de l'emplacement de l'initiateur et de la cible sur la puce. Dans ce cas, il est nécessaire d'utiliser une approche analytique pour estimer le coût énergétique de l'activité.

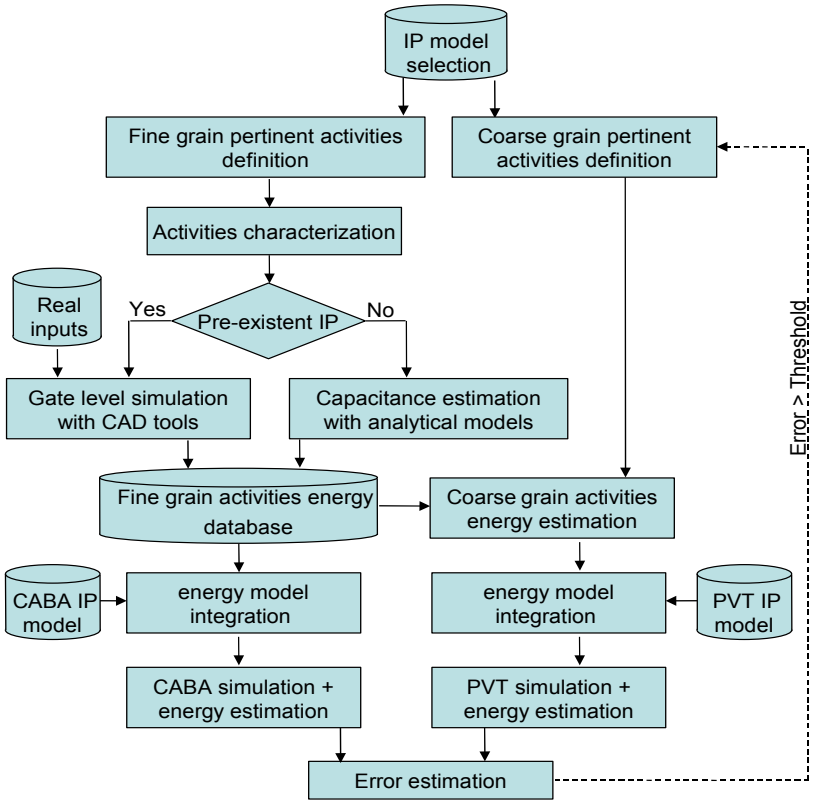


Figure 1.4: *Méthodologie de modélisation de l'énergie*

Notre méthodologie de développement des modèles de consommation aux niveaux CABA et PVT est présentée comme suit (voir la figure 1.4):

1. Identifier les activités pertinentes consommant une part significative d'énergie sur les deux niveaux de granularité qui nous intéressent CABA (grain-fin) et PVT (gros-grain).
2. Caractériser la consommation d'énergie des activités de grain-fin:
   - Avec des simulations de bas niveau en utilisant des outils CAO pour les composants préexistants. Les simulations sont réalisées avec des données réelles.
   - Avec des modèles analytiques pour les composants non préexistants.
3. Déduire la consommation des activités de gros-grain à partir de leurs définitions et de la consommation des activités de grain-fin de l'étape 2.
4. Intégrer les modèles de consommation développés dans le simulateur d'architecture et réaliser des simulations.
5. Evaluer l'erreur d'estimation en comparant les résultats obtenus au niveau PVT à ceux du niveau CABA. Ces derniers sont considérés comme étant plus précis. Si l'erreur est supérieure à un seuil fixé, l'étape de définition des activités de gros-grain est refaite. Dans ce cas, la définition des activités de gros-grain est corrigée en intégrant de nouvelles activités de grain-fin afin d'obtenir une meilleure évaluation de la consommation dans le composant.

Cette méthodologie a été appliquée à l'ensemble de composants matériels aux niveaux CABA et PVT à savoir le processeur MIPS R3000, la mémoire cache, le réseau d'interconnexion, la mémoire de données et d'instructions, le contrôleur DMA et l'accélérateur matériel TCD-2D. Dans nos expériences, la technologie de fabrication à 90nm a été adoptée. Pour évaluer les paramètres physiques, nous avons utilisé le modèle BPTM (*Berkeley Predictive Technology Model*) [6] qui se base sur l'outil BSIM4 [13]. Le texte ci-après donne une idée sur le nombre important de paramètres à configurer (de l'ordre de 200) pour pouvoir réaliser les simulations de bas niveau.

```
* PTM 90nm NMOS

.model  nmos  nmos  level = 54

+version = 4.0          binunit = 1           paramchk= 1
+capmod  = 2            igcmod  = 1           igbmod  = 1
+diomod  = 1            rdsmod  = 0           rbodymod= 1
+permod  = 1            acnqsmod= 0           trnqsmod= 0

+tnom    = 27           toxe    = 2.05e-9     toxp    = 1.4e-9
+dtox    = 0.65e-9      epsrox  = 3.9         wint    = 5e-009
+ll      = 0            wl      = 0           lln     = 1
+lw      = 0            ww      = 0           lwn     = 1
+lwl     = 0            wwl     = 0           xpart   = 0
+xl      = -40e-9
+vth0    = 0.397        k1      = 0.4         k2      = 0.01

...
```

```
* PTM 90nm PMOS

.model  pmos  pmos  level = 54

+version = 4.0          binunit = 1            paramchk= 1
+capmod  = 2            igcmod  = 1            igbmod  = 1
+diomod  = 1            rdsmod  = 0            rbodymod= 1
+permod  = 1            acnqsmod= 0            trnqsmod= 0

+tnom    = 27           toxe    = 2.15e-009    toxp    = 1.4e-009
+dtox    = 0.75e-9      epsrox  = 3.9          wint    = 5e-009
+ll      = 0            wl      = 0            lln     = 1
+lw      = 0            ww      = 0            lwn     = 1
+lwl     = 0            wwl     = 0            xpart   = 0
+xl      = -40e-9
+vth0    = -0.339       k1      = 0.4          k2      = -0.01
....
```

La manipulation de ces paramètres nécessite une expérience dans le domaine de la micro-électronique. En effet, pour la même technologie de fabrication, nous pouvons avoir plusieurs versions favorisant soit le critère de rapidité ou bien le critère de faible consommation. Ces difficultés nous ont poussé à développer des modèles de consommation qui dérivent des simulations de bas niveau, mais qui dépendent d'un nombre réduit de paramètres. Suivant le type de composant, un ensemble de paramètres spécifiques sera choisi. A titre d'exemple, pour une mémoire cache, le concepteur n'a pas à spécifier l'ensemble des paramètres en relation avec la structure des transistors nmos ou pmos utilisés, comme ceux donnés dans le tableau précédent. Il n'aura simplement, qu'à déterminer la taille des blocs, la taille totale du cache, l'associativité et la technologie utilisée. Ceci facilite la tâche des concepteurs de haut niveau pour aborder l'estimation de la consommation comme critère de développement.

1.5 Modèles de consommation d'énergie

1.5.1 Modèle de consommation du processeur MIPS R3000

Depuis quelques années, les concepteurs de processeurs accordent une importance de plus en plus grande à la consommation d'énergie. L'objectif des travaux réalisés ces dernières années est, là aussi, de pouvoir explorer les différentes alternatives matérielles et logicielles le plus tôt possible dans la chaîne de conception du système. Disposer d'un outil d'évaluation de la consommation d'énergie dans la phase où la sélection et le paramétrage du processeur est faite, permet sans aucun doute de réaliser un gain en temps et en consommation dans le produit final. En effet, jusqu'à une certaine date récente, la plupart des outils d'évaluation de la consommation d'énergie dans les processeurs existants travaillent au niveau de la fabrication physique du circuit. En plus du fait, que cette évaluation arrive tard et donc limite le nombre d'alternatives architecturales pouvant être testées, elle nécessite un temps de développement et de simulation très important et des outils très coûteux. Si ceci est vrai pour

tous les composants du MPSoC, pour le composant processeur l'avantage de réaliser rapidement des évaluations de la consommation a une importance de premier ordre. En effet, le processeur est responsable d'une part considérable de la consommation totale du système. En plus, plusieurs familles de processeur peuvent être explorées telles que les GPP (General Purpose Processor), les DSP (Digital Signal Processor), les VLIW (Very Large Instruction Word), etc. Pour chaque famille de processeur, plusieurs fréquences de fonctionnement peuvent être choisies pour obtenir différent compromis entre les critères de performance et de consommation d'énergie.

Dans le cadre de notre étude, nous utilisons un processeur MIPS R3000 puisque nous possédons sa description SystemC ce qui facilite son intégration dans notre plateforme. L'architecture détaillée de ce processeur a été décrite dans la section **??** du chapitre **??**. Notons que le modèle d'énergie proposé peut être adapté à d'autres types de processeurs tels que la famille ARM [3] ou les processeurs spécialisés DSP.

Estimation de la consommation au niveau CABA

Au niveau CABA, l'implémentation d'un processeur comporte la description de chaque étage de la micro-architecture au cycle précis. Au cours de l'exécution des instructions, les différents étages (chargement, décodage, exécution, etc.) participent à la consommation totale du processeur. Ce fonctionnement aux valeurs nominales de tension et de fréquence correspond au mode actif nominal. Cependant, le processeur peut être mis en attente dans le cas de défaut de cache ce qui correspond au mode inactif. Un processeur peut avoir d'autres modes de fonctionnement tels qu'actif à tension et fréquences différentes ou repos en basse consommation (*standby*), etc. Ces derniers modes sont utilisés pour optimiser la consommation d'énergie. La transition entre les différents modes de fonctionnement est assurée par la FSM qui contrôle le processeur. Chaque mode lui correspond une consommation d'énergie due aux activités des différents étages de la micro-architecture comme le montre la figure 1.5. Au niveau CABA, ces activités sont considérées de type grain-fin. La contribution de chaque activité dépend du mode de fonctionnement dans lequel se trouve le processeur. Dans cette thèse nous allons nous limitons aux deux principaux modes; le mode actif nominal et le mode inactif. Comme dans ce dernier mode, toutes les unités du processeur sont connectées à l'alimentation, le passage entre les 2 modes ne nécessite pas de temps.

Au niveau CABA, la consommation d'énergie dans chaque mode de fonctionnement est calculée cycle par cycle. Pour évaluer les coûts énergétiques correspondants, trois approches existent. La première approche est basée sur la documentation technique fournie par le constructeur du processeur. Le tableau 1.1 présente les caractéristiques techniques fournies par Intel pour le processeur StrongARM SA-1100 [7]. Dans ce tableau, nous retrouvons la puissance dissipée respectivement dans les modes actif (*normal mode*), inactif (*idle mode*) et repos (*sleep mode*) pour deux valeurs différentes de la fréquence d'horloge (133MHz et 190MHz). L'énergie par cycle est déduite par l'équation suivante:

$$E = P \cdot T_f = \frac{P}{F} \tag{1.2}$$

Cette approche permet d'obtenir des estimations précises de la consommation puisque les valeurs d'énergie sont fournies par le concepteur du circuit. Néanmoins, il n'est pas possible d'obtenir des statistiques sur la contribution de chaque étage de la micro-architecture. De plus, cette approche n'est pas flexible si nous nous intéressons à explorer des paramètres du

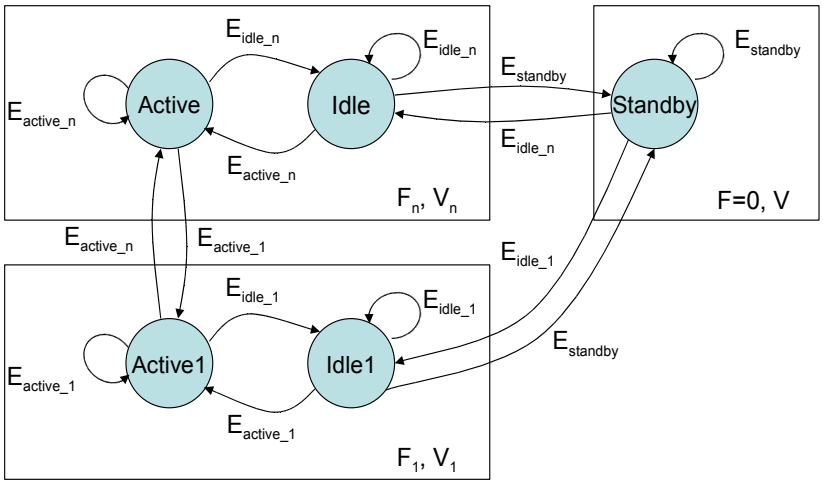


Figure 1.5: *La FSM qui contrôle un processeur avec 3 états de fonctionnement*

processeur comme: le nombre d'UAL entière ou la technique de prédiction de branchement utilisée.

Une deuxième approche pour estimer les coûts énergétiques de chaque mode de fonctionnement est d'utiliser la description du processeur au niveau RTL ou physique. Cette approche fait appel à des outils de CAO pour modéliser le circuit, faire des tests avec les données réelles et mesurer la consommation de l'énergie dans chaque mode de fonctionnement. Ainsi, cette approche permet des estimations précises. L'obstacle ici est le temps de simulation considérable pour obtenir les résultats. Ce temps dépend entre autres de la taille du circuit. Malheureusement, cette approche n'est pas applicable dans notre travail puisque nous ne possédons pas une description de bas niveau du processeur MIPS R3000.

Une troisième approche qui consiste à utiliser des simulateurs au niveau de la micro-architecture pour estimer la consommation des processeurs a été adoptée dans notre thèse. L'outil PowerAnalyzer [17] est un exemple de ce type de simulateur. Cet outil calcule la capacitance physique de chaque étage du pipeline du processeur et prend en compte la consommation statique. Avant l'exécution de l'application, le simulateur PowerAnalyzer nécessite deux fichiers de configuration. Le premier pour spécifier les paramètres de la technologie de fabrication adoptée, 90nm dans notre cas. Le deuxième fichier spécifie les paramètres de la micro-architecture du processeur. Le nombre de paramètres est de l'ordre de 40 et la liste ci-après présente quelques'un.

```
-fetch:ifqsize      \# Taille de la file de chargement des instructions
-bpred:bimod        \# Type de prédiction {nottaken|taken|bimod}
-decode:width       \# Bande passante du décodeur  (insts/cycle)
-issue:width        \# Bande passante d'exécution (insts/cycle)
-issue:inorder      \# Exécution dans l'ordre
-commit:width       \# Bande passante pour la terminaison des instructions (insts/cycle)
-ruu:size           \# Taille du registre RUU
-lsq:size           \# Taille de la file load/store
-res:ialu           \# Nombre d'UAL entières
-res:fpalu          \# Nombre d'UAL flottantes
```

Clock	133 MHz	190 MHz
Performance	150 Dhrystone 2.1 MIPS	220 Dhrystone 2.1 MIPS
Core power supply	**Vss** = 0.0 V dc **Vdd** = 1.5 V dc ± 5%	**Vss** = 0.0 V dc **Vdd** = 1.5 V dc ± 5%
I/O power supply	**Vssx** = 0.0 V dc **Vddx** = 3.3 V dc ± 10%	**Vssx** = 0.0 V dc **Vddx** = 3.3 V dc ± 10%
Typical power dissipation	Normal mode = <230 mW Idle mode = <50 mW Sleep mode = <50 µA	Normal mode = <330 mW Idle mode = <65 mW Sleep mode = <50 µA
Ambient operating temperature	0°C (32°F) min. 70°C (158°F) max.	0°C (32°F) min. 70°C (158°F) max.
Storage temperature	–20°C to +125°C (–4°F to +257°F)	–20°C to +125°C (–4°F to +257°F)
Packaging	208-pin LQFP†† 256 mBGA	208-pin LQFP†† 256 mBGA
Process technology	.35 µm, 3-layer metal	.35 µm, 3-layer metal
Transistor count	2.5 million	2.5 million

Table 1.1: *Caractéristiques techniques du processeur StrongARM SA-1100 [7]*

....

Après l'exécution de l'application, l'outil PowerAnalyser fournit différentes statistiques concernant la consommation de puissance statique et dynamique dans chaque étage comme le montre le texte ci-après.

```
alu access                 11012152 # number of times alu is accessed
alu max power                0.0001 # Maximum power for alu
alu total power            792.8749 # Total power for alu
alu avg power                0.0000 # Avg power for alu
mult access                       0 # number of times mult is accessed
mult max power               0.0000 # Maximum power for mult
mult total power             0.0000 # Total power for mult
mult avg power               0.0000 # Avg power for mult
fpu access                        0 # number of times fpu is accessed
fpu max power                0.0000 # Maximum power for fpu
fpu total power              0.0000 # Total power for fpu
fpu avg power                0.0000 # Avg power for fpu
...
uarch.switching         234658.9612 # uarch total in switching power dissipation
uarch.avgswitching           0.0098 # uarch avg in switching power dissipation
uarch.internal          824905.6538 # uarch total internal power dissipation
uarch.avginternal            0.0343 # uarch avg internal power dissipation
uarch.leakage            23044.0609 # uarch total leakage power dissipation
uarch.avgleakage             0.0010 # uarch avg leakage power dissipation
uarch.pdissipation     1082608.6757 # uarch total power dissipation
```

Notons ici, que l'outil PowerAnalyzer exprime la consommation en terme de puissance et non pas en terme d'énergie. Cependant, cette valeur peut être déduite facilement à partir

de la puissance et le temps d'exécution de l'application. Différents modèles de consommation de puissance sont utilisés dans chaque étage de la micro-architecture pour estimer la consommation dans les différents modes de fonctionnement du MIPS R3000. Par la suite, les énergies consommées par les étages sont additionnées pour obtenir le coût énergétique total. Le tableau 1.2 résume les coûts en puissance et énergie obtenus pour les modes actif et inactif à une fréquence de fonctionnement de 100MHz. L'avantage d'une telle approche est de fournir des mesures relativement précises. De plus, elle permet d'explorer les paramètres du processeur comme le nombre d'UAL utilisée et leur type, la prédiction de branchement, etc.

Processeur	Vdd (V)	f(MHz)	P_{active}(mW)	E_{active}(nJ)	$P_{inactive}$(mW)	$E_{inactive}$(nJ)
MIPS R3000	1.1	100	45	0.45	30	0.3

Table 1.2: *La consommation du processeur MIPS R3000*

Au niveau CABA, nous avons donc simplifié le fonctionnement du processeur et nous avons considéré que son automate d'état peut effectuer deux types d'opérations: l'exécution d'une instruction ou l'attente de données ou d'instructions. Ce qui correspond respectivement aux modes actif et inactif. Deux compteurs correspondant aux deux modes précédents ont été ajoutés à la description du composant. A la fin de la simulation, les valeurs de ces compteurs seront multipliées par les coûts d'énergie de chacun des deux modes pour calculer la consommation totale du processeur. L'équation suivante résume la façon avec laquelle la consommation est calculée au niveau CABA:

$$E_{processor} = n_{running} \cdot E_{running} + n_{idle} \cdot E_{idle} \tag{1.3}$$

$n_{running}$ représente le nombre de cycles pendant lequel le processeur exécute des instructions (actif) et n_{idle} le nombre de cycles où le processeur est en attente (inactif).

Estimation de la consommation au niveau PVT

L'approche d'estimation de la consommation au niveau de la micro-architecture précédemment décrite ne peut pas être appliquée au niveau PVT. En effet dans ce deuxième niveau, nous avons fait le choix d'utiliser un simulateur au niveau des instructions (ISS) pour modéliser le fonctionnement du processeur. Ce type de simulateur exécute l'application instruction par instruction sans se référer à la micro-architecture du composant. Par conséquent, les activités de chaque étage ne peuvent pas être récupérées durant la simulation. Pour cela, il nous a fallu concevoir une autre méthode d'estimation de la consommation mieux adaptée à la description au niveau PVT. Au cours de la simulation, l'information pertinente qui peut être dégagée au niveau PVT est le type de l'instruction en cours d'exécution. D'où l'idée de développer un modèle de consommation qui considère une énergie dissipée pour chaque type d'instruction. L'ensemble des valeurs de la consommation des différentes instructions constituera le modèle d'énergie du processeur. Pour caractériser le coût de chaque instruction, nous avons effectué des simulations à l'aide de l'outil PowerAnalyzer. Le texte ci-après montre un exemple de code pour évaluer le coût de l'instruction *add*.

```
main()
{
```

```
int i=0;
for(i=0;i<nbr_iter;i++)
__asm(
         "add    r4,r3,r4\n"
     );
return 0;
}
```

Nous avons exécuté cette boucle deux fois, une fois avec l'instruction à tester et une deuxième fois sans cette instruction. Pour évaluer la consommation de l'instruction, la différence entre les deux simulations sera divisée par le nombre d'itérations de la boucle (nbr_iter fixé à 1 million dans les expériences) .

Les résultats de simulation montrent une faible variation de l'énergie dissipée par cycle entre les différentes instructions. Cette valeur est de 8.3% ($E_{min} = 0.436nJ$, $E_{max} = 0.472nJ$). En effet, le processeur MIPS R3000 possède une architecture scalaire simple exécutant une instruction par cycle. Nous considérons ici le cas du mode actif. Les expérimentations réalisées sur les processeurs Hitachi HS-4 et StrongARM SA-1100 qui sont des architectures similaires au MISP R3000 ont donné des résultats proches [18] avec une variation maximum de consommation de 8%. Pour minimiser l'erreur, dans notre modèle de processeur nous avons utilisé un coût énergétique différent pour chaque instruction. Dans d'autres processeurs plus complexes comme ceux qui possèdent une architecture superscalaire, il est difficile d'appliquer cette méthode puisqu'il est assez difficile d'évaluer la part de chaque instruction dans un groupe de plusieurs instructions s'exécutant de façon concurrente. En effet, le coût d'une instruction peut dépendre des interactions avec les autres instructions concurrentes.

Pour notre architecture de processeur MISP R3000, le processeur est mis à l'état "inactif" dans le cas de défaut de cache. Le nombre de cycles d'attente est obtenu après chaque transaction *read (adr, data, time)* ou *write (adr, data, time)*. Le paramètre *time* est ici récupéré par le processeur pour déterminer le délai de la transaction en ns ou en cycles horloge. Dans le cas de défaut de lecture en cache, le temps d'attente du processeur est égal à la somme des valeurs suivantes: temps d'accès au cache, temps de transmission de la requête de lecture via le réseau d'interconnexion, temps d'accès de la mémoire partagée et enfin temps de transmission du bloc de cache vers le processeur via le réseau d'interconnexion. Comme il a été expliqué dans les chapitres précédents, ce temps d'attente n'est pas constant car il dépend des contentions sur le réseau d'interconnexion. L'estimation de ce paramètre influe directement sur la précision de l'estimation de la consommation du mode inactif. Pour évaluer le coût énergétique de ce mode nous avons utilisé l'approche décrite au niveau CABA.

De l'autre côté, pour calculer l'occurrence de chaque instruction et le nombre de fois où le processeur passe dans le mode inactif, des compteurs ont été ajoutés à la description du composant. A la fin de la simulation, les valeurs de ces compteurs seront multipliées par les coûts d'énergétiques pour calculer la consommation totale du processeur. En conclusion, au niveau PVT, nous avons obtenons pour le processeur MIPS R3000 le modèle de consommation suivant:

$$E_{processor} = n_{add} \cdot E_{add} + n_{mul} \cdot E_{mul} + ... + n_{idle} \cdot E_{idle} \tag{1.4}$$

1.5.2 Modèle de consommation pour la mémoire SRAM

La mémoire dans les systèmes embarqués est responsable d'une grande partie de la consommation totale. Cette source de consommation s'amplifie avec les applications de traitement de données intensif due à la grande quantité de données transitant entre les mémoires, les processeurs et les accélérateurs matériels. Dans la littérature deux types de circuits mémoires existent: statique (SRAM) et dynamique (DRAM). Jusqu'à une date récente, seules les mémoires de type SRAM pouvaient être intégrés avec le reste du système sur la même puce. Néanmoins, aujourd'hui avec le progrès technologique, il est possible de combiner des circuits mémoires de DRAM dans un SoC [16]. Dans ce travail nous sommes intéressés à étudier le comportement des circuits mémoires de type SRAM d'un point de vue consommation d'énergie. En général, les mémoires du type SRAM peuvent avoir plusieurs modes de fonctionnement au cours de l'exécution d'une application à savoir: lecture, écriture, inactif et repos ((*standby*). Le passage à ce dernier état est contrôlé extérieurement à partir de l'entrée *clock* ou bien *chip select*. Dans le module SRAM que nous avons développé, ce dernier mode n'a été utilisé.

Estimation de la consommation au niveau CABA

Au niveau CABA, les mémoires consomment une énergie moyenne par cycle pour chaque état de fonctionnement. Pour ce composant, nous avons identifié les trois activités pertinentes qui consomment de l'énergie: READ, WRITE et IDLE qui correspondent à des opérations de lecture, écriture et inactif. Pour estimer le coût énergétique des activités, trois approches existent. La première approche consiste à utiliser la documentation technique fournie par les concepteurs des circuits mémoires. Cette documentation spécifie les valeurs de l'intensité du courant pour chaque mode de fonctionnement. En général, les valeurs $I\_nominal$ et $I\_inactif$ qui correspondent respectivement aux modes nominal et inactif sont fournis. L'énergie pourra être calculée à partir de ces valeurs et des valeurs nominales de fréquence et de tension fournies elles aussi dans la documentation technique en utilisant l'équation suivante:

$$E = \frac{I \times V}{f} \tag{1.5}$$

Dans ce cas, l'énergie nominale correspond à l'énergie moyenne dissipée dans les modes de lecture et écriture. Le tableau 1.3 donne les valeurs d'énergie dans les modes nominal et inactif extraits de la documentation technique du circuit SRAM μPD431000A de NEC [15].

Mémoire	Taille	Tension	Fréquence	I_{nom}	I_{inac}	E_{nom}	E_{inact}
SRAM NEC μPD431000A	128Kox8bit	2.7 V	14 MHz	70 mA	100 μA	13.5 nJ	19.28 pJ

Table 1.3: *Consommation d'énergie de la mémoire SRAM µPD431000A de NEC*

Cette approche présente l'avantage de la précision puisque les valeurs d'énergie dérivent de la documentation technique fournie par le concepteur du circuit. Malheureusement, cette approche n'est pas applicable si l'objectif est de réaliser une exploration de l'architecture de la mémoire.

Une deuxième approche qui consiste à utiliser des équations analytiques pour évaluer le coût énergétique de chaque mode de fonctionnement de la SRAM existe [10, 11]. Cette approche est basée sur l'utilisation de l'équation suivante:

$$E = 0.5 \times C \times V^2 \tag{1.6}$$

L'énergie dissipée dépend par conséquent de la tension d'alimentation (fixe) et de la capacitance totale. La difficulté est ici de calculer la capacitance totale pour chaque mode de fonctionnement. Cette approche peut être appliquée lorsque le circuit est décrit au bas niveau (schéma en transistors) et nécessite la connaissance des détails du fonctionnement du circuit. Nous pensons que cette approche n'est pas facilement abordable par les concepteurs de systèmes de haut niveau. Notons néanmoins que cette approche permet de fournir un modèle paramétrable.

Pour remédier aux inconvénients des deux approches précédentes, nous proposons une approche qui se base sur des simulations de bas niveau afin d'obtenir un modèle paramétrable. Pour estimer le coût énergétique des activités, nous avons simulé des circuits SRAM de différentes tailles au niveau physique avec l'outil ELDO de Mentor Graphics [2]. Ces mémoires ont été conçues au Laboratoire d'Informatique de Paris 6 (LIP6)[1], la figure 1.6 présente la structure générale des circuits SRAM simulés. Notre mémoire possède un seul port pour la lecture et l'écriture, deux lignes de précharge pour chaque colonne et une ligne de mot pour chaque ligne mémoire. Les composants de base qui constituent cette mémoire sont la cellule mémoire de base qui sert à stocker un bit, le décodeur d'adresses pour sélectionner une ligne mémoire et le *sense amplifier* pour accélérer la phase de décharge au cours de l'opération de lecture.

La synthèse logique, le placement, le routage et l'extraction du layout sont réalisés par des outils de CAO de l'environnement de conception VLSI ALIANCE [1]. Notre objectif est d'avoir un modèle paramétré des coûts énergétiques pour les différentes activités d'une mémoire SRAM en fonction du nombre de mots (M) et du nombre de bits par mot (N). Ceci permet ultérieurement de faire de l'exploration architecturale. A partir des coûts énergétiques mesurés ainsi que les temps d'accès, nous avons déduit les coûts d'énergie des activités. Un tel modèle offre un niveau de précision relativement élevé du fait qu'il prend en considération toutes les sources de consommation possibles. Les résultats de simulation montrent que les coûts énergétiques varient linéairement avec le nombre de mots et le nombre de bits par mot. La figure 1.7.a, respectivement 1.7.b, donne le coût en énergie d'une opération de lecture en faisant varier le nombre de mots (M) avec une taille fixe de (8, 16 ou 32) bits par mot, respectivement le coût en faisant varier le nombre de bits (N) avec un nombre fixe de (32, 64 ou 128) mots.

Les résultats expérimentaux nous ont permis d'écrire les équations suivantes pour chaque type d'activité:

$$\begin{aligned} E_{read} &= (R_0 + R_1 \cdot N) \cdot (R_2 + R_3 \cdot M) \\ E_{write} &= (W_0 + W_1 \cdot N) \cdot (W_2 + W_3 \cdot M) \\ E_{idle} &= (I_0 + I_1 \cdot N) \cdot (I_2 + I_3 \cdot M) \end{aligned} \tag{1.7}$$

Le coût énergétique de chaque activité est le produit de deux fonctions affines ayant N et M comme paramètres. Les constantes R_1, R_3, W_1, W_3, I_1 et I_3 représentent les coefficients

[1] http://www.lip6.fr

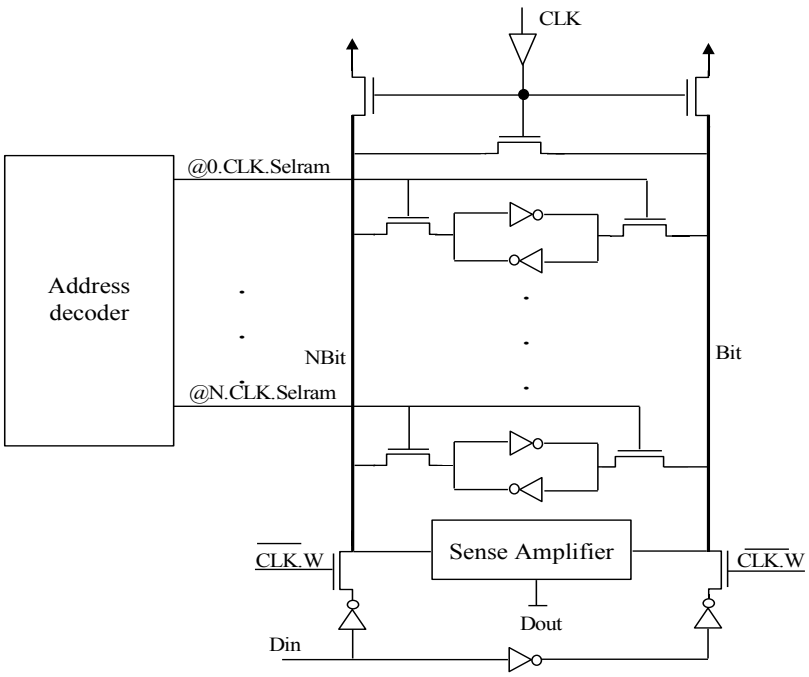


Figure 1.6: *Structure d'une mémoire SRAM*

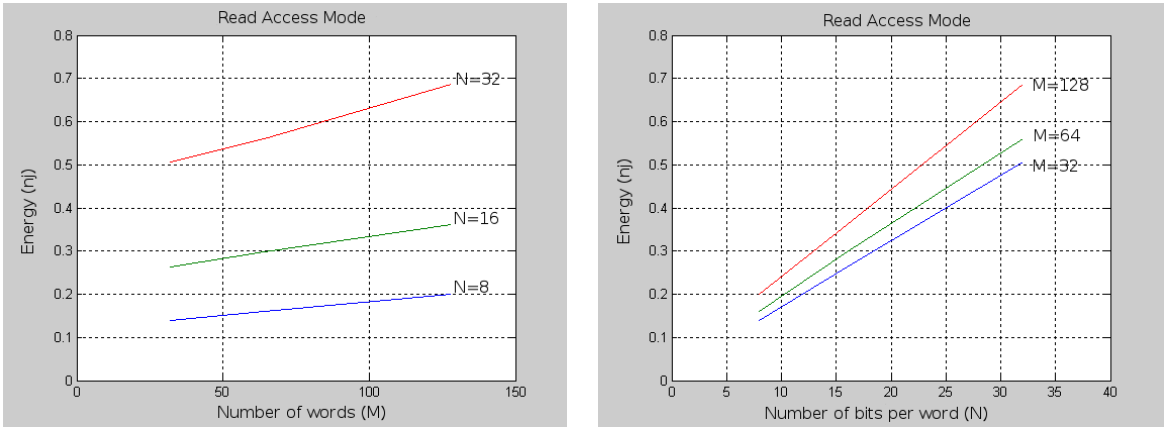
directeurs des fonctions affines alors que R_0, R_2, W_0, W_2, I_0 et I_2 représentent les ordonnées à l'origine. A partir des valeurs expérimentales obtenues par les mesures sur les circuits existants, nous avons résolu ce système d'équations afin de déterminer les valeurs des coefficients R_i, W_i et I_i. Nous avons obtenu ainsi un modèle simple pour estimer la consommation des activités de base dans le composant SRAM. Il faut mentionner que ces paramètres sont valables pour une technologie donnée et pour avoir un modèle générique il faut trouver le coefficient de pondération α entre les paramètres d'une technologie λ_0 et une autre λ, on écrit alors:

$$\begin{aligned} R_i &= R_{i0} \cdot (\lambda/\lambda_0)^{\alpha} \\ W_i &= W_{i0} \cdot (\lambda/\lambda_0)^{\alpha} \\ I_i &= I_{i0} \cdot (\lambda/\lambda_0)^{\alpha} \end{aligned} \tag{1.8}$$

[9] propose l'équation 1.9 pour de modéliser la consommation dans une technologie W à partir d'une technologie référence W'.

$$\begin{aligned} S &= W/W' \\ U &= V/V' \\ P' &= P \cdot (S/U^3) \\ f' &= f \cdot (S^2/U) \end{aligned} \tag{1.9}$$

Ici V' est la tension d'alimentation de référence, V est la tension d'alimentation ciblée. S et U représentent respectivement le facteur d'échelle de la technologie et de la tension. P' et f' représentent respectivement la puissance et la fréquence après la mise en échelle. L'énergie peut être déduite à partir de la puissance et du temps d'exécution.



(a) Energy cost variation in terms of M

(b) Energy cost variation in terms of N

Figure 1.7: Coûts énergétiques en fonction du nombre de mots (M) et du nombre de bits par mot (N)

Après avoir déterminé les coûts élémentaires des opérations, il est nécessaire de connaître le nombre de chaque type d'opération. Ainsi, des compteurs d'événements par opération ont été ajoutés dans la FSM qui contrôle le composant SystemC (figure 1.8). Nous avons ainsi ajouté des compteurs pour estimer le nombre de lectures, d'écritures et de cycles d'attente. Au niveau CABA, la consommation d'énergie de la SRAM pour une application

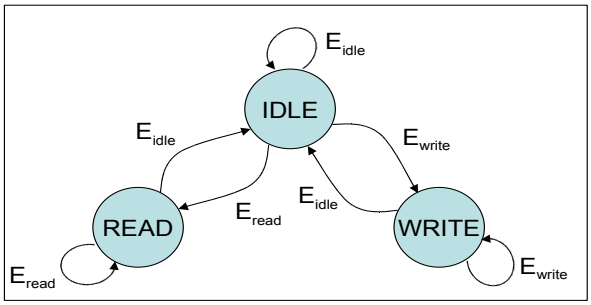


Figure 1.8: *FSM de la mémoire SRAM*

est déterminée par le modèle suivant:

$$E_{SRAM} = n_{read} \cdot E_{read} + n_{write} \cdot E_{write} + n_{idle} \cdot E_{idle} \tag{1.10}$$

n_read, *n_write* et *n_idle* sont les valeurs respectives des compteurs d'accès en lecture, en écriture et les cycles d'attente.

Estimation de la consommation au niveau PVT

Malgré que la FSM qui contrôle le composant SRAM à ce niveau n'est pas implémentée, nous définissons la même granularité pour les activités pertinentes que celle au niveau CABA. Ceci est vrai parce qu'il est possible de récupérer les activités définies (READ, WRITE et IDLE) au niveau transactionnel ce qui n'est pas toujours le cas avec les autres composants. Néanmoins, cette récupération n'est pas directement obtenue comme nous allons l'expliquer.

Au niveau PVT, la mémoire est décrite comme un composant passif comprenant deux méthodes: read() et write(). Dans la description du composant, deux compteurs sont ajoutés à chaque méthode pour déduire les occurrences correspondantes. L'obstacle ici est de déduire les occurrences qui correspondent aux cycles d'attente puisque la description au niveau PVT est dépourvue de la notion d'horloge. En plus, la fonction qui décrit la mémoire n'est exécutée que pendant les accès en lecture ou écriture. Pour résoudre cet obstacle nous allons utiliser le paramètre *time* des requêtes *read(address, data, time)* et *write(address, data, time)*. Ces requêtes sont transmises par les initiateurs. Le paramètre *time* contient la valeur du compteur de temps local de l'initiateur. Au niveau de la mémoire, ce paramètre est récupéré pour calculer les temps de repos (en nano seconde ou en cycles) de la mémoire. La mémoire déduit ainsi son horloge en utilisant les horloges locales des processeurs.

A chaque intervalle de temps, il est possible de calculer de la consommation d'énergie au cours de l'exécution de l'application. L'intervalle de temps est calculé entre deux transactions successives issues du même processeur. A titre d'exemple, si le processeur 0 transmet une première requête à la mémoire partagée à l'instant 1000ns ensuite une deuxième à l'instant 1300ns alors il est possible de calculer le temps de repos entre les instants 1000 et 1300ns et par la suite la consommation d'énergie correspondante. Le temps de repos de la mémoire est égal à la différence de l'intervalle concerné (dans notre exemple 300ns) et la somme des temps d'accès à la mémoire en lecture ou en écriture. Supposons qu'il y a eu quatre accès en lecture de 50ns chacun au cours de cet intervalle de temps. Le temps de repos est égal à 100ns (300 - 4x50).

Après le calcul des occurrences des différentes activités, l' équation décrite au niveau CABA (équation 1.10) est de nouveau utilisée pour déduire la consommation totale dans le composant SRAM. En conclusion, la méthodologie présentée pour développer un modèle de consommation précis pour les mémoires SRAM nécessite un effort considérable. Cet effort nous a permis aussi d'avoir un modèle simple ne contenant pas un nombre important à configurer. Dans notre plateforme, le modèle de SRAM développé est utilisé pour estimer la consommation de la mémoire partagée d'instructions, la mémoire de données et que des registres du contrôleur DMA.

1.5.3 Modèle de consommation de la mémoire cache

Estimation de la consommation au niveau CABA

Dans le chapitre précédent, nous avons détaillé la description du composant xcache au niveau CABA. Ce composant est formé principalement d'un cache de données, un cache d'instructions et une FIFO pour stocker les demandes de lecture ou d'écriture en mémoire partagée. Ces différents blocs participent à la consommation totale du composant. Au cours de l'exécution de l'application, les accès à ces blocs sont contrôlés par les deux FSM des caches de données et d'instructions. Les tableaux 1.4 et 1.5.3 représentent les différents états des FSM qui contrôlent le cache de données, le cache d'instructions et la FIFO de requêtes ainsi les activités qui leurs correspondent. A titre d'exemple, l'état INIT correspond à l'initialisation du cache de données ou d'instructions, l'état WRITE_UPDT correspond au chargement d'une donnée dans le cache, etc. Dans ces tableaux, *R* représente un accès en lecture, *W* un accès en écriture et "–" un état de repos. Chaque état correspond à un ensemble d'activité grain-fin de type écriture ou lecture dans le répertoire (Tag), écriture ou lecture dans la partie donnée ou écriture dans la FIFO. Exemple l'état INIT correspond à des

écritures successives dans le répertoire et de cette façon nous identifions les activités de type grain-fin qui correspondent à chaque état.

	Activity Type		
DATA cache FSM	TAG	DATA	FIFO
DCACHE_INIT	W	–	–
DCACHE_IDLE	R	R	–
DCACHE_WRITE_UPDT	–	W	–
DCACHE_WRITE_REQ	R	R	W
DCACHE_MISS_REQ	–	–	W
DCACHE_MISS_WAIT	–	–	–
DCACHE_MISS_UPDT	W	W	–
DCACHE_UNC_REQ	–	–	W
DCACHE_UNC_WAIT	–	–	–

Table 1.4: *La FSM du cache de données*

	Activity Type	
INST cache FSM	TAG	DATA
ICACHE_INIT	W	–
ICACHE_IDLE	R	R
ICACHE_WAIT	–	–
DCACHE_UPDT	W	W
DCACHE_UNC_WAIT	–	–

Table 1.5: *La FSM du cache d'instructions*

En conclusion, l'estimation de la consommation d'un état revient à évaluer le coût d'accès en lecture ou écriture à une mémoire SRAM (le tableau des étiquettes ou le tableau de données) et le coût d'écriture dans la FIFO. Prenons comme exemple l'état INIT, nous avons $E_{INIT} = E_{write\_tag}$ et pour l'état IDLE, correspondant à un succès de cache, nous avons $E_{idle} = E_{read\_tag} + E_{read\_data}$. Ainsi, nous définissons la consommation d'énergie qui correspond à chaque état. La figure 1.9 montre les différentes énergies qui correspondent à chaque transition d'état de la FSM du cache de données.

Pour évaluer le coût d'accès en lecture ou écriture à la SRAM (le tableau des étiquettes ou le tableau de données), le travail est déjà fait dans la section précédente et pour la FIFO nous avons procédé de la même façon que la SRAM. Nous avons simulé avec l'outil ELDO des FIFO de différentes tailles pour trouver un modèle paramétrable. Dans chacun des automates des caches, nous avons inséré des compteurs: WRITE_TAG, READ_TAG, WRITE_DATA, READ_DATA, WRITE_FIFO et IDLE. A la fin de la simulation, nous obtenons la valeur de chaque compteur qui sera multipliée avec le coût énergétique de l'activité pour trouver la consommation totale dans le composant xcache.

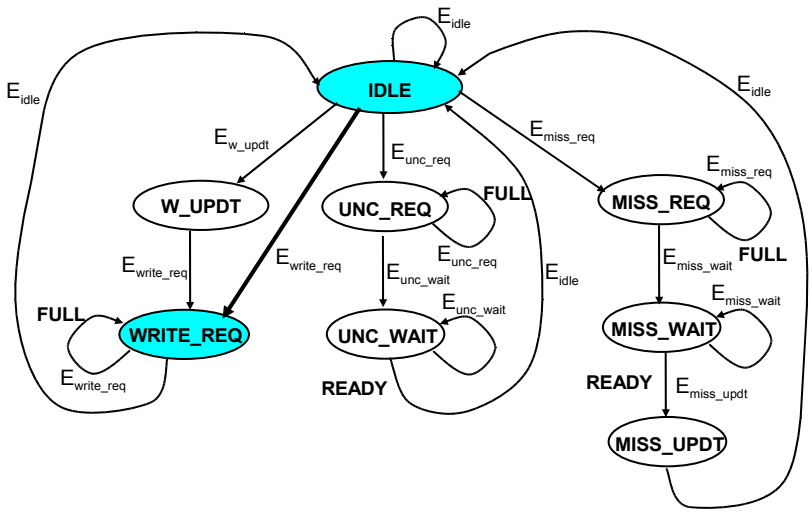


Figure 1.9: *FSM du cache de données*

Estimation de la consommation au niveau PVT

Au niveau PVT, cette méthodologie pour estimer la consommation du cache d'instructions ou de données ne peut pas être appliquée du fait que la FSM qui contrôle le composant n'est pas implémentée. Les activités de grain-fin ne peuvent pas être récupérées. Pour cela nous avons défini un ensemble d'activités de gros-grain qui sont: *Read_hit et Read_miss* qui correspondent à un succès ou un défaut en lecture, *Write_hit et Write_miss* qui correspondent à un succès ou un défaut en écriture, et *Idle* qui représente l'état inactif du composant. Chaque activité de gros-grain synthétise plusieurs transitions au niveau des états de la FSM du cache. En d'autres termes, une activité de gros grain correspond à un ensemble d'activités de grain-fin de type écriture ou lecture dans le répertoire (Tag), écriture ou lecture dans la partie donnée et écriture dans la FIFO.

La figure 1.10 montre la définition de chaque activité de gros-grain en fonction des états de la FSM du cache. A titre d'exemple, l'activité Read_miss correspond à l'exécution de l'état MISS_REQ (demande de lecture d'un bloc de cache) ensuite l'exécution de l'état MISS_UPDT M fois pour la mise à jour de la ligne dans le cache. Le paramètre M présente la taille du bloc de cache. L'évaluation du coût énergétique de chaque activité de gros-grain est déduite à partir de sa définition et des coûts des activités de grain-fin qui découlent des mesures expérimentales. Ce qui nous permet d'achever le maximum de précision. Toujours dans le même exemple, la consommation d'énergie de l'activité gros-grain Read_miss est la somme des consommations de l'état MISS_REQ et M fois l'état MISS_UPDT.

Au niveau PVT, un compteur est alloué à chaque activité de gros-grain dans la description du composant pour calculer l'occurrence correspondante au cours de la simulation. Comme nous l'avons déjà précisé pour la mémoire SRAM, un obstacle nous rencontre pour calculer les temps de repos du composant cache. Nous résolvons le problème de la même façon en utilisant le paramètre *time* des requêtes *read(address, data, time)* et *write(address, data, time)*. Néanmoins, le problème ici est plus simple car le composant cache n'est accédé que par un seul initiateur. Dans le cas de succès de cache, le temps de repos est calculé entre deux requêtes successives de l'initiateur. Dans le cas de défaut de cache, le composant instancie une

nouvelle requête (*read(address, data, time)*) qui sera transmise via le réseau d'interconnexion. Le paramètre *time* sera récupéré pour savoir le temps mis pour avoir la réponse ce qui correspond au temps de repos.

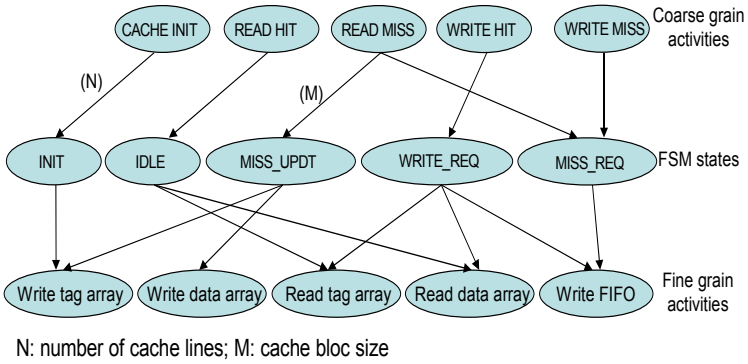


Figure 1.10: *Définition des activités du cache*

Une autre méthodologie pour évaluer les coûts des activités de fin-grain et gros-grain est l'utilisation de l'outil CACTI [5]. CACTI est un outil pour estimer le temps d'accès, le temps de cycle, la surface et la consommation statique et dynamique pour les mémoires caches. En intégrant tous ces paramètres ensemble, le concepteur peut explorer différentes organisations et tailles de cache à fin de retrouver la meilleure configuration pour son application. CACTI modélise analytiquement d'une façon fine les différents étages d'une mémoire cache et tient compte des paramètres de la technologie de fabrication, donc il permet une estimation de la consommation avec une précision acceptable. Cet outil modélise la consommation pour une architecture conventionnelle d'une mémoire cache. Pour les systèmes MPSoC, généralement nous utilisons des architectures plus complexes afin de gérer le problème de cohérence de cache. Dans ce cas, le concepteur doit prendre en considération les modifications nécessaires dans le modèle de consommation pour obtenir des estimations assez précises.

1.5.4 Modèle de consommation du crossbar

Le modèle de consommation d'un réseau d'interconnexion dépend principalement de sa topologie (bus, crossbar, etc.) et du mode de communications (synchrone ou asynchrone). Ces derniers temps les systèmes GALS (Globalement Asynchrone, Localement Synchrone) deviennent de plus en plus répondus. Ces systèmes sont composés de plusieurs sous-systèmes (monoprocesseur ou multiprocesseurs) chacun utilisant sa propre horloge et d'un réseau d'interconnexion entre les sous-systèmes. Les unités qui composent un sous-système utilisent la même horloge et sont donc synchrones alors que le réseau d'interconnexion est conçu de façon asynchrone. Cette approche offre plusieurs avantages: évite la diffusion de l'horloge à tout le système (distorsion de l'horloge ou problème du "clock skew"), simplifie la réutilisation des composants et enfin diminue la latence de communication et la consommation d'énergie. Dans notre MPSoC nous utilisons le composant crossbar de la bibliothèque SoCLib comme réseau d'interconnexion. Le fonctionnement de ce composant est synchrone avec le reste du système. Il permet à l'utilisateur de spécifier le nombre d'initiateurs et de

cibles ainsi que les latences de communication en nombre de cycles. Néanmoins, les valeurs exactes de latences dépendent principalement des longueurs de fils entre les différents ports.

Au niveau du crossbar, il y a trois principales sources de consommation d'énergie [19]:

- La consommation dissipée dans chaque fil de connexion lors d'une transition d'un bit de 0 à 1 ou de 1 à 0. Nous considérons que les énergies correspondant aux états statiques ($E_{bit0\rightarrow0}$) et ($E_{bit1\rightarrow1}$) sont nulles.
- La consommation des FIFO utilisées pour stocker les requêtes en attente à chaque entrée et sortie du réseau d'interconnexion. C'est le modèle de consommation de la SRAM qui a été développé dans la section 1.5.2 qui sera utilisé pour estimer le coût de ces FIFO. Le nombre d'accès en lecture et en écriture ainsi que les cycles d'attente sont donc calculés pour estimer la consommation totale des FIFOs.
- La consommation due à la logique nécessaire pour implémenter les fonctions de routage et d'arbitrage dans le crossbar. Des multiplexeurs ou démultiplexeurs sont généralement utilisés pour réaliser ces tâches. Dans notre travail nous considérons que cette source de consommation est négligeable.

Estimation de la consommation au niveau CABA

Parmi les trois sources de consommation celle des fils de connexions est considérée comme étant la plus importante. Au niveau CABA, le transfert de données entre deux ports de l'interface VCI présente l'activité pertinente de grain-fin d'un point de vue consommation d'énergie. La prise en compte de la consommation d'énergie en niveau du crossbar doit se faire en prenant en compte la taille en nombre de bits des différents ports. En effet cette valeur diffère d'un port à l'autre. A titre d'exemple, le port ADDRESS comprend 32 bits alors que le port CMD ne comprend que 2 bits. Rappelons ici que généralement, le transfert de données entre deux interfaces du crossbar nécessite plusieurs cycles et la consommation d'énergie dans un cycle donné dépend du nombre de ports utilisés. La figure 1.11 montre un exemple de communication entre un initiateur et une cible (2 transactions). La première transaction de lecture prend un seul cycle et lui correspond une énergie totale (E_{C0}) égale à la somme des énergies dissipées par les ports ADDRESS, RDATA, CMD, CMDVAL et EOP. L'énergie de chaque port peut être calculée par la formule suivante:

$$E_{i,j} = \alpha \cdot E_0(L_{i,j}) \tag{1.11}$$

i et j: *Numéro des ports: initiateur et cible*

Le coût de transfert d'une donnée $E_{i,j}$ dépend du nombre de bits qui ont commuté entre les états "0" et "1" et vice versa, notée α), et de E_0, qui représente le coût pour transférer un seul bit. α est compris entre 0 et n avec n le nombre de bits du port considéré. Le paramètre α peut être calculé durant la simulation par des compteurs. Cette approche est précise, mais risque de ralentir la simulation du fait qu'il faut calculer à chaque cycle la valeur α). Un certain nombre de simulateurs comme Wattch [4] estime ce nombre à la moitié du nombre de bits au niveau de l'interface (soit ici n/2). Dans nos travaux, nous avons utilisé aussi cette solution.

Le coût élémentaire pour transférer un seul bit E_0 dépend de la longueur du fil de connexion $L_{i,j}$ entre les deux ports i et j. Le problème que nous avons rencontré concerne

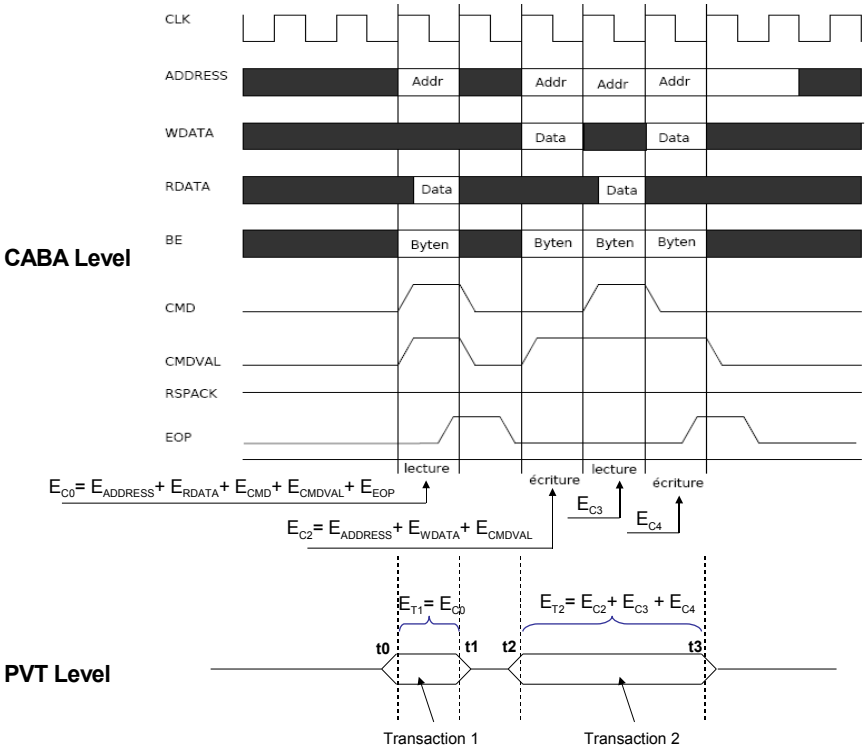


Figure 1.11: *Consommation de la partie communication aux niveaux CABA et PVT*

l'estimation de la longueur des connexions entre les 2 composants i et j. En effet cette longueur dépend de l'emplacement final des composants sur la puce. Nous avons utilisé l'outil GRAAL, éditeur de layout, de l'environnement ALLIANCE pour mesurer la taille des mémoires SRAM, du contrôleur DMA et des caches d'instructions et de données en fonction du paramètre technologique λ. Pour le processeur MIPS R3000, nous avons utilisé l'approche présentée dans [12]. L'estimation de la taille de notre accélérateur matériel TCD-2D est déduite à partir de l'implémentation au niveau RTL réalisée sur le composant FPGA STRATIX II EP2S60. A partir des tailles de nos composants, nous avons développé un modèle pour estimer les longueurs des fils de connexion tout en tenant compte du nombre de processeurs et des mémoires utilisés. Une fois la phase d'estimation des longueurs des fils de connexions est terminée, l'énergie E_0 est calculée comme suit:

$$E_0 = C(L_{i,j}) \cdot V_{dd}^2 \tag{1.12}$$

$C(L_{i,j})$ est la capacitance du fil de longueur $L_{i,j}$. Pour la technologie 90nm adoptée dans nos expérimentations, la capacitance du fil est estimée à $0.25 f\mathrm{F}/\mu\mathrm{m}$. V_{dd} est la tension des fils fixée dans nos travaux à 1.1v. Pour récupérer les occurrences de chaque port des interfaces VCI de type commande ou réponse, des compteurs ont été ajoutés à la description du crossbar. Pour alléger la simulation, la consommation dans les fils représentant les signaux de contrôle (1 bit) a été négligée. La consommation des fils de connexion va être ajoutée à celle des FIFO des entrées et des sorties du crossbar afin de calculer la consommation totale du crossbar.

Estimation de la consommation au niveau PVT

Au niveau PVT, les transferts de données se font de façon atomique. Ils sont considérés comme des activités de gros-grain et correspondent à plusieurs phases élémentaires de transfert de données. La figure 1.11 montre que l'énergie d'une transaction au niveau PVT correspond à la somme des consommations relatives aux transferts de plusieurs mots mémoire (comme c'est le cas de la Transaction 2). Chaque mot est responsable de l'activation de plusieurs ports (activités de grain-fin). La consommation d'énergie à ce niveau est calculée par paquet pouvant contenir un ou plusieurs mots. L'équation suivante présente l'énergie dissipée lors de la transmission d'un paquet d'une source i à une cible j:

$$E = \sum_{N} E_{i,j} \tag{1.13}$$

i et j: *Respectivement, le numéro de l'initiateur et de la cible*
N: *Nombre de mots transmis dans le paquet*
$E_{i,j}$: *Coût énergétique du transfert d'un mot entre i et j*
Le coût énergétique pour transférer un mot entre deux interfaces i et j est calculé d'une façon sommaire (pas au niveau de chaque port) avec l'équation suivante:

$$E_{i,j} = \alpha \cdot E_0(L_{i,j}) \tag{1.14}$$

Ici, α est le nombre de bits qui ont commuté entre 0 vers 1 et vice versa dans chaque interface. Dans le cas du protocole VCI, l'interface VCI commande est composée de 93 bits ($0 < \alpha \leq 93$) et celle de réponse est composée de 46 bits ($0 < \alpha \leq 46$). Dans notre travail, α est estimée à la moitié du nombre de bits de chaque interface. Au niveau PVT, des compteurs ont été ajoutés à la description du crossbar pour calculer le nombre de mots transférés entre les interfaces de type initiateur et les interfaces de type cible. Les occurrences obtenues seront multipliées par les coûts énergétiques pour déduire la consommation des fils de connexion. La valeur de cette consommation va être ajoutée à celle des FIFO des entrées et des sorties du crossbar afin de retrouver la consommation totale de ce composant.

1.5.5 Modèle de consommation de l'accélérateur matériel TCD-2D

L'objectif essentiel de l'utilisation des accélérateurs matériels dans la conception des systèmes embarqués est de pouvoir atteindre le maximum de performance entre termes de temps d'exécution et de consommation. Cet objectif est généralement atteint avec une implémentation figée (ASIC) sur la puce au prix d'un manque de flexibilité dans notre système. Pour atteindre un meilleur compromis entre performance et flexibilité, une nouvelle approche consiste à intégrer des ressources reconfigurables (FPGA) dans les systèmes embarqués [14]. Ces ressources seront utilisées pour implémenter en particulier des accélérateurs matériels.

La consommation d'énergie dans un accélérateur matériel dépend essentiellement de la fonctionnalité réalisée par le composant. Pour cela, il n'existe pas dans la littérature des modèles génériques pour l'estimation de la consommation dans ces composants. Par ailleurs, les accélérateurs matériels ont généralement des architectures dédiées et s'apprêtent donc très mal à une réutilisation de celles-ci. Pour cette raison, l'utilisation des modèles analytiques pour estimer les coûts énergétiques des activités pertinentes pour ce type de composant n'est pas réalisable. Ainsi, la méthode la plus adéquate pour caractériser les accélérateurs matériels, du point de vue de la consommation, est de réaliser des simulations de bas niveau.

Estimation de la consommation au niveau CABA

Pour estimer la consommation d'énergie de l'accélérateur matériel TCD-2D, nous avons synthétisé et implémenté le composant sur la plateforme FPGA STRATIX II EP2S60 (figure 1.12). L'outil Quartus d'Altera nous permet l'estimation de la consommation statique et dynamique. Pour notre accélérateur matériel, nous allons considérer deux modes de fonctionnement possibles. Le premier est le mode actif correspondant à l'exécution de la TCD-2D sur un bloc d'image de 8x8. Le deuxième est le mode inactif et correspond à l'état d'attente des données.

Pour évaluer le coût énergétique du mode actif, des expérimentations ont été menées sur plusieurs blocs d'image de différentes fréquences. L'objectif est d'estimer l'énergie dissipée par cycle dans ce mode de fonctionnement. Les résultats de simulation montrent une faible variation de consommation entre les différents blocs. Cette variation est inférieure à 5%. Pour cette raison, nous avons adopté une méthode d'évaluation d'énergie qui considère une valeur moyenne par cycle (E_{actif}). Pour évaluer le coût énergétique du mode inactif, l'outil Quartus fournit la consommation statique de l'architecture dissipée à l'état inactif. Le tableau 1.6 résume les coûts en puissance et énergie obtenus pour les modes actif et inactif à une fréquence de fonctionnement égale à 100MHz.

	Vdd (V)	f(MHz)	P_{active}(mW)	E_{active}(pJ)	$P_{inactive}$(mW)	$E_{inactive}$(pJ)
Accélérateur TCD-2D	1.1	100	2.13	21.3	0.51	5.1

Table 1.6: *La consommation de l'accélérateur matériel TCD-2D*

Au niveau CABA le modèle d'énergie suivant est utilisé pour estimer la consommation de l'accélérateur matériel TCD-2D:

$$E_{TCD-2D} = n_{running} \cdot E_{running} + n_{idle} \cdot E_{idle} \tag{1.15}$$

$n_{running}$ représente le nombre de cycles pendant les quels l'accélérateur matériel exécute la TCD-2D (actif) et n_{idle} le nombre de cycles où le composant est en attente (inactif). Deux compteurs, qui représentent les deux modes actif et inactif, ont été ajoutés à la description du composant. A la fin de la simulation les valeurs de ces compteurs seront multipliées par les coûts d'énergie correspondants pour calculer la consommation totale de l'accélérateur matériel.

Estimation de la consommation au niveau PVT

Au niveau PVT, la description du composant n'intègre pas un compteur de temps. En effet, l'accélérateur TCD-2D est considéré comme une cible qui répond aux requêtes du contrôleur DMA. L'information pertinente que nous pouvons récupérer est le nombre de blocs traités. Nous allons considérer que l'exécution de la TCD-2D sur un bloc d'image de 8x8 est l'activité pertinente de ce composant à ce niveau. Le coût énergétique de cette activité de gros-grain est estimé à partir des mesures du niveau RTL comme ceci a été décrit précédemment. Le calcul de la consommation de l'état inactif peut se faire à chaque intervalle de temps (en nano seconde ou en cycles) ou bien à la fin de la simulation. Dans le premier cas, l'accélérateur TCD-2D récupère le paramètre *time* des requêtes *read()* et *write()* transmises par le contrôleur

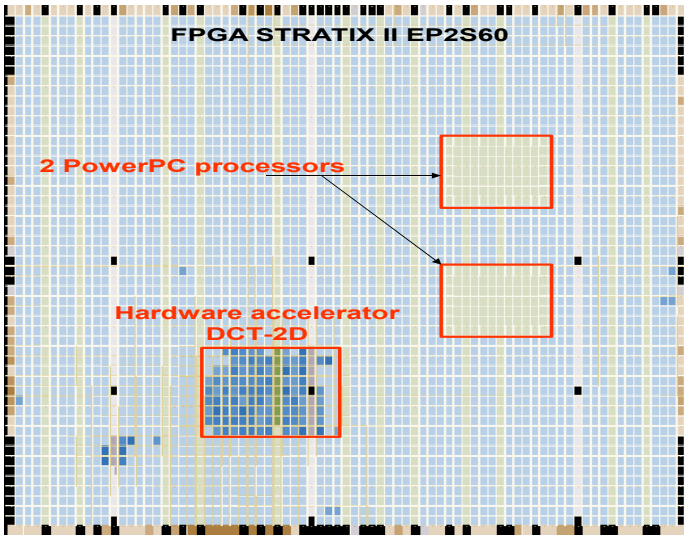


Figure 1.12: *L'accélérateur TCD-2D implémenté sur le FPGA STRATIX II EP2S60*

DMA afin de calculer son temps de repos entre deux requêtes successives. La consommation totale d'énergie de l'accélérateur matériel TCD-2D est donnée à la fin de l'exécution de l'application. Le temps de repos est égal à la différence entre le temps d'exécution de l'application et la somme des temps de traitement des blocs traités et nous utilisons le coût énergétique de l'état inactif mesuré précédemment pour calculer la consommation statique. Des compteurs ont été ajoutés à la description du composant au niveau PVT afin de calculer les occurrences des différentes activités. Au niveau PVT le modèle d'énergie suivant est adopté:

$$E_{TCD-2D} = n_{blocs} \cdot E_{bloc} + n_{idle} \cdot E_{idle} \tag{1.16}$$

n_{blocs} représente le nombre de blocs d'image 8x8 aux quels nous avons appliqué la TCD-2D et n_{idle} le nombre de cycles où l'accélérateur matériel est à l'état inactif.

1.6 Résultats expérimentaux

Les différents modèles de consommation développés dans les deux niveaux CABA et PVT ont été intégrés dans les deux environnements de simulations MPSoC correspondants. Ces 2 environnements peuvent être utilisés pour évaluer les différentes alternatives de l'espace de solutions architecturales en prenant en compte la performance et la consommation d'énergie.

Parmi les 3 sous-niveaux PVT présentés dans le chapitre précédent, nous avons choisi le sous-niveau PVT-TA pour intégrer les modèles de consommation des différents composants. En effet, ce sous-niveau présente le meilleur compromis entre l'accélération de la simulation et la précision d'estimation de performance. A l'opposé, le sous-niveau PVT-PA comporte une erreur d'estimation de performance pouvant atteindre 27%. Cette erreur influera sur la précision de l'estimation de la consommation. Par ailleurs à ce sous-niveau, l'architecture des processeurs n'est pas implémentée ce qui rend difficile l'estimation de la consommation de la partie calcul. L'intégration des modèles de consommation au sous-niveau PVT-EA permet de fournir sans aucun doute les estimations les plus précises. Néanmoins, les accélérations que nous avons obtenues à ce sous-niveau sont assez limitées.

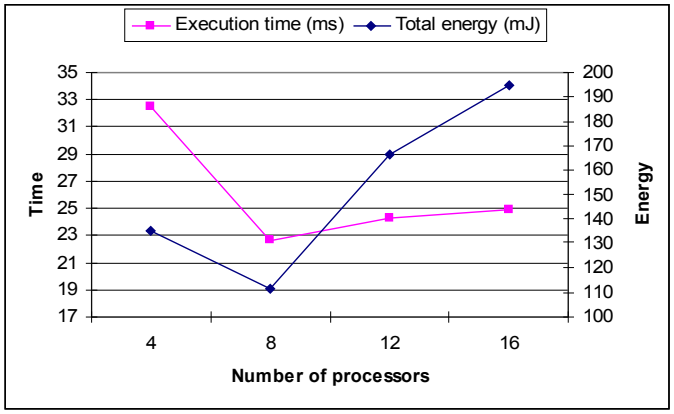


Figure 1.13: Variation des performances et de l'énergie en fonction du nombre de processeurs

Dans le chapitre précédent, nous avons mesuré pour le sous-niveau PVT-TA l'accélération de la simulation, la précision d'estimation de performance et l'effort de développement en comparaison avec le niveau CABA. Dans cette section, nous allons utiliser le même environnement expérimental et l'application H.263 pour évaluer la précision de l'estimation dans la consommation d'énergie du niveau PVT. Cette métrique sera présentée pour diverses tailles du cache de données et d'instructions et avec un nombre variable de processeurs. Notons cependant que notre environnement peut être utilisé pour déterminer, dans un intervalle de temps raisonnable et une précision acceptable, la configuration optimale pour d'autres paramètres architecturaux tels que: le type de processeur, le type de réseau d'interconnexion, etc.

1.6.1 Résultats de simulation au niveau CABA

Pour évaluer l'impact du nombre de processeurs sur les performances et la consommation d'énergie totale du système, nous avons exécuté l'application du codeur H.263 sur un système MPSoC formé de 4 à 16 processeurs. La taille des caches d'instructions et de données est fixée à 8 Ko et la fréquence des processeurs MIPS est fixée à 100 MHz. La figure 1.13 rapporte le temps d'exécution de l'application en ms et la consommation totale d'énergie en mJ estimés par le simulateur. En général, l'augmentation du nombre de processeurs fait diminuer le temps d'exécution de l'application. Ceci est le cas de notre système lorsque le nombre de processeurs passe de 4 à 8 processeurs où nous constatons une réduction de 30%. A l'opposé, le temps d'exécution augmente lorsque le nombre passe 12 de 16 processeurs du fait des collisions. En effet, les différents processeurs, partageant des ressources (cibles) communes, ne peuvent pas accéder en même temps à ces cibles. Du point de vue de la consommation, l'énergie totale du système décroît avec la diminution du temps d'exécution dans une première phase (entre 4 et 8 processeurs). Mais à partir d'une certaine limite (dans notre cas 8), l'augmentation du nombre de processeurs s'avère inefficace vis-à-vis de la consommation. En effet, là aussi, à cause de l'augmentation des conflits sur le réseau d'interconnexion des cycles d'attentes seront générés ce qui augmente la consommation totale.

Une deuxième étude a était faite sur la plateforme à 8 processeurs. L'objectif ici est d'évaluer l'impact des tailles des caches sur les performances et la consommation d'énergie totale du système. Pour cela, nous avons exécuté notre application en faisant varier la taille

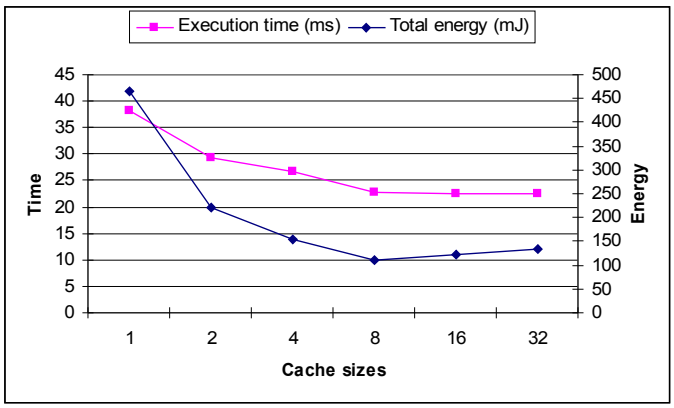


Figure 1.14: Variation des performances et de l'énergie en fonction de la taille des caches

des caches de données et d'instructions entre 1 Ko et 32 Ko. La configuration des caches adoptée est la suivante: associativité=1, taille du bloc= 32 octets et stratégie d'écriture est "Write-Through". La figure 1.14 présente les résultats de simulation. Généralement, plus grande est la taille du cache, plus petits sera le temps d'exécution. Les résultats dépendent cependant de la taille des tâches (code) et des données à traiter. Dans notre exemple, le passage de 1 ko à 2 ko fait augmenter les performances de 23% alors que pour des tailles supérieures à 8 ko ces performances ne changent pas. Du point de vue de la consommation, l'énergie totale du système décroît avec la diminution du temps d'exécution de l'application lorsque des tailles de caches de plus en plus grandes sont utilisées. Au-delà d'une certaine limite (8 Ko), la consommation du système à tendance à accroître.

La figure 1.15 présente la participation de chaque composant dans la consommation totale du système pour 8 processeurs. Les valeurs sont rapportées en pourcentage pour différentes tailles de mémoires caches. Dans notre MPSoC, les unités de calcul constituées par les processeurs MIPS R3000 et leurs caches représentent la première source de consommation dans le système. En effet, ces unités sont responsables de 37% à 59% de la consommation totale du système. Il est intéressant de noter que l'augmentation des tailles de caches réduit: le nombre de défauts de cache, le temps d'exécution des processeurs, les communications sur le réseau d'interconnexion et enfin les accès en mémoire centrale. Par conséquent, l'utilisation des caches de grandes tailles fait diminuer significativement la consommation d'énergie des autres composants. Pour les caches, la consommation devient de plus en plus importante avec l'accroissement de la taille à cause de l'augmentation des coûts énergétiques des activités (lecture, écriture et repos). Le crossbar consomme entre 25% (pour 32Ko) et 45% (pour 1Ko) de la consommation totale alors que la mémoire partagée d'instructions et de données consomme moins de 20%. L'accélérateur matériel TCD-2D et le contrôleur DMA sont responsables d'un faible pourcentage de la consommation totale. Cette valeur est autour de 4%.

Nos outils peuvent aussi être utilisé pour mesurer la puissance dissipée par le système est étudiée. Dans un circuit, la puissance consommée est responsable de la dissipation thermique qu'il faut prendre en considération pour des raisons de fiabilité. Ce problème est plus crucial pour les MPSoC qui intègrent plusieurs unités de calcul. La puissance dissipée peut être calculée directement à partir de l'énergie et du temps d'exécution donné par le nom-

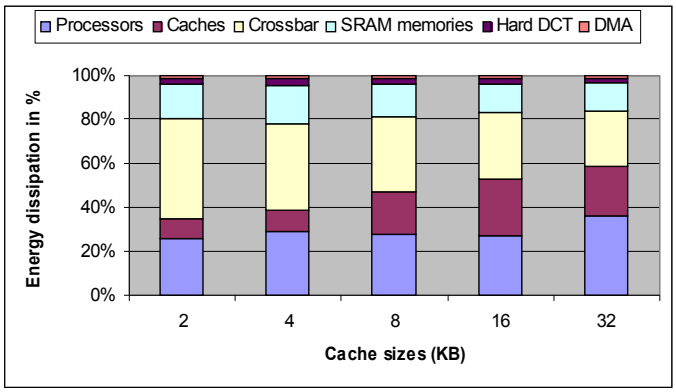


Figure 1.15: Consommation d'énergie dans les différents composants

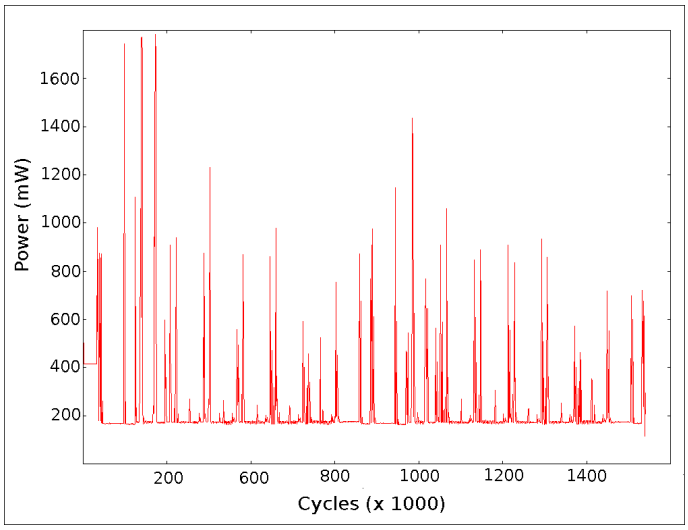


Figure 1.16: Dissipation de puissance dans le crossbar sur des intervalles de 1000 cycles

bre de cycles. L'environnement que vous avons développé permet de calculer la puissance dissipée par cycle ou par intervalle de temps dans chaque composant. Ceci nous permet de contrôler la dissipation thermique totale ainsi les pics de puissance qui diminuent le temps de vie des batteries. La figure 1.16 représente la consommation de puissance dans le crossbar pour des intervalles 1000 cycles, de cette façon nous pouvons vérifier le bon fonctionnement du circuit lors de l'exécution de l'application.

1.6.2 Résultats de simulation au niveau PVT

Au sous-niveau PVT-TA, nous avons intégré les différents modèles de consommation développés afin d'évaluer les solutions architecturales en se basant sur les deux critères d'énergie et de performance. Nous commençons cette section par la mesure de l'erreur dans l'estimation de l'énergie au sous-niveau PVT-TA. Cette erreur est calculée par rapport aux valeurs données dans le niveau CABA. Pour cela, nous avons exécuté notre application H.263 sur différentes configurations du MPSoC. La figure 1.17 donne l'erreur d'estimation

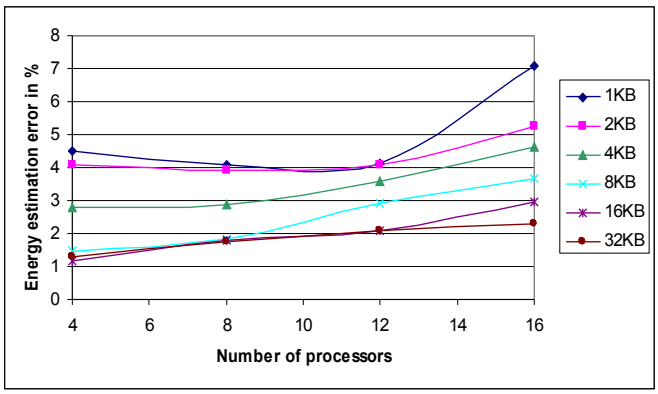


Figure 1.17: Erreur d'estimation d'énergie dans le niveau PVT

en fonction du nombre de processeurs et de la taille des caches.

Dans cette figure, nous pouvons voir que dans le niveau PVT, l'erreur d'estimation d'énergie est de l'ordre de 7%. La courbe de cette erreur a la même tendance que celle de l'erreur de performance. Ceci est dû principalement à l'impossibilité de détecter les contentions dans le réseau d'interconnexion. En effet, ces contentions sont responsables de plusieurs cycles d'attente, ce qui génère une consommation statique des composants à l'état inactif. L'addition de nouveaux processeurs au système ou l'utilisation de caches de tailles réduites amplifie les contentions sur le réseau d'interconnexion et fait augmenter par conséquent l'erreur d'estimation de l'énergie.

Pour localiser les sources d'imprécision dans nos modèles de la consommation, nous avons mesuré l'erreur d'énergie au niveau de chaque composant. La figure 1.18 illustre l'erreur, en pourcentage, mesurée sur la plateforme à 8 processeurs en faisant varier la taille des caches de données et d'instructions entre 1 Ko et 32 Ko. Pour des caches de petite taille, les modèles de consommation du processeur et de la mémoire SRAM sont responsables d'une erreur importante de l'estimation (respectivement 52% et 30%). Ce comportement change avec l'augmentation des tailles des caches qui entraîne une diminution des cycles d'attente du processeur ainsi que des accès à la mémoire SRAM. Pour les caches, leur contribution dans l'erreur totale devient importante à partir d'une certaine taille. Dans notre exemple, cette erreur est supérieure à 29% pour une taille de 16 ko. Pour les autres composants: le crossbar, le contrôleur DMA et l'accélérateur matériel TCD-2D, l'erreur est en général inférieure à 10%.

En conclusion, l'intégration des modèles de consommation au niveau PVT, nous a permis d'obtenir un environnement efficace pour l'exploration architecturale des systèmes MPSoC. Cet environnement offre une accélération de la simulation qui peut atteindre 18 en comparaison avec le niveau CABA et des erreurs d'estimation inférieures à 8%.

1.7 Intégration des modèles de consommation dans Gaspard

Pour faciliter l'évaluation de la consommation dans les systèmes MPSoC à différents niveaux d'abstraction, nous avons intégré les modèles d'énergie développés dans l'environnement de conception Gaspard. Cette intégration a commencé par une hiérarchisation de ces mod-

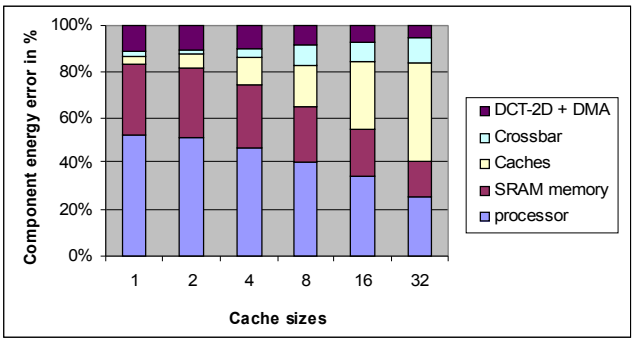


Figure 1.18: Erreur dans l'estimation de la consommation d'énergie dans les différents composants

èles dans une bibliothèque selon le type du composant et le niveau d'abstraction. Cette bibliothèque peut éventuellement intégrer des modèles de consommation conçus par d'autres chercheurs. Pour adapter ces modèles à nos composants, il est nécessaire d'identifier les activités pertinentes afin d'ajouter les compteurs d'occurrence nécessaires dans la description de chaque composant. Dans notre approche, à un composant donné peut lui être associé plusieurs modèles de consommation, offrant ainsi différents compromis entre la précision et la rapidité d'estimation. Pour les composants dérivant d'autres bibliothèques et ayant des architectures proches des nôtres, le concepteur a la possibilité d'utiliser nos modèles de consommation pour l'évaluation de l'énergie.

Dans nos transformations dans l'environnement Gaspard, depuis la description de haut niveau jusqu'à la phase de génération de code, un lien sera créé entre les composants et les modèles de consommation. Dans une première phase, ces derniers nécessitent la spécification des paramètres d'architecture et de la technologie. Au cours de la simulation, les valeurs des compteurs sont transmises aux modèles de consommation permettant de calculer la dissipation d'énergie par cycle (CABA) ou par intervalle de temps (PVT). La dissipation totale sera calculée à partir des valeurs des compteurs transmises à la fin de la simulation. Dans le chapitre suivant, l'implémentation de notre proposition en se basant sur une approche dirigée par les modèles sera détaillée.

1.8 Conclusion

Une exploration architecturale fiable pour les systèmes MPSoC nécessite des outils d'estimation de performance et de la consommation d'énergie à différents niveaux d'abstraction. Ceci permet une réduction dans le temps de conception du système et une meilleure exploration de l'espace des solutions.

Dans ce chapitre, notre contribution a consisté à enrichir les simulateurs MPSoC décrits aux niveaux CABA et PVT par des modèles de consommation d'énergie. Nous avons proposé une méthodologie hybride d'estimation de la consommation basée sur des simulations de bas niveau et des modèles analytiques offrant un niveau acceptable de précision et de flexibilité. L'intégration du critère de la consommation dans notre travail nous a permis d'obtenir un environnement flexible pouvant être utilisé dans une exploration rapide et précise des systèmes MPSoC.

Dans notre travail, nous avons développé des modèles de consommation pour un nombre réduit de composants. Notre objectif était de valider la méthodologie de modélisation de la consommation et d'obtenir des estimations de bonne qualité. Dans les perspectives, il serait intéressant d'appliquer cette méthodologie pour autres types de processeurs et de réseaux d'interconnexion et ainsi enrichir la liste des IP pouvant être utilisés dans l'environnement Gaspard.

Les modèles des composants ainsi que les outils d'estimation de performance et de consommation d'énergie décrits dans les trois chapitres précédents, ont pour but de faciliter l'exploration sur plusieurs niveaux. Afin de faciliter l'utilisation de ces modèles et outils, nous proposons dans le le chapitre suivant l'intégration de ces derniers dans l'environnement Gaspard basé sur une méthodologie d'ingénierie dirigée par les modèles pour la conception des systèmes MPSoC.

Bibliography

[1] ALLIANCE home page. http://www-asim.lip6.fr/recherche/alliance/.

[2] Mentor home page. http://www.mentor.com.

[3] ARM Technologies Consortium. http://www.arm.com/products/CPUs/.

[4] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: a framework for architectural-level power analysis and optimizations. In *The 27th annual international symposium on Computer architecture*, 2000.

[5] CACTI Home Page. http://research.compaq.com/wrl/people/jouppi/CACTI.

[6] Y. Cao, T. Sato, M. Orshansky, D. Sylvester, and C. Hu. New paradigm of predictive MOSFET and interconnect modeling for early circuit simulation. In *IEEE Custom Integrated Circuits Conference*, 2000.

[7] Intel StrongARM SA-1100 Microprocessor for Portable Applications. http://netwinder.osuosl.org/pub/netwinder/docs/intel/datashts/27808706.pdf.

[8] J. Laurent. *Estimation de la consommation dans la conception système des applications embarquées temps réels*. PhD thesis, Laboratoire d'Electronique des Systèmes TEmps Réel, Université de Bretagne Sud, 2002.

[9] J. Li and S.-L. Lu. Low power design of two-dimensional dct. In *Ninth Annual IEEE International ASIC Conference and Exhibit*, 1996.

[10] M. Mamidipaka, K. Khouri, N. Dutt, and M. Abadir. IDAP: a tool for high-level power estimation of custom array structures. In *International Conference on Computer Aided Design*, 2003.

[11] M. Mamidipaka, K. Khouri, N. Dutt, and M. Abadir. Analytical models for leakage power estimation of memory array structures. In *international conference on Hardware/software codesign and system synthesis*, 2004.

[12] S. Marc, K. Reiner, L.-P. Josep, U. Theo, and V. Mateo. Transistor count and chip-space estimation of simplescalar-based microprocessor models. In *Workshop on Complexity-Effective Design*, 2001.

[13] M. Miyama, S. Kamohara, M. Hiraki, K. Onozawa, and H. Kunitomo. Pre-silicon parameter generation methodology using BSIM3 for circuit performance-oriented device optimization. *IEEE Transaction on Semiconductor Manufacturing*, 14(2):134–142, May 2001.

[14] MORPHEUS project. Multi-purpose dynamically reconfigurable platform for intensive heterogeneous processing. http://www.morpheus-ist.org/.

[15] NEC Data sheet. http://necel.com/memory/en/download/M11657EJCV0DS00.pdf.

[16] P. R. Panda and N. D. Dutt. Memory architectures for embedded systems-on-chip. In *9th International Conference High Performance Computing*, 2002.

[17] Power Analyzer home page. www.eecs.umich.edu/panalyzer/.

[18] A. Sinha and A. Chandrakasan. Jouletrack - a web based tool for software energy profiling. In *Proceedings of the 38th DAC Conference*, 2000.

[19] T. T. Ye, G. D. Micheli, and L. Benini. Analysis of power consumption on switch fabrics in network routers. In *The 39th Design Automation Conference*, 2002.