République Tunisienne
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université de Sfax
Ecole Nationale d'Ingénieurs de Sfax
Département de Génie Informatique

Cycle de Formation doctorale dans la discipline *Ingénierie des Systèmes Informatiques*

Kaïs Loukil
Thèse Doctorat
N° d'ordre:

THESE

présentée à

L'Ecole Nationale d'Ingénieurs de Sfax

en vue de l'obtention du

DOCTORAT
Dans la discipline
Ingénierie des Systèmes Informatiques

par

Kaïs LOUKIL

Approche de gestion de performances/contraintes pour les systèmes embarqués temps réel

Soutenu le 06/12/2011, devant la commission d'examen:

M.	Mohamed Jmaiel	***Président***
M.	Habib Youssef	*Membre*
M.	Samir Ben Ahmed	*Membre*
M.	Jean Philippe Diguet	*Membre*
M.	Mohamed Abid	*Membre*

Remerciements

Cette thèse a été réalisée au sein du laboratoire Computer & Embedded System, (CES) dirigé par ***Pr. Mohamed AbID****.*

Il m 'est agréable de saisir cette occasion pour adresser mes sincères remerciements et ma gratitude la plus profonde à mon directeur de thèse ***Pr. Mohamed AbID****, professeur à l'ENIS pour son aide précieuse, ses conseils bienveillants, ses compétences, ses grandes expériences et ses qualités humaines m'ont été d'une grande importance et m 'ont permis de mener à bien ce travail. Qu'il trouve ici l'expression de mes profondes reconnaissances.*

Je tiens à remercier chaleureusement mon encadrant et ami ***Nader Ben Amor*** *maitre assistant à l'ENIS. Merci Nader pour l'intérêt que tu as porté à mon travail, tes remarques et tes conseils utiles pour la réussite de cette thèse.*

Mes remerciements s'adressent à ***Pr. Samir Ben Ahmed*** *et* ***Pr. Habib Youssef*** *d'avoir accepté d'êtres les rapporteurs de cette thèse. Je tiens à remercier également,* ***Pr Mohamed Jmaiel*** *qui a accepté de présider le jury et* ***Pr Jean Philippe Diguet****, qui a bien voulu juger ce travail et pour l'attention qu'il m'a accordée.*

Je remercie également ***Mouna Ben Saïd*** *et* ***Lina Jarboui*** *pour leurs contributions ainsi que tous les membres du laboratoire* ***CES*** *pour leurs coopérations et encouragements.*

Une pensée particulière à tous mes ***enseignants*** *qui ont participés à ma formation.*

Je n'oublierai jamais tous ceux qui partagent ma vie personnelle et qui m'ont soutenu durant cette thèse, et bien souvent durant les années antérieures. Je pense évidement à mes ***parents*** *et ma* ***famille*** *proche.*

Table des matières

Liste des figures

Liste des tableaux

Glossaire

SoC	System-On-a-chip
ASIC	Application Specific Integrated Circuit
ASIP	Application Specific Integrated Processor
FPGA	Field programmable gate array
DSP	Digital Signal Processing
MIPS	Million d'instructions par seconde
QoS	Quality of service
3D	Trois dimensions
PDA	Personal Digital Assistant
RTOS	Real time operating system
VHDL	Very High Description language
OS	Operating System
SW	software
HW	hardware
Texe	Temps d'exécution
IP	Intellectual Properties
VSIA	Virtual Identification Soft IP
TTM	Time To Market
SOPC	System on programmable chip
DDV	Durée de vie
ISR	Interrupt Service Request
EDF	Earliest deadline first

CHAPITRE 1: Introduction générale

1 Motivation

Depuis longtemps, on assiste à un fort accroissement du rythme des innovations technologiques dans le domaine des systèmes électroniques. Ce progrès technologique permet de répondre aux exigences des applications de plus en plus complexes de traitement de l'information (notamment dans le domaine multimédia). Il rend possible l'utilisation de ces applications non seulement sur les ordinateurs de bureau, mais aussi sur des systèmes embarqués nomades de faible encombrement. La conception de ce type de systèmes très populaire est actuellement au cœur d'enjeux économiques très importants, liés à l'expansion du marché des applications « mobiles » (téléphones portables, tablette, terminaux vidéos, etc.), à la réduction des délais de mise sur le marché et la concurrence farouche du domaine.

Jouissant des progrès scientifiques dans les domaines du traitement multimédia, et du traitement du signal, les fonctionnalités offertes par ces systèmes sont devenues de plus en plus variées et complexes. Elles demandent par conséquent des capacités de calcul de plus en plus importantes. Cet accroissement persistant en complexité est également justifié par l'évolution de la technologie qui permet de réaliser de tels systèmes sur une seule puce de silicium. En effet, selon la loi de Moore [Moo65], la densité d'intégration augmente de 50% chaque année. Ce qui permet de créer des systèmes mixtes (logiciel/matériel) hétérogènes comportant divers modules (comme des processeurs, des DSP, des ASIP, etc) sur une même puce (SoC).

Ces nouvelles générations de systèmes posent néanmoins de nouveaux défis aux concepteurs. Avec un tel niveau de complexité, les phases de conception de ces systèmes sur puce deviennent de plus en plus ardus (spécification initiale, simulation, fabrication, tests …) vu le nombre et la nature des applications à gérer, la nature hétérogène de l'architecture et surtout les délais courts de temps de mise sur le marché (time to market) avec toujours la nécessité d'un « zero default ».

Afin de mettre en place un système informatique, les concepteurs ont longtemps disposé d'un choix restreint à deux alternatives. La première consiste à mettre en place un système multiprogrammé via un microprocesseur [Zhi08]. Cette solution est moins couteuse mais elle n'arrive pas assurer l'exécution de toutes les fonctionnalités du système avec la qualité requise. La seconde, se base sur la réalisation d'un circuit spécifique à l'application développée (ASIC) [Kri09]. De même cette solution souffre de quelques handicaps tels que la complexité de la phase de conception et le temps de fabrication du produit final. Comme solution la technologie programmable offre plus de flexibilité au produit final avec un temps de conception et un coût

assez réduit.

Les circuits reconfigurables correspondent à des circuits matériels dont l'architecture peut être modifiée en fonction de l'application à développer [Gan10, Kon05]. Les plus populaires sont les circuits FPGA. Ces circuits permettent de faciliter la phase de mise en place d'un nouveau système multimédia (de la spécification au prototypage). En effet, leur architecture est riche et répond aux besoins des applications actuelles. De plus, leur nature reconfigurable réduit les phases de test, le coût associé aux erreurs de conception et le temps de mise sur le marché. Ces architectures se présentent comme une solution intéressante au défi des systèmes sur puce. Elles ont permis la conception de nouvelles applications bénéficiant de leurs caractéristiques propres telles qu'un fort parallélisme matériel et des possibilités de reconfiguration statique et/ou dynamique.

Malgré toutes ces améliorations technologiques, l'implémentation d'applications multimédia sur un système embarqué reste une tâche compliquée qui doit répondre à un ensemble de contraintes antagonistes.

L'une des contraintes est la complexité des applications supportées. En effet, ces nouvelles fonctionnalités sont exigeantes aussi bien au niveau puissance de calcul que capacité mémoire. Elles sont en même temps très consommatrices d'énergie. La norme de compression MPEG4 par exemple est d'une complexité beaucoup plus importante que ses deux prédécesseurs MPEG1 ou MPEG2 puisqu'elle est destinée à couvrir un ensemble plus vaste d'applications [Tou00]. A titre d'exemple, l'encodage (profil simple) de la séquence "Weather" à 10 image/s au format QCIF (qualité moyenne de vidéo) demande un total de 1500 MIPS avec plus de 250 millions d'accès mémoire [Kim03]. Cette augmentation de la complexité implique aussi une consommation en énergie plus importante, ce qui limite l'autonomie des systèmes embarqués.

Une autre contrainte est la gestion de la consommation. C'est une tâche très difficile compte tenu le nombre de facteurs favorisant son augmentation. D'une part, le nombre et la nature des fonctionnalités multimédia supportées par un système embarqué ne cessent d'évoluer. D'autre part, le progrès technologique n'a pas apporté de solutions acceptables à ce problème. En effet, si la technologie autorise la réduction de la taille des transistors permettant ainsi de diminuer les tensions d'alimentation et donc la consommation dynamique, elle entraînera une augmentation relative des courants de fuite et donc de la consommation statique négligée auparavant. En plus l'augmentation de la densité d'intégration sur la même puce permet de réaliser des architectures assez complexes ; ce qui favorise la consommation [Kuh98].

L'exécution temps réel est une contrainte pour les systèmes temps réel. Diverses applications nécessitent une exécution en un temps limité et continu dans le temps pour des raisons de sécurité (dans le cas par exemple d'un système de navigation aérien) où pour des raisons de confort visuel (dans le cas de la compression vidéo par exemple, il faut assurer une cadence de 25 image/s soit un temps de traitement de 40 ms/image).

Une autre contrainte des systèmes embarqués est liée à la nature mobile des systèmes électroniques actuels. Il est nécessaire de tenir en compte un autre paramètre externe cette fois qui est l'environnement où évolue le système mobile. Perturbations atmosphériques, liaison radio variable, flux de données changeant sont des exemples de paramètres qui peuvent influencer de façon importante sur le fonctionnement du système et la qualité des services offerts (QoS).

De ce fait, ces systèmes doivent être d'une part performants pour pouvoir traiter les applications multimédia complexes et d'autre part flexibles pour s'adapter à l'environnement externe variable non seulement pour respecter les contraintes temps réel, mais aussi afin de préserver les ressources d'énergie du système et respecter ainsi la contrainte de durée de vie [Lic06, Kri08]. Le challenge consiste à concevoir des systèmes qui donnent une bonne qualité de service avec une consommation limitée d'énergie. Mais ce but de conception est difficile à atteindre puisque pour avoir une meilleure qualité de service il faut utiliser au maximum les ressources du système ce qui augmente la consommation du système. Il apparait ainsi important de pouvoir moduler l'utilisation des ressources matérielles selon les besoins de l'application d'une part et également en tenant compte des paramètres externes au système d'autre part.

Tirant partie de ce fait, les concepteurs des systèmes embarqués se trouvent devant un compromis entre la QoS à fournir et l'utilisation des ressources d'énergie et de calcul. Il est alors nécessaire de définir des systèmes adaptatifs qui permettent d'adapter leur fonctionnement non seulement suivant les contraintes du système mais aussi suivant les préférences de l'utilisateur et l'état de l'environnement externe.

2 Contribution de la thèse

Le but de la thèse est de définir un système multimédia adaptatif capable de gérer de façon autonome et efficace ces ressources d'énergie et de calcul pour produire un maximum de QoS tout en respectant les contraintes d'énergie et performance.

Notre approche prend en compte un certain nombre de contraintes qui agissent sur les performances d'un système multimédia embarqué pendant son exécution. Elles se situent à

différents niveaux:

- Type de données traitées. Une tâche peut voir ces caractéristiques (temps d'exécution, occupation mémoire, consommation) changer en fonction des données traitées (valeur, quantité, type).
- Influence du contexte dans lequel évolue le système embarqué. L'augmentation du taux d'erreur de transmission par exemple des données peut imposer le changement de l'algorithme de décodage et de correction utilisé et donc une augmentation de la consommation.
- Influence des choix d'applications. L'utilisateur peut privilégier l'exécution de certaines fonctionnalités de son système mobile aux dépends d'autres qu'il peut désactiver. Ceci engendre des changements de la consommation d'énergie à une sollicitation plus importante des ressources du système.
- Influence du niveau d'énergie de la batterie. La quantité d'énergie disponible dans le système peut imposer un mode de fonctionnement particulier ayant un niveau de performance limité.

Diverses techniques ont été proposées [Kan11, Pha04, Yua06] pour le respect des contraintes du système tout en donnant une meilleure qualité de service. Ces techniques peuvent intervenir dans l'une des couches application, système d'exploitation ou architecturale telles que l'ajustement de la tension d'alimentation ou la fréquence de fonctionnement du système, la gestion optimisée des accès mémoires, l'utilisation des techniques d'ordonnancement de basse consommation et la modification de l'apparence d'un objet 3D sur l'écran.

Nous proposons dans cette thèse une nouvelle approche multi couches combinant l'adaptation au niveau RTOS, applicatif et architectural.

Le modèle d'adaptation proposé doit être performant sans pour autant être trop complexe pour ne pas contribuer à dégrader les performances du système et ses ressources d'énergie.
Dans ce contexte se situe notre travail qui consiste à ajouter une couche middleware (couche intergiciel entre la couche système exploitation et application) permettant l'auto adaptation du système. Nous proposons une approche originale et générique d'adaptation qui comporte essentiellement deux gestionnaires (global manager and local manager). Le gestionnaire global peut intervenir dans les trois couches afin de répondre aux grandes variations des contraintes du système (QoS et énergie). Le gestionnaire local intervient seulement dans les couches application et système d'exploitation. Il est mis en place pour contrôler le respect de la contrainte temps réel

du système. En cas de besoin ce dernier peut interroger le gestionnaire global pour reconfigurer la totalité du système s'il ne peut plus modifier les paramètres applicatifs de l'application pour résoudre le problème localement.

Le principe de fonctionnement de cette approche se base sur l'utilisation d'un jeu de configurations pour chaque application, pré-caractérisées hors ligne. A chaque fois que cette approche détecte un non respect des contraintes du système, elle intervient pour choisir une nouvelle configuration pour chaque application présente sur le système. Compte tenu du nombre croissant d'applications que les systèmes actuels peuvent exécuter simultanément et la présence de plusieurs configurations pour chaque application notre approche doit résoudre un problème NP-complet pour trouver une combinaison de configuration qui lui permet de respecter ses contraintes. Partant du fait que la tâche d'adaptation ne doit pas dégrader les performances du système on a eu recours à des méthodes d'optimisation (algorithme génétique et recuit simulé) pour résoudre ce problème NP-complet.

3 Organisation du document

Le manuscrit de cette thèse est organisé en six chapitres :

Chapitre 2: conception des systèmes sur puce adaptatifs : état de l'art

La première partie du deuxième chapitre est consacrée à la présentation du flot de conception traditionnel des systèmes sur puce ainsi que les facteurs qui ont favorisé l'ajout de l'aspect adaptatif. Dans la deuxième partie de ce chapitre, une étude sur les approches d'adaptations existantes dans la littérature sera présentée ; nous présentons aussi l'apport et les limites de chacune d'entre elles. Nous clôturons cette partie par une synthèse des travaux existants et l'introduction de l'approche d'adaptation multicouche proposée.

Chapitre 3 : approche d'adaptation multicouche

Ce chapitre est consacré à la description des étapes et les différentes techniques qui ont conduit à la mise en place de l'approche d'adaptation. A ce niveau, nous détaillons les trois étapes (observation, adaptation et mise en place de la base des configurations) nécessaires pour la mise en place de l'approche d'adaptation proposée. Au niveau de la tâche d'observation, une nouvelle technique de contrôle du respect de la contrainte temps réel est proposée. Pour la réalisation de l'étape d'adaptation, on a présenté quelques méthodes d'optimisation pour le choix des configurations adéquates du système. Afin de mettre en place la base des configurations, une approche de partitionnement hardware software a été proposée ainsi qu'un ensemble de modèles qui permet de caractériser chaque configuration en termes de consommation, temps d'exécution et

niveau de qualité de service qui ont été développés.

Chapitre 4 : étude de cas

Le quatrième chapitre présente les éléments nécessaires de notre étude de cas faite à travers l'application de synthèse d'images 3D et l'environnement de conception Hardware/software d'Altera. Nous clôturons ce chapitre par la présentation du système d'exploitation temps réel MicroC_OS-II.

Chapitre 5 : expérimentation et validation

Le cinquième chapitre illustre, dans sa première partie, les étapes nécessaires pour la mise en place de l'approche décrite dans le troisième chapitre sur la plateforme de conception. La deuxième partie illustre les résultats expérimentaux de l'exécution de l'application de synthèse d'images 3D sur la plateforme conçue.

Chapitre 6 : conclusions et perspectives

Nous concluons cette thèse par le bilan des travaux effectués et nous détaillons les contributions apportées ainsi que la réponse à la problématique abordée avant de proposer quelques perspectives à nos travaux.

CHAPITRE 2 : Conception de système sur puce adaptatif : état de l'art

1 Introduction

La complexité croissante des applications actuelles nécessite des performances de plus en plus importantes, par ailleurs, afin de répondre à la contrainte imposée par le marché (time to market), la conception des systèmes embarqués multimédia impose l'utilisation des méthodes de conception spécifiques et performantes. Ces méthodes s'intègrent dans les méthodes de conception mixte logiciel/matériel (codesign).

Par ailleurs la diversité des contraintes auxquelles doivent répondre les systèmes embarqués (augmentation de la puissance de calcul, diminution de la consommation d'énergie, réduction des coûts, flexibilité), l'utilisation des méthodes de conception traditionnelle ne répond plus aux besoins des concepteurs. Des travaux de recherche sont menés afin de combler les limites en ajoutant de nouveaux facteurs dans les méthodes de conception traditionnelle.

Le but de ce chapitre est de présenter les notions de base reliées au domaine de conception des systèmes sur puce. Nous énumérons également, les limites du flot de conception traditionnelle. Nous terminons la première partie de ce chapitre par la présentation de nouvelles tendances pour la conception des systèmes embarqués. La deuxième partie de ce chapitre sera consacrée à la présentation des différents niveaux d'adaptation ainsi que quelques approches existantes qui traitent la notion d'adaptation dans les systèmes embarqués.

2 Système embarqué

Tout d'abord nous définissons l'élément de base de notre travail qui est le système embarqué (plus spécifiquement un système sur puce « SoC en anglais System-on-Chip »).

Un SoC est un système complexe et indépendant sur une seule puce. Il contient au moins un processeur (partie SW), de la mémoire sur puce, des composants spécifiques pour un traitement donné (accélérateur ou coprocesseur matériel HW), des périphériques externes (clavier, écran, interface d'entrées, sorties) peuvent compléter le fonctionnement du système [Hen99]. Il intègre dans la plupart des cas une interface homme/machine et il est géré par un système d'exploitation puisqu'un SOC peut être interfacé avec le monde réel. Il peut souvent incorporer des composants analogiques.

2.1 Domaines d'application

Actuellement, les systèmes embarqués ont envahi beaucoup de domaines tels que l'astronautique (satellite artificiel, fusée, sonde spatiale), le militaire (fusée), le transport (aéronautique, automobile, avionique) et surtout la télécommunication (téléphonie, routeur, pare-feu, serveur de temps, téléphone portable). Cette large gamme de domaine d'utilisation englobe aussi les produits d'électroménager (télévision, four à micro-ondes), d'impression (imprimante multifonctions, photocopieur), d'informatique (disque dur, lecteur de disquette), le multimédia (console de jeux vidéo, tablette). D'autres domaines d'application prospèrent de l'évolution des systèmes embarqués dont on peut citer les guichets automatiques bancaires, l'équipement médical, l'automate programmable industriel ou la métrologie [Fré00].

2.2 Caractéristiques des systèmes embarqués

2.2.1 Encombrement

La plupart des systèmes embarqués actuels sont conçus pour répondre à une contrainte d'encombrement (petite taille et faible poids) tels que les téléphones portables les PDA etc). La fabrication de ces systèmes fait appel à une technologie d'électronique et de logiciel portable qui favorise la réduction aussi bien de l'encombrement que de la consommation. Par conséquent, la mise en place d'un système embarqué de faible surface qui englobe de l'électronique numérique, analogique et des composants radio fréquence est une tâche assez complexe [Ben07].

Cette caractéristique peut limiter les fonctionnalités offertes vu que les composants doivent être d'une taille assez petite. Par exemple on ne peut pas utiliser un ventilateur pour le refroidissement des composants.

2.2.2 L'autonomie

On dit qu'un système est autonome lorsqu'il dispose de toutes les ressources qui assurent son fonctionnement. Il s'agit de ressources matérielles (processeur, mémoire, etc), de ressources logicielles (OS, applications) et de ressources d'énergie qui peuvent être continues ou rechargeables à travers des cumulateurs d'énergie pour pouvoir fonctionner [Fré00, Jal09].

Un grand nombre de systèmes sur puce actuels sont mobiles et fonctionnent avec des ressources d'énergie limitées, il est donc extrêmement important de réduire au

maximum leur consommation afin d'augmenter leur durée de vie.

2.2.3 Le temps réel

La plupart des systèmes sur puce actuels communiquent avec leur environnement externe afin de recevoir les données à traiter ou d'envoyer des consignes suite à un traitement effectué. Dans certains cas, la validité d'un résultat dépend de l'instant de son arrivée (échéance). Ce type de système est appelé système temps réel [Aud 04, Bru06].

On distingue le temps réel dur qui correspond à un résultat catastrophique lors du non respect des échéances comme dans le cas des systèmes de transport. On parle de temps réel mou si le non respect des échéances s'accompagne par une dégradation de la qualité du service fourni par le système. Par exemple un GSM fait un retard à chaque décodage de trame le système devient inexploitable et les paroles des utilisateurs ne peuvent pas être synchrones.

Deux techniques peuvent être distinguées [Fré00]: le développement monolithique et l'utilisation d'un système d'exploitation temps réel. Le développement monolithique est fait dans un langage de bas niveau. Il consiste à écrire un seul programme dans le système qui aura la totalité de la charge de travail du système. Les contraintes temporelles sont prises en compte lors de l'écriture du programme. L'utilisation des systèmes d'exploitation temps réel vient pour combler les défauts de la première technique. Elle permet l'exécution de plusieurs applications dans le système et elle offre les mécanismes nécessaires pour garantir le respect des contraintes du système. Cette deuxième méthode souffre aussi de quelques limites car la plupart des RTOS existants sont spécifiques et avec un code source fermé et non extensible.

2.2.4 Qualité de service

Les systèmes embarqués évoluent généralement dans des conditions environnementales imprévisibles et souvent non maitrisables vu qu'ils sont portables. La plupart d'entre eux sont incorporés dans des systèmes mobiles. Ils sont donc, soumis à des variations et à d'autres contraintes environnementales qui peuvent causer des défaillances : radiation, vibration, corrosion, chocs, variation d'alimentation, interférences radio fréquence, humidité, température etc. Il est donc nécessaire de prendre en compte l'impact de la variation des conditions environnementales lors de la conception de ces systèmes.

Les systèmes embarqués sont de plus en plus sophistiqués et utilisés dans des domaines assez critiques dans lesquels un disfonctionnement peut causer des nuisances, des pertes économiques et voir même des catastrophes sur la vie de l'être humain et l'environnement notamment dans le domaine médical, ou le nucléaire ou le domaine de transport.

Généralement, les utilisateurs des systèmes embarqués sont très exigeants en termes de fiabilité, de robustesse et de QoS. La plupart des utilisateurs acceptent un temps de disfonctionnement de l'ordinateur par exemple pour quelques heures à cause d'une panne ou de coupure de tension électrique. Par contre, ils sont généralement beaucoup moins patients vis-à-vis du disfonctionnement des systèmes incorporés sous forme de système embarqué et surtout dans le domaine de télécommunication par exemple.

2.2.5 Complexité

Grâce à l'augmentation du taux d'intégration, les systèmes embarqués peuvent avoir des architectures matérielles trop complexes suivant les besoins de l'application. Un système embarqué peut contenir un processeur, de la mémoire et d'autres composants pour assurer les performances exigées par les applications. Il peut contenir plus qu'un processeur dans une seule puce voire des dizaines de processeurs. Toutes ces évolutions ont poussé les concepteurs à mettre en place des outils de conception plus sophistiqués pour pouvoir mettre en place ce type d'architecture assez complexe.

3 Conception de système embarqué

La conception des systèmes embarqués nécessite généralement une très grande quantité de travail manuel et une grande expertise pour choisir l'architecture adéquate, écrire les modules de gestion de périphériques, concevoir les interfaces de communication et/ou configurer les systèmes d'exploitation commerciaux. A ce stade, le concepteur se trouve face à la tâche la plus difficile qui consiste à faire fonctionner l'ensemble de ces éléments qui sont conçus sur mesure pour répondre aux exigences d'une application.

Afin de réduire l'effort de conception et les risques d'erreurs de conception, une des techniques adoptées est l'utilisation des composants prédéfinis (IP). L'organisation VSIA (VSI Alliance) a proposé une méthodologie basée sur l'assemblage d'IPs préconçus [Seo10]. Malgré les inconvénients de ce type d'approche un grand nombre de méthodologies adopte ce concept en utilisant des IP et des interfaces de communication

standard de HW et de SW standard et paramétrables. Les pénalités de performances de cette solution architecturale sont acceptables et ce afin de répondre à une autre contrainte plus importante qui est bien évidemment le temps de mise sur le marché TTM et le prix final du produit.

Dans le but d'accélérer le flot de conception des systèmes embarqués, plusieurs travaux ont été menés. Ces travaux se basent sur l'utilisation de nouveaux outils capables d'automatiser le processus de conception. Ces outils se concentrent sur l'automatisation du raffinement de la communication et la réutilisation de blocs préconçus avec la génération automatique des interfaces. Ces outils utilisent différents flots de conception. On se contente ici de présenter le flot traditionnel.

Le flot de conception traditionnel est généralement constitué de trois étapes principales: (1) la spécification, (2) le partitionnement, et (3) la validation conjointe HW/SW.

3.1 Flot de conception Logiciel/Matériel

Le flot de conception SOPC traditionnelle est représenté par la Figure 1. Il admet comme entrée une spécification fonctionnelle de l'application et fournit en sortie une architecture adéquate qui répond aux différentes contraintes de l'application.

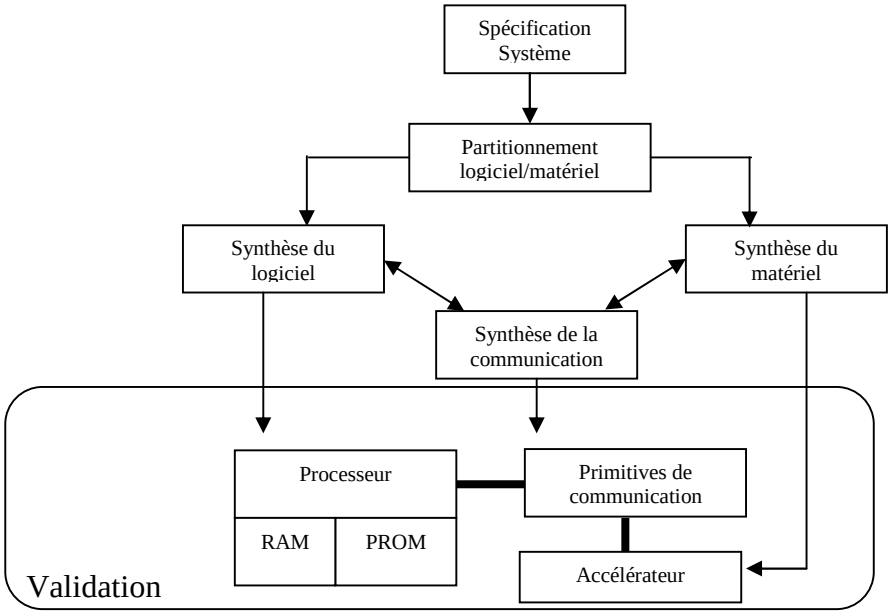

Figure 1: Flot de conception

Le flot de conception des SOPCs de la Figure 1 regroupe essentiellement trois étapes :

1. En partant d'une spécification fonctionnelle complète du design, le concepteur partitionne l'application en utilisant les outils adéquats ou manuellement en une partie logicielle et une autre matérielle. L'optimisation des techniques de partitionnement constitue un thème de recherche très sollicité par les scientifiques [Ara05, Cat01]. Ces activités de recherche s'intéressent surtout à l'automatisation de cette étape.

2. Une fois le partitionnement effectué, on implémente la partie matérielle en utilisant un langage de description matérielle telle que VHDL, VERILOG ou SystemC. La synthèse de la partie matérielle est réalisée par des outils de conception assistée par ordinateur (CAO) tels que Design Compiler de Synopsys [Syn], Allegro de Cadence [Cad] et Agility Compiler de Celoxica [Cel]. La partie logicielle en revanche, sera implémentée en langage évolué par exemple C/C++. La compilation de la partie logicielle dépend du processeur cible. Ensuite, on passe à la synthèse de la communication des différents blocs du système. La plus part des systèmes sont conçus à base d'IP qui sont des composants matériels et logiciels déjà existants dans la bibliothèque d'IPs de l'environnement de conception il est donc nécessaire de mettre en place les mécanismes de communication entre eux. La synthèse des communications permet de raffiner les interfaces des sous-systèmes communicants. Ce raffinement se fait d'une manière interactive jusqu'à figer les mécanismes de communication (protocole, contrôleur, interface).

3. Dans la phase finale du flot, on injecte le SOPC sur une plateforme adaptée de type FPGA. Cette plateforme représente un environnement idéal pour l'implémentation des SOPC contenant un ou plusieurs processeurs qui supportent la partie logicielle, une surface de portes logiques (pour supporter la partie matérielle et les bus de communication), des blocs mémoires et des interfaces de communication.

Dans les paragraphes qui suivent, nous allons détailler les différentes étapes du flot de conception comme présentées dans la Figure 1.

3.1.1 Spécification haut niveau

La spécification d'un système fixe les fonctions principales exigées par l'utilisateur. En spécifiant le système, le concepteur doit tenir compte des performances techniques (paramètres architecturales) et économiques (coût de fabrication) du système sur puce. L'objectif de l'étape de spécification est de fixer un modèle fonctionnel et de le tester tout en vérifiant les contraintes d'implémentation de l'application. La création d'un modèle fonctionnel s'intéresse à la

structuration ou l'organisation du comportement du système au cours du temps. La spécification de l'application dépend des paramètres imposés par la plateforme d'implémentation.

Divers types de modèles de spécification fonctionnelle existent dans la littérature tels que Statechart, les réseaux de pétri et UML (Unified Modeling Language). Ces modèles de spécification, restent inadaptés pour exprimer les contraintes non fonctionnelles de l'application telles que les performances temporelles (durée, latence et débit) et les ressources d'implémentation nécessaires (énergie, mémoire et surface d'implémentation). La vérification de l'intégrité fonctionnelle de l'application nécessite l'utilisation d'un langage de description de haut niveau associé à un noyau de simulation comme SystemC [Sys].

3.1.2 Partitionnement logiciel / matériel

L'étape de partitionnement logiciel/matériel détermine les tâches qui vont être implémentées sur un ou plusieurs processeurs et les tâches de traitement effectuées par des accélérateurs matériels. Durant cette étape de partitionnement, le concepteur fixe également les interfaces entre la partie HW et SW [Ara05, Ana05].

L'étape de partitionnement automatique d'une spécification est un problème complexe (un problème NP-complet). Afin de décomposer ce problème, on peut subdiviser ce processus en trois parties principales :

- Une partie qui effectue l'allocation des différentes ressources matérielles et logicielles en fixant leurs types et nombres. Dans le cas d'un processus d'allocation statique, le concepteur dimensionne à un niveau d'abstraction très haut l'architecture globale de l'application embarquée. Le concepteur fixe ainsi la limite de performance du design qui dépend des composants déployés dans l'architecture.
- Une autre partie qui effectue le partitionnement spatial ou temporel en affectant les tâches qui constituent l'application sur la partie matérielle ou logicielle. Dans les architectures statiquement reconfigurables, cette partie se limite souvent à un problème de partitionnement spatial dans lequel on affecte les différentes tâches aux différents composants fonctionnels. Cependant, dans le cas des architectures reconfigurables dynamiquement, le partitionnement temporel est devenu une étape indispensable dans la conception du système.
- Une troisième partie qui effectue l'ordonnancement de l'exécution et de la reconfiguration des différentes tâches ainsi que la communication entre elles. A ce niveau, le concepteur doit explorer l'espace de solution afin de sélectionner une

architecture qui respecte les contraintes de traitement temps réel imposées.

3.1.3 Validation

Une fois qu'un partitionnement satisfaisant a été trouvé et que l'architecture a été définie, l'étape de synthèse post-exploratoire permet de générer le système. Il y a ici deux étapes, d'une part la synthèse matérielle dont le but est de générer soit un « netlist » pour la création d'un ASIC soit le code à télécharger « bitstream » dans un composant programmable de type FPGA, et d'autre part la synthèse logicielle (compilation) dont le but est de générer le code qui sera exécuté par un ou plusieurs microprocesseurs du système.

Dans cette partie, nous avons présenté les étapes essentielles du flot de conception mixte typique. Le but des approches de conception existantes est de proposer des architectures le mieux adaptées au traitement requis afin d'optimiser la surface des architectures et leur temps de développement. Toutefois, avec l'apparition des systèmes embarqués, avec des ressources d'énergie limitées, la consommation est devenue une contrainte prioritaire qui doit être prise en compte dans la conception des systèmes multimédia. Ce sera l'objet du paragraphe suivant.

3.2 Gestion de la consommation

La maîtrise de la consommation de puissance et d'énergie est un problème souvent rencontré dans le domaine des systèmes embarqués autonomes. Elle est devenue, ces dernières années, un facteur essentiel dans l'étape de conception. Ceci est dû au fait que les nouvelles applications deviennent de plus en plus complexes, et requièrent des architectures très complexes et par conséquent un nombre croissant de transistors sur la puce.

L'augmentation des puissances consommées rend l'amélioration de l'autonomie du système embarqué un objectif principal dès sa conception. Elle est aussi un facteur déterminant pour son succès commercial. Cet objectif peut être atteint par diverses techniques :

- Augmentation de la capacité de stockage d'énergie des piles : cet objectif est très difficile à atteindre et l'évolution dans ce domaine ne suit pas le progrès des besoins des systèmes actuels. Ceci a causé un « gap » entre l'évolution de la complexité des applications et l'évolution de la densité d'énergie des piles (exprimée en Wh/kg) comme le montre la Figure 2. En effet, la loi d'Eveready prévoit une lente évolution de la capacité des piles ne dépassant pas 40 %, évolution 4 fois moins élevée que celle de la capacité

d'intégration des circuits prédite par la loi de Moore [Tur03]. Cette évolution ne permet plus de suivre la complexité croissante des applications prédite par la loi de Shannon [Kar03].

- Diminution de la consommation du système embarqué. Diverses méthodes situées à différents niveaux d'abstraction peuvent être investiguées : au niveau algorithmique et au niveau architectural.

Comme il reste toujours difficile d'augmenter la capacité de stockage d'une batterie sans en augmenter le poids, le volume et le prix, il est donc nécessaire d'investir d'avantage dans la deuxième solution. Par conséquent, des méthodes de gestion d'énergie efficaces doivent être définies et incluses dans les différentes phases de conception des systèmes [Ben07].

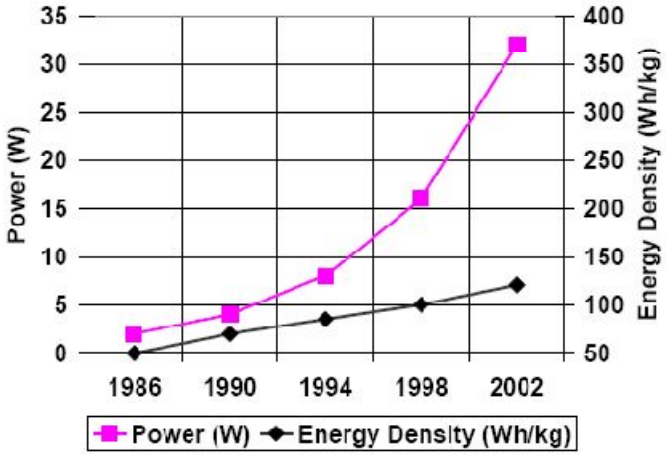

Figure 2: "Gap" entre l'évolution des batteries et l'évolution des semi-conducteurs [Kan02]

3.3 Techniques de réduction de la consommation

Cette partie présente quelques techniques utilisées pour concevoir un système embarqué à faible consommation. Ces techniques peuvent être réparties en trois catégories : des techniques matérielles, d'autres logicielles et celles qui combinent les deux techniques précédentes, logicielles et matérielles.

Diverses techniques ont été proposées pour mettre en place un système à faible consommation. La plus répandue est celle qui conçoit des composants spécifiques qui consomment le moins possible. La seconde méthode consiste à fournir la partie logicielle qui consomme la faible quantité d'énergie et ce à travers des méthodes d'optimisation du code de l'application à exécuter. La dernière méthode consiste à combiner les deux techniques

précédentes c'est-à-dire, utiliser des composants qui consomment le moins et adapter le code de l'application suivant l'architecture utilisée.

3.3.1 Technique matérielle

3.3.1.1 Au niveau composant

Actuellement, on assiste à une très forte évolution dans le domaine de fabrication des circuits électroniques. Ceci est dû à l'augmentation du taux d'intégration et de la fréquence de fonctionnement des circuits actuels. Ce qui implique une augmentation au niveau de la consommation du système.

Il est à noter que la consommation du système est la somme de deux types [Nab10] :

- la consommation statique qui est proportionnelle au nombre de transistors et au courant de fuite
- la consommation dynamique qui est proportionnelle à la fréquence de fonctionnement et la tension d'alimentation du système

Diverses techniques ont été proposées pour remédier à ce problème, telles que la diminution de la tension d'alimentation, l'activation séparée des blocs logiques et le contrôle du taux de basculement des bits.

- **Diminution de la tension d'alimentation**

La tension d'alimentation est un facteur qui a un impact très important sur la puissance dissipée par un circuit qui peut être calculé par la formule suivante:

$Pdyn= \alpha.C.f.V^2$

Avec :

C : la capacité équivalente

F : la fréquence de fonctionnement

V : la tension d'alimentation

Ces dernières années, la tension d'alimentation des circuits intégrés n'a cessé de diminuer. Au début, cette valeur était fixée à 5v ; alors que, actuellement, la plupart des circuits travaillent avec une valeur de 3.3v ou 1.1v. Il est à noter qu'il y a des composants qui utilisent 0.5v et ceci est dû non seulement aux progrès de la conception des circuits intégrés mais aussi à la maitrise des techniques de fabrication ; on parle maintenant de la technologie 45nm [Mat09].

- **Activation séparée de composants**

La deuxième technique proposée consiste à activer séparément les composants d'un circuit suivant les besoins de l'application [Koi06]. Cette technique n'est pas toujours réalisable et peut envisager des traitements supplémentaires pour l'activation des parties nécessaires au

bon moment. Cette technique a été utilisée pour diminuer la consommation de la mémoire cache. Elle consiste à diviser la mémoire en un ensemble de blocs qui peuvent être activés indépendamment les uns des autres.

- **Diminution du basculement des bits**

La troisième technique consiste à diminuer le nombre de basculements des bits entre 0 et 1, et ce, à travers la réutilisation des informations existantes sur le bus. Il est à noter que ce nombre intervient dans le facteur α déjà cité pour le calcul de la puissance dissipée par un circuit.

3.3.1.2 Au niveau système

La réduction de la consommation d'un système n'intervient pas seulement aux niveaux des composants mais aussi au niveau du choix de l'architecture du système complet [Jul04]. Parfois, là où la consommation du système coûte cher, on sera obligé à choisir des composants moins performants que d'autres et qui consomment bien entendu beaucoup moins d'énergie. Par exemple la plupart des systèmes actuels utilisent des supports de stockage de type mémoire flash bien que leur capacité de stockage soit beaucoup plus faible que celle d'un disque dur. Ce choix est notamment fait à cause de la faible consommation qui peut atteindre jusqu'à 90% de gain par rapport au disque dur [Lor98].

3.3.2 Technique logicielle

Cette technique consiste à modifier le code susceptible d'être exécuté sur le système dans le but de diminuer la consommation induite à son exécution. Cette technique s'avère facile, mais en pratique, c'est très difficile de mettre en place des outils qui permettent l'automatisation de cette technique car elle demande une connaissance très précise de l'application ainsi que les spécificités de l'architecture du système [Mat07].

Cependant, des travaux d'optimisation du code de l'application peuvent avoir lieu manuellement pour minimiser la consommation du système ; le programmeur peut utiliser des langages de bas niveau tel que l'assembleur pour faire des instructions spécifiques d'une manière précise qui consomme moins. Une deuxième technique a été proposée ; elle consiste à remplacer l'appel des fonctions par des fonctions en lignes.

3.3.3 Technique mixte

Cette technique est basée sur la collaboration entre les composants matériels et logiciels. Les mécanismes utilisés pour la réduction de la consommation proviennent aussi bien du matériel que du logiciel. Ce dernier prend en charge la prise de décision d'activation des mécanismes matériels nécessaires pour la réduction de la consommation [Chr11].

Ces techniques de réduction de la consommation sont très efficaces et permettent un gain très important au niveau d'énergie. Le problème avec ces techniques est que le concepteur se trouve parfois devant un compromis performance/consommation d'où la nécessité d'intégrer la gestion de la consommation dans le flot de conception afin de guider le concepteur dans son choix. Dans le paragraphe suivant nous présentons quelques approches classiques de codesign faible consommation.

3.4 Approches de codesign faible consommation

Les méthodes de réduction de puissance et d'énergie sont plus efficaces, plus qu'elles sont adressées le plus tôt possible dans le processus de conception (globalement au niveau système). Cependant, la majorité des travaux existants sur l'optimisation de puissance adresse séparément les parties matérielles, logicielles et de communications après avoir fixé l'architecture du système. Seules quelques approches de codesign tiennent compte de la gestion de la consommation à un niveau d'abstraction plus élevé. Ces approches commencent en général par une étape d'estimation de la consommation des parties du système (tâches, fonctions, communications etc.) [Mou03a] pour déterminer ensuite, et le plutôt possible, la consommation totale du système. Parmi ces méthodes, on trouve:

1. Dave et Al. Ils ont présenté l'environnement COSYN-LP [Dav97], qui est l'extension de l'environnement COSYN pour faire l'optimisation de la consommation au niveau système. Leur méthode inclut une première étape d'estimation de la consommation et des temps d'exécution des tâches du système. Ensuite, un algorithme de type heuristique fait le partitionnement/ordonnancement des tâches du système sur l'architecture cible multiprocesseur. Cette approche permet de réduire la consommation dans le système jusqu'à 25%.

2. Fornaciari et Al dans [For98] présentent des métriques de consommation efficaces pour guider le partitionnement Hw/Sw au niveau système. Les métriques d'évaluation de la consommation ont été définies pour explorer largement l'espace de solutions au niveau élevé d'abstraction.

3. Dans [Jal09] les auteurs s'intéressent à l'exploration d'architecture basse consommation. Pour ceci ils ont proposé un flot de conception qui dispose d'une librairie d'IP modélisés en consommation à l'aide de paramètres de haut niveau. Par ailleurs des modèles

de performances riches et une technique d'exploration basse consommation est proposée. Cette approche permet de considérer un certain nombre de paramètres algorithmiques et architecturaux sur la consommation. Un modèle complet est proposé afin de déduire les performances globales du système qui seront utilisées lors de l'exploration à travers une technique basée sur le recuit simulé.

3.5 Limitations des approches présentées

Le codesign faible consommation permet la création de systèmes à basse consommation en tenant compte des caractéristiques de l'application et de l'architecture cible. Cependant, ces approches ne tiennent pas compte d'un ensemble de paramètres « dynamiques » imprévisibles à l'avance et donc difficilement analysables hors ligne.

En effet, un certain nombre de paramètres peuvent modifier les performances d'un système multimédia embarqué pendant son exécution. Ils se situent à différents niveaux, certains étant aléatoires et non liés à une application cible :

- Le comportement de la batterie du système
- La variabilité des données et des applications
- Préférences de l'utilisateur
- Adéquation algorithme architecture
- Influence de l'environnement

3.5.1 Le comportement de la batterie du système

La plupart des systèmes multimédia mobiles opèrent avec des batteries. Leur évolution reste lente par rapport à la demande des nouvelles applications. Il est en effet toujours difficile d'augmenter la capacité de stockage d'une batterie sans en augmenter le poids, le volume et essentiellement le prix [Azz04]. Ceci a poussé les chercheurs à opter pour l'optimisation de l'utilisation de l'énergie de ces batteries puisque la durée de vie du système en dépend directement. Dans ce contexte, les travaux de [kan02] ont montré que l'augmentation de la durée de vie de la batterie du système n'est pas en liaison directe avec la réduction de la puissance moyenne consommée. En effet, le profil de courant instantané influe sur la capacité de la batterie. Ainsi l'exploration basse consommation est insuffisante pour augmenter la durée de vie du système. Il faudrait plutôt suivre en ligne l'évolution des ressources disponibles et agir en conséquence sur les éléments (architecture, application…) du système.

3.5.2 La variabilité des données et des applications

Le progrès des systèmes informatiques mobiles et embarqués (téléphones mobiles et les PDAs) impose l'utilisation de plusieurs applications multimédia complexes [Ram05]. Ces applications, traitent un nombre variable de données, et la première source de variabilité de la performance de tels systèmes. En effet, la diversité des applications exécutées sur la même plateforme produit une variabilité au niveau de la charge de travail du système complet. La caractérisation de l'évolution instantanée de la charge de travail des différentes applications est une étape nécessaire pour estimer les performances d'exécution du système implémenté sur la plateforme. Par ailleurs, la variabilité des performances de traitement associée à une application dépend largement de la nature du flux de données traitées [Rut02]. Une architecture embarquée qui implémente une application traitant des données multimédias, comme les applications de vision artificielle, d'imagerie 3D et de codage audio, représente un exemple significatif qui illustre cette variabilité temporelle. Dans le cas des applications multimédias, l'origine principale de cette variabilité de performance est l'instabilité au niveau des propriétés des données traitées (images ou séquence vidéo) [Var04].

3.5.3 Préférences de l'utilisateur

L'utilisateur, maitre de cet environnement, peut donner ses consignes au système, tels que la durée de vie souhaitée du système et le niveau de qualité de service acceptable pour le fonctionnement du système. Ces exigences peuvent être modifiées à n'importe quel moment. Par ailleurs, l'utilisateur peut privilégier l'exécution de certaines fonctionnalités de son système mobile au dépend d'autres qu'il peut désactiver ou dégrader leur qualité de service. Ceci engendre des changements de la consommation d'énergie et de l'allocation des charges de travail des différentes applications du système.

3.5.4 Choix architectural

Une architecture système typique qui traite des données multimédias comporte des accélérateurs matériels spécifiques, un ou plusieurs composants programmables (processeurs, DSP ou contrôleurs) et de la mémoire [Sof07]. Une telle architecture permet d'améliorer les performances du système en termes de temps d'exécution et par conséquent en termes de qualité de service. Mais cette amélioration est pénalisée par l'augmentation de la consommation du système. D'une part, puisque ces composants ont des consommations dites statiques càd puisqu'ils consomment de l'énergie même s'ils ne sont pas en cours d'utilisation. D'autre part, il est évident qu'une tâche implémentée en hardware s'exécute plus rapidement que celle en

software. Mais il faut tenir en compte qu'ils consomment aussi plus que la tâche software [Ben07]. Par conséquent, le concepteur se trouve devant plusieurs choix architecturaux qui peuvent être utiles pour le système dans un contexte de fonctionnement, mais lorsque les exigences du système changent au cours du fonctionnement l'utilisation de ces composants devient inutile compte tenu de leur influence sur la consommation du système.

3.5.5 Influence de l'environnement

Le fonctionnement d'un système embarqué peut changer d'un emplacement à un autre si les caractéristiques de l'environnement change [Fre02]. Par exemple l'utilisation d'un caméscope dans un emplacement lumineux consomme beaucoup plus moins d'énergie que son utilisation dans un milieu sombre car il nécessite l'utilisation d'une source d'éclairage. Même chose pour les applications exécutées pouvant dépendre des contraintes imposées par l'environnement. Par exemple l'utilisation d'un téléphone dans un endroit bruité peut imposer l'exécution des algorithmes pour l'élimination des bruits de fonds.

Afin de répondre à ces nouvelles exigences dictées par la prolifération de ces nouveaux systèmes embarqués, de nouvelles méthodes de conception doivent être mises en place. Ces méthodes doivent d'une part tenir compte des propriétés des applications multimédia pour générer l'architecture adéquate. Ainsi l'architecture générée est optimisée pour l'application ciblée ce qui permet de réduire les coûts et d'augmenter les performances. D'autre part, ces nouvelles méthodes de conception doivent permettre d'obtenir des systèmes qui peuvent s'adapter efficacement à leur environnement externe (contraintes de fonctionnement, énergie disponible…).

4 Méthodologies d'adaptation

L'adaptation est une caractéristique nécessaire aux systèmes que nous venons de présenter, afin de maintenir leur fonctionnalité face à des modifications de leur environnement. Laddaga présente la notion de logiciel auto-adaptatif, c'est-à-dire « capable de surveiller, comprendre et modifier sa fonction à l'exécution » [Lad01]. L'objectif est de permettre de réagir au dynamisme de l'environnement d'exécution afin de fournir une nouvelle fonctionnalité ou d'améliorer la qualité de celles déjà rendues. Bien entendu, elle nécessite de la part du système une connaissance de cet environnement qui l'entoure.

La tendance actuelle au niveau de la conception des systèmes sur puce est de générer des architectures réactives qui s'adaptent en lignes pour mieux satisfaire les besoins de l'application. Ces besoins s'expriment en termes de performance, de qualité de service ou de consommation d'énergie. Ils sont fixés, d'une part, suivant des paramètres fonctionnels telle que la charge de travail affecté à chaque application pour le respect de la contrainte temps réel et d'autre part, ils sont liés à des paramètres non fonctionnels telles que l'énergie disponible dans la batterie et les préférences de l'utilisateur (QoS et DDV). Le besoin d'adapter le comportement d'un système est une conséquence de la variabilité des ressources ou des contraintes d'un système sur puce fluctuant dans un environnement variable.

Récemment, il y a eu de nombreuses contributions de recherches sur l'auto adaptation pour les systèmes autonomes dans plusieurs couches : couche matérielle, couche du système d'exploitation (OS), et couche applicative. Dans cette section, nous allons présenter les différents niveaux d'adaptation ainsi que les techniques utilisées ; ensuite nous citons quelques approches d'adaptation ; nous présentons également leurs apports et leurs limitations.

4.1 Adaptation au niveau matériel

Diverses techniques ont été proposées dans la couche matérielle pour l'adapter suivant les exigences du système. Parmi ces techniques on peut citer en premier lieu, la graduation dynamique de tension (DVS) qui est employée pour ajuster la vitesse et la puissance du CPU. En second lieu, nous citons la technique de gestion de la consommation dynamiquement « DPM » cette technique se base sur l'arrêt des composants inutilisables dans le système. En troisième lieu, pour les plateformes reconfigurables, le changement de l'architecture du système est fait suivant les besoins de l'application et les contraintes du système

4.1.1 Technique DVS

Beaucoup de méthodes d'adaptation [Luo02, Man03, Shi04, Mar05,Yut11, Muh11] reposent sur le changement dynamique de la tension d'alimentation « Vdd » et de la fréquence de fonctionnement « F» pendant le fonctionnement du système. Ces méthodes sont motivées par l'apparition de circuits électroniques à tension d'alimentation et fréquences variables [Oku01]. On peut citer par exemple la technologie PowerNow ! d'AMD et les technologies SpeedStep et XScale d'Intel. L'ordonnancement temps réel dans ce cas consiste non seulement à déterminer l'ordre d'exécution des tâches mais également à fixer la fréquence de fonctionnement ainsi que la tension d'alimentation. Beaucoup de techniques qui sont développées pour des tâches périodiques ou apériodiques reposent sur deux approches

d'ordonnancement : un ordonnancement dynamique (on-line) [Lee00, Qua01] ou un ordonnancement statique (off-line) [Bam01, Gru01, Shi04]. L'approche dynamique recalcule à chaque fois les priorités des tâches et la tension d'alimentation pendant l'exécution du système. Pour l'approche statique, l'ordonnancement des tâches et le voltage sont calculés avant que le système n'entre en fonctionnement pour éviter l'overhead dû au calcul dynamique. Dans [Shi01] deux algorithmes sont présentés : l'un statique, basé sur un ordonnancement RM avec un gain en consommation d'énergie de 12%, et l'autre dynamique, basé sur un ordonnancement EDF avec un gain en énergie de 32%. Dans [Ouh03] les auteurs proposent une méthode de DVS dans le cadre de codesign logiciel/matériel.

4.1.2 Gestion dynamique de la consommation « DPM »

La gestion dynamique de la consommation DPM (Dynamique Power Management) est une approche efficace pour la gestion de la consommation sans dégradation des performances du système. Elle consiste à arrêter des parties du système pendant qu'elles sont inoccupées (idle) [Hua06, Xin07, Muh10]. Les algorithmes DPM observent l'arrivée des événements dans le système et prévoient les périodes d'inoccupation qui peuvent être déterminées par plusieurs méthodes. Cette technique est omniprésente dans les ordinateurs portables et les PDA : elle consiste à arrêter les composants après un temps fixe d'inactivité ; par exemple l'écran et le disque dur.

Dans [Swa01] les auteurs présentent leur algorithme en lignes de DPM, qui s'appelle LEDS (Low Energy Device Scheduler), qui fait l'ordonnancement à faible consommation pour les périphériques d'entrée/sortie des systèmes temps réel strictes. Il prend en entrée un ordonnancement prédéterminé des tâches et une liste d'usage des périphériques d'E/S pour chaque tâche et il produit une séquence d'états de type actif/inactif pour chaque périphérique d'E/S. Il garantit la non violation des contraintes temps réel et la réduction au minimum de l'énergie consommée par les unités d'E/S utilisées par l'ensemble des tâches. Il présente aussi un exemple où la consommation d'énergie est réduite de 50% avec un ordonnancement EDF des tâches.

4.2 Adaptation au niveau système d'exploitation

Vu la complexité des systèmes actuels, la complexité et la diversité des applications et la présence de fortes contraintes, l'utilisation des systèmes d'exploitation temps réel dans les systèmes sur puce est devenue indispensable.

Un système d'exploitation offre divers types de services de communication et de synchronisation entre les tâches du système. Il propose aussi un ensemble de routines de gestion des ressources matérielles. Mais le service le plus important d'un système d'exploitation est l'ordonnancement et l'affectation de la charge de travail aux différents processus présents dans le système afin qu'il soit ordonnançable.

Puisqu'on travaille dans un contexte variable au niveau de l'application exécutée et de données traitées les systèmes d'exploitation actuels doivent prendre en compte ces variations en modifiant la charge de travail CPU affectée à chaque tâche.

Beaucoup de travaux ont été effectués dans le système d'exploitation et la couche middleware pour fournir l'allocation prévisible de l'unité centrale de traitement et l'adaptation des services [Bav00, Ban02, Kan11]. Dans [Fli01, Cor01, Wan03]. Les gestionnaires de ressources d'unité centrale de traitement, fournissent des garanties de performances en temps réel. D'autres travaux se sont focalisés sur le changement de la politique d'ordonnancement (ordonnanceur basse consommation) parmi ces travaux on peut citer [Bav00, Ban02]. Les auteurs de [Wan03, Fli01, Bra02] utilisent une couche middleware qui est une couche intermédiaire située entre la couche applicative et le système d'exploitation pour faciliter aux applications l'adaptation de leur QoS.

4.3 Adaptation au niveau applications

Beaucoup de projets préconisent l'économie d'énergie dans la couche application. Par exemple, les auteurs de [Fli99] explorent comment adapter le comportement d'application suivant l'énergie. Mesarina et autres [Mes02] discutent comment réduire l'énergie dans le décodage de MPEG. Dans [Van02, Pha04] deux approches sont proposées pour la dégradation de la qualité d'un objet 3D pour satisfaire des contraintes de ressources et d'environnement. Notre modèle prend en compte la modification des paramètres de l'application pour économiser de l'énergie et par conséquent augmenter la durée de vie du système.

Afin de bénéficier des avantages des méthodes citées ci-dessus et d'en réduire les limites, des travaux ont été faits en se basant sur des méthodes qui travaillent sur les différentes couches du système tel que le modèle GRACE [Wan03]. Ces approches, cependant, supposent que la couche de matériel soit statique. La seule modification prise en compte est au niveau de la fréquence et de la tension du CPU. A la suite de cette section on présentera le principe de fonctionnement de quelques méthodologies d'adaptation.

4.4 Approches d'adaptation existantes

Différentes approches d'adaptation ont été proposées dans la littérature. Elles exploitent un ou plusieurs niveaux d'adaptation et gèrent différents types de contraintes comme l'énergie consommée, la qualité perçue (QoS), le temps réel etc.

Dans la suite de ce chapitre nous présenterons quelques approches existantes. Nous traiterons également l'apport et les limites de chacune d'elles.

4.4.1 Gestion de la QoS pour assurer une interaction de trames

L'auteur de [Pha04] présente un travail de gestion de QoS pour garantir une interaction de trame par dégradation de la QoS de l'objet 3D sous des contraintes de ressources et d'environnement.

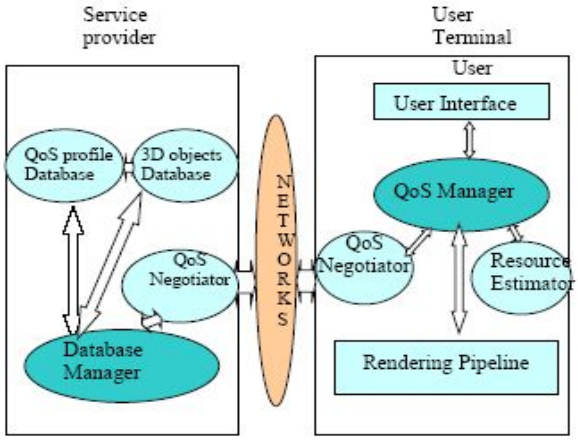

Figure 3: Schéma du système d'adaptation [Pha04]

Le but de ce travail Figure 3 est de mettre en place une approche qui permet de faire une interaction entre un émetteur et un récepteur d'une scène formée par plusieurs objets 3D. Afin de réduire la quantité d'informations échangées entre les deux, le traitement des objets 3D est

fait chez le récepteur. La contrainte à prendre en considération est que le temps de traitement de la scène chez le récepteur ne doit pas dépasser le temps d'envoi d'une trame sur le réseau. Puisque dans le cas contraire le récepteur n'arrive pas à afficher toutes les trames reçues.

Comme solution il a proposé de modifier la qualité des objets 3D suivant leur profondeur dans l'écran dans le but de réduire le temps de traitement chez le récepteur.

En premier lieu une étude sur les différents facteurs qui influent sur la qualité des objets 3D à été menée. En second lieu ils ont présenté leur problème comme étant un problème d'optimisation qui cherche à choisir les meilleurs paramètres applicatifs pour chaque objet permettant de fournir la meilleure QoS possible sous contrainte que le temps de traitement chez le récepteur ne dépasse pas le temps d'envoi d'une trame sur le réseau.

Comme solution ils ont présenté un algorithme qui définit pour chaque objet une liste de (bénéfice, coût) puis commence des itérations pour trouver une solution qui présente un bénéfice maximum pour tous les objets tout en respectant la contrainte temporelle. La complexité de l'algorithme proposé dans le pire cas est O (N.L.logL.) où L est le nombre de niveaux de qualité de l'objet et N représente le nombre d'objets visibles pour un point de vision considéré.

Cette approche s'avère être utile dans des cas bien spécifiques mais elle ne peut pas être appliquée dans plusieurs systèmes où le traitement se fait aussi bien sur l'émetteur que sur le récepteur. De plus on constate bien qu'elle ne tient en compte que d'une seule contrainte qui est le temps d'envoi d'une trame sur le réseau et abandonne les autres, surtout la consommation et les préférences de l'utilisateur. En plus l'algorithme proposé nécessite un traitement assez complexe par conséquent il consomme beaucoup de ressources du système; ce qui influe sur les performances du système.

4.4.2 Adaptation à base d'affectation de budget de ressource

L'auteur de [Van02] présente une approche de gestion de ressources du système entre les différentes applications présentes. Le principe se base sur l'affectation d'un budget de chaque type de ressource à chaque application. Les applications ne doivent pas dépasser leur budget. Dans le cas contraire cette approche peut modifier les paramètres applicatifs en dégradant la QoS de l'application pour respecter le budget, la prochaine itération.

Cette approche peut être utilisée pour l'adaptation de la qualité dans le cas où le système exécute plusieurs applications simultanément, elle se compose essentiellement d'un

Ressource Manager (RM), plusieurs « Ressource Consuming Entities » (RCE), plusieurs « Domain Quality Manager » (DQM) et d'un « Global Quality Manager » (GQM) Figure 4.

La gestion de ressource est basée sur des budgets de ressource fournis par le RCE et alloués par le gestionnaire global de qualité (GQM). Dans ce travail, ils considèrent une ressource, le CPU, et deux RCE, 3D et vidéo. Les budgets sont exprimés en pourcentage de temps de traitement CPU avec une granularité donnée, par exemple 70% du temps de processeur toute les 20 ms. Le dépassement du budget pour une période donnée peut causer un retard de RCE jusqu'à la période du budget suivante. Le contrôleur de RCE s'assure que le RCE s'exécute acceptablement dans les limitations de son budget.

Le gestionnaire de qualité détermine les paramètres de qualité préférée et le budget du RCE. Le Global Quality Manager (GQM) est indépendant du domaine (3D, vidéo) et prend des décisions dans de multiples domaines. Le GQM coopère avec un ou plusieurs Domaine Quality Manager (DQM) qui a la connaissance de détail de domaine. Pour un système donné, il peut y avoir plusieurs DQM, un par domaine sémantique.

Puisque 3D et vidéo sont deux domaines sémantiques séparés, les systèmes avec 3D et vidéo exigeront un 3D Quality Manager (3D_QM) et un Vidéo Quality Manager (VQM)

En cas de surcharge, si un contrôleur de RCE ne peut pas maintenir un niveau de qualité acceptable dans les limitations de son budget, les gestionnaires de qualité doivent renégocier les budgets de tous les RCE.

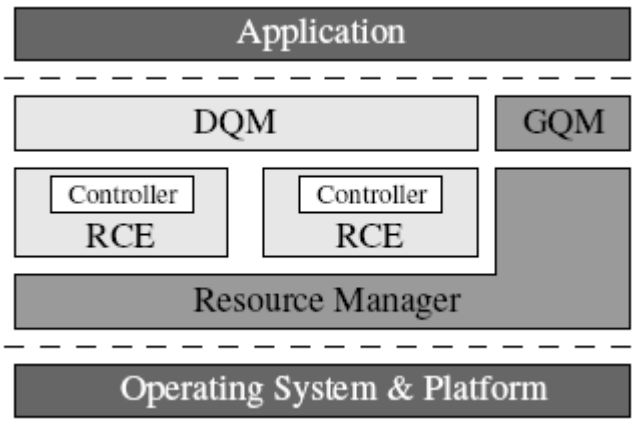

Figure 4: Structure de la couche d'adaptation [Van02]

Cette méthode traite le cas de plusieurs applications et peut être utile dans le cas où on connait auparavant les besoins en termes de ressources pour chaque type d'application. Or comme on

l'a déjà mentionné au début de ce chapitre les besoins des systèmes actuels différent d'une situation à une autre.

Par ailleurs cette approche ne tient pas compte de la possibilité d'adaptation au niveau de la couche hardware.

4.4.3 Gestion de la QoS basée sur le partitionnement HW/SW

L'auteur de [Pha03] présente un travail de gestion de la qualité de service pour des plateformes reconfigurables à base de FPGA. Le changement de l'architecture est fait suivant le type de l'application exécutée (MPEG ou 3D) et en fonction des ressources disponibles (capacité de l'FPGA, temps d'exécution). Pour ce faire, il propose d'ajouter une couche Middleware de gestion de la qualité de service dont la fonction principale est de décider quelles sont les tâches qui vont être implémentées en hardware et celles en software de telle manière que l'utilisateur reçoive la meilleure qualité possible.

L'objectif de la gestion de la QoS basée sur le partitionnement HW/SW est de trouver une solution de partitionnement qui permettrait de maximiser la qualité de l'application sous des contraintes de ressource.

Cette méthode est utilisée pour les plateformes reconfigurables, mais, elle aussi, ne prend pas en charge la consommation qui est une contrainte très importante surtout pour les systèmes embarqués. D'autre part, les algorithmes de partitionnement présentés ont une grande complexité ; ce qui influe sur les performances du système.

Cette approche peut être étendue afin qu'elle intervienne aussi sur la couche applicative et système d'exploitation pour en bénéficier de l'apport de chacune d'entre elles.

4.4.4 Approche d'adaptation multicouche « GRACE »

L'approche GRACE présentée dans [Yua06], intervient dans les trois couches du système pour assurer à l'utilisateur la meilleure QoS possible tout en respectant la contrainte temps réel et la durée de vie souhaitée du système. Au niveau architectural ils ont utilisé la technique de variation de la fréquence de fonctionnement du système. Au niveau système d'exploitation elle intervient dans la charge de travail affectée à chaque tâche et au niveau applicatif ils ont proposé de modifier les paramètres applicatifs de l'application pour réduire les ressources consommées.

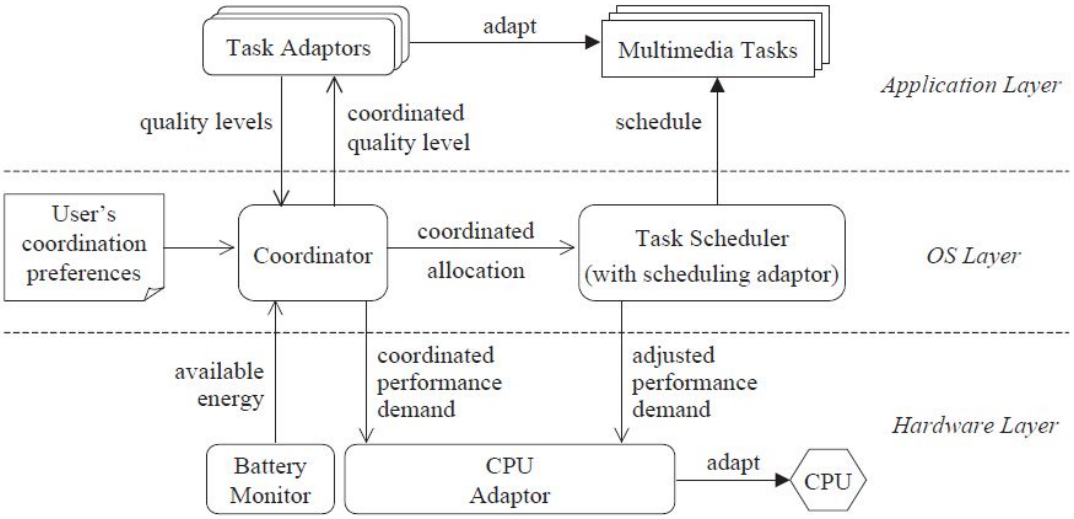

Figure 5: Schéma de l'approche Grace [Wan03]

Le modèle GRACE présentée par la Figure 5 permet de faire une adaptation sur trois couches. En plus, il prend en considération la gestion de la consommation du système. Mais ce modèle présente quelques inconvénients puisque la plupart des processeurs embarqués ne permettent pas de changer dynamiquement la fréquence de fonctionnement. De plus cette approche ne tient pas en considération la possibilité de modifier l'architecture du système suivant les besoin des applications exécutées.

4.4.5 Approche d'adaptation du Lab-STICC

Une approche d'auto-adaptation multicouche a été mise en place [Jph11]. Cette approche se base sur l'utilisation d'une base de configurations pré-caractérisées dans une étape qui se fait hors ligne. Au cours du fonctionnement, l'approche décide les prochaines configurations du système. La décision se base sur un algorithme de vote multidimensionnel mis à jour par des mesures et des estimations. Son principe de fonctionnement est illustré par la Figure 6 :

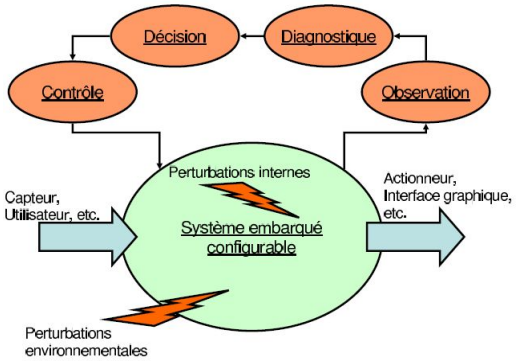

Figure 6: Principe de fonctionnement de l'approche du Lab-STICC

Cette approche est focalisée essentiellement sur la migration des tâches software en hardware et la prise en compte de ce changement dans le système d'exploitation. Cependant, elle ne contient pas les mécanismes nécessaires pour le contrôle du respect de la contrainte temps réel.

4.4.6 Approche d'adaptation multi contrainte « class »

Dans le cadre de la thèse de Nader BEN AMOR, une approche d'adaptation à contraintes multiples « class » (Cross Layer Adaptation Simulator System) Figure 7 a été établie [Ben07]. Il a proposé une méthode reposant sur la régulation du compromis durée de vie / Temps réel / Qualité de service. Cette méthode suppose d'une part l'existence de divers modes de fonctionnements du système et d'autre part que celui-ci est capable de passer d'un mode à un autre suivant l'évolution des paramètres durée de vie, temps d'exécution et QoS.

Cette méthode a été validée à travers un environnement de prototypage virtuel. Cet environnement émule le fonctionnement d'un système embarqué réel exécutant une application test. Le simulateur utilise plusieurs modules.

Le premier module indique le scénario que le simulateur doit exécuter.

Le second module est un modèle de batterie qui permet de déterminer la durée de vie du système connaissant la puissance que le système consomme. Une procédure de suivi de l'évolution des ressources d'énergie du système est utilisée. La fréquence utilisée pour le suivi de l'application est variable. Cela permet d'adapter d'une part le compromis entre le coût d'adaptation et le gain d'adaptation, et d'autre part, de suivre la vitesse d'évolution de l'utilisation des ressources du système.

Le module suivant est un modèle de consommation du système. Il permet, en connaissant le scénario, de prévoir la consommation du système. Ce modèle tient compte aussi de l'effet de la variation des données sur la consommation du système.

Le dernier module est la procédure de choix des configurations. Ce choix suit la priorité des trois contraintes. L'algorithme de choix assure la contrainte prioritaire tout en faisant du mieux possible pour les deux autres.

L'environnement utilise aussi une base de configurations pré caractérisées et mise à jour au cours du fonctionnement du système. Cet environnement a été développé pour la validation de la méthode d'adaptation générale ainsi que pour le test de différents algorithmes et techniques d'adaptation.

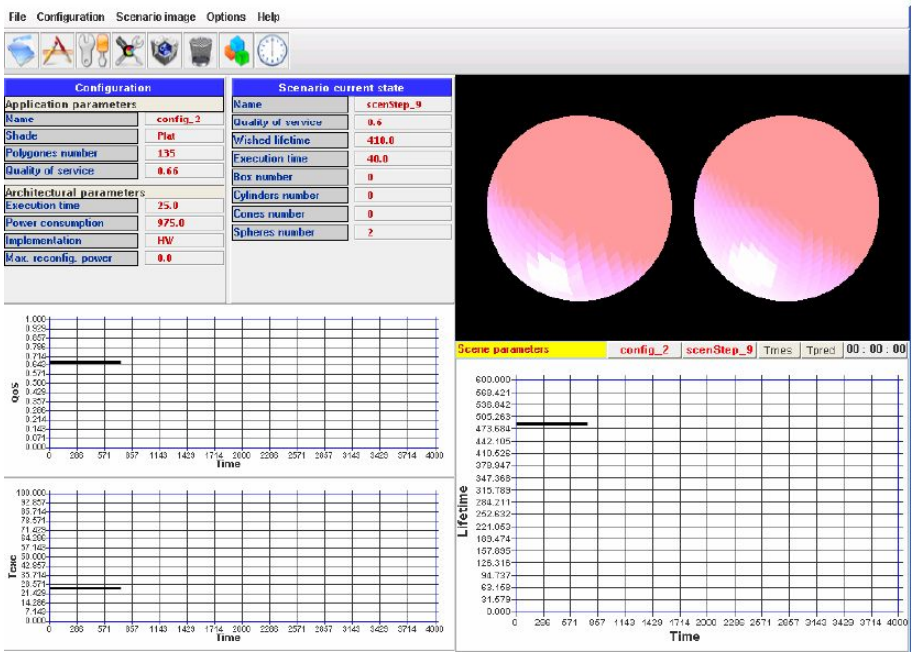

Figure 7: Interface du simulateur « class »

Cette approche présente un nouvel apport. Elle consiste à utiliser une base de configurations et à ce que le système puisse passer d'un mode à un autre au cours de son fonctionnement. En revanche, elle présente quelques limites puisqu'elle est validée à travers un démonstrateur et non pas un système réel d'une part. D'autre part, elle ne prend pas en considération l'utilisation d'un RTOS et les contraintes temps réel.

4.5 Discussion

Suite à l'étude faite sur les approches d'adaptation existantes on a remarqué que ces méthodes souffrent de quelques « handicaps ». Afin de surmonter quelques limites nous proposons dans le cadre de notre travail une approche d'adaptation multicouche qui permet de maximiser la QoS fournie pour l'utilisateur tout en respectant les contraintes du système (Ddv/Texe/QoS). Elle intervient dans la couche applicative en agissant sur les paramètres applicatifs pour modifier les ressources utilisées. Dans la couche système d'exploitation elle utilise la technique d'affectation des charges de travail pour chaque application et pour la couche hardware elle se base sur la modification de l'architecture du système en cours de fonctionnement.

5 Conclusion

La consommation est devenue un facteur important et limitatif dans la conception des systèmes embarqués par conséquent le flot de conception traditionnelle ne répond plus à leurs exigences. Pour surmonter ce problème, différentes méthodes de réduction de la consommation ont vu le

jour. Ces Méthodes sont d'un apport inestimable dans la conception des systèmes sur puce mais elles restent limitées puisque les systèmes embarqués ne cessent d'évoluer ; ce qui est traduit par l'apparition de nouvelles contraintes qui sont imprévisibles et de natures fluctuantes. De nouvelles approches d'adaptation ont été proposées pour répondre à ces nouvelles exigences. Dans la dernière partie de ce chapitre nous avons présenté quelques approches d'adaptation. Nous avons également présenté leurs apports et leurs limites.

Dans ce contexte les travaux menés dans cette thèse portent essentiellement sur la proposition et la mise en place d'une approche d'adaptation pour les systèmes embarqués temps réel. Le chapitre suivant sera consacré à la présentation de l'approche d'adaptation multicouche proposée. Nous présenterons également les différentes étapes qui ont conduit à sa mise en place.

CHAPITRE 3 : Approche d'adaptation multicouche

1 Introduction

Avec l'évolution des systèmes embarqués actuels, différents facteurs interviennent dans leur conception. Ces facteurs peuvent être de deux types prédictibles ou aléatoires au cours du temps. Nous avons montré dans le chapitre précédent l'importance d'avoir un système performant, flexible et adaptatif qui puisse gérer ses ressources selon des contraintes externes reliées à l'énergie disponible, la bande passante, les choix de l'utilisateur, etc. Dans ce contexte, nous avons présenté différentes méthodologies d'adaptation qui ont été mises en place pour permettre au système d'assurer un fonctionnement satisfaisant en milieu perturbateur et en présence de ressources d'énergie et de calcul réduites. Afin de combler les limites des approches existantes dans ce chapitre nous proposons une approche d'adaptation multicouche ainsi que les étapes qui ont conduit à sa mise en place. Cette approche intervient dans les trois couches du système (application, système d'exploitation et architecture).

Le principe du système d'adaptation proposé est illustré par la Figure 8. Il comporte :

- Une activité d'observation qui permet de suivre l'évolution et le respect des différents paramètres Texe, QoS et Ddv.
- Une activité d'adaptation qui permet de choisir une configuration pour le système afin qu'il puisse satisfaire les différentes contraintes de fonctionnement.
- La mise en place de la base des configurations pré-caractérisées.
- Nous supposons que le système possède divers modes de fonctionnement et qu'il est capable de passer d'un mode à un autre selon les consignes du système d'adaptation au cours de son fonctionnement. Une configuration (mode fonctionnement) du système représente une version de l'application multimédia et un type d'implémentation HW/SW. L'activité d'adaptation utilise la base de configurations mise en place et caractérisée hors ligne pour choisir une configuration pour le système qui fournit la meilleure QoS possible tout en respectant les contraintes du système.

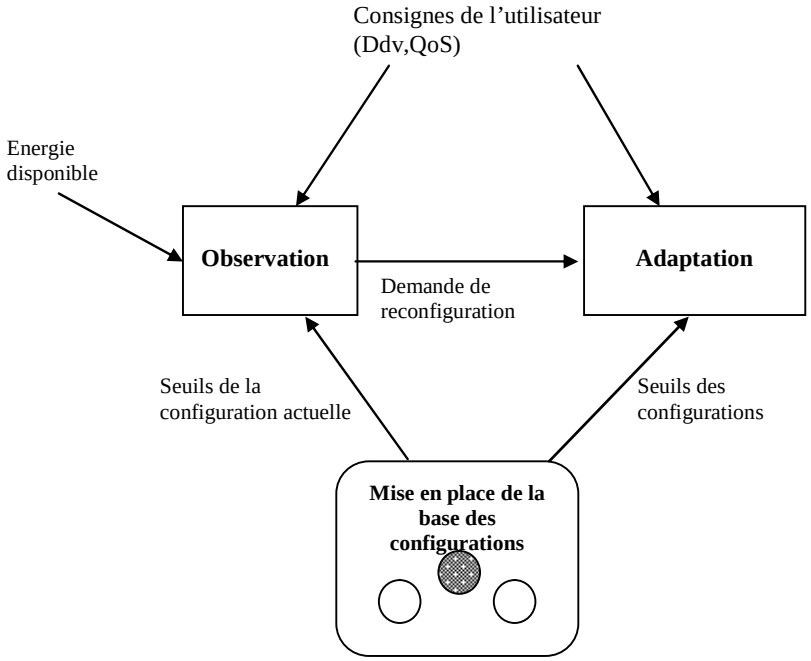

Figure 8: Schéma du système d'adaptation

Dans la suite de ce chapitre, nous détaillerons chacune de ces trois tâches. Dans la section (2), nous présenterons la tâche d'observation. Dans la section (3) nous présenterons la tâche d'adaptation développée, ses composants ainsi que leur principe de fonctionnement. Nous clôturerons ce chapitre par la présentation et la mise en place de la base des configurations.

2 Activité d'observation

L'activité d'observation permet, d'informer la tâche d'adaptation globale de tout type de changement qui apparait dans le système tel que l'apparition ou la disparition d'une tâche ou bien le changement des préférences de l'utilisateur.

Par ailleurs, l'activité d'observation, doit suivre en lignes l'évolution et le respect des trois paramètres Ddv, QoS est le respect des contraintes temps réel (noté Texe). Au cas où elle trouve des anomalies elle devra activer la tâche d'adaptation adéquate (global manager « GM » ou local manager « LM ») pour remédier au problème.

Le suivi de ces paramètres en ligne nécessite d'ajouter au système les routines logicielles et les structures matérielles adéquates. La Ddv d'un système dépend de la quantité d'énergie résidante dans sa source d'alimentation.

Le suivi de la Ddv d'une batterie nécessite l'utilisation d'un estimateur de l'état de charge de la batterie (gas gauge). Connaissant la puissance en cours de consommation et les paramètres de notre batterie, nous pouvons déterminer la durée de vie de notre système.

Le suivi de la contrainte temps réel se fait au niveau du système d'exploitation à l'aide de la technique du watchdog qui permet de détecter tout dépassement d'échéance.

Le suivi de la QoS se fait également au niveau du système d'exploitation. Il est déterminé à l'aide du modèle de la QoS.

2.1 Paramètres de l'activité d'observation

L'observation est une activité périodique. Nous notons dans la suite du document « Pobs » sa période. Une valeur faible de « Pobs » permet un suivi fin des différents paramètres à contrôler ce qui permet au système de réagir rapidement aux changements de consignes (ou de données à traiter). Cette rapidité de réaction est obtenue au dépend d'une augmentation de la consommation de l'activité d'observation due à une sollicitation plus importante de la jauge batterie. Pour remédier à ce problème nous avons proposé de faire varier la valeur de Pobs suivant la stabilité du système. Nous avons défini pour ceci deux valeurs limites pour « Pobs » :

- Une borne inférieure à Pobs notée Pobs_lim_inf : ce paramètre est introduit afin de limiter les coûts associés à l'activité d'observation (due à une valeur trop faible de Pobs). Elle est déterminée après la construction des différentes configurations du système.
- Une borne supérieure de Pobs notée Pobs_lim_sup, elle est spécifiée par l'utilisateur. Cette borne garantit que Pobs ne devienne pas trop importante ce qui empêcherait le système d'adaptation de suivre l'évolution des paramètres Ddv et QoS.

Durant le fonctionnement du système embarqué, Pobs doit satisfaire la condition suivante :

Pobs_lim_inf <Pobs< Pobs_lim_sup

Au cours du fonctionnement du système, la valeur de Pobs est ajustée selon la stabilité du système. Nous détaillons dans la suite l'ajustement dynamique de Pobs.

2.2 Ajustement dynamique de Pobs

Le principe de la mise à jour de Pobs se base sur le principe suivant :

- Pobs commence toujours à partir de la valeur initiale Pobs_lim_inf

- A chaque période de l'activité d'observation Pobs si la quantité d'énergie qui reste dans la batterie assure la durée de vie souhaitée par l'utilisateur et que la période Pobs est inférieure Pobs_lim_sup. la valeur de Pobs sera ajustée suivant la formule de l'équation E1. Dans le cas contraire (c'est-à-dire à chaque fois que la tâche d'adaptation globale est activée) la période Pobs est remise à sa valeur initiale E2.

$Pobs=Pobs+a*Pobs$ avec $0<a<1$ E1

$Pobs=Pobs\_lim\_inf.$ E2

3 Activité d'adaptation

Dans cette section, nous étalons l'intervention du modèle d'adaptation dans les couches hardware, OS et application. Nous montrons également l'interaction entre les différentes couches afin de bénéficier d'une meilleure qualité de service tout en respectant les contraintes. Notre technique d'adaptation peut être appliquée à trois niveaux : niveau architectural (hardware), niveau OS et niveau application.

3.1 Modèle d'adaptation niveau application

Nous considérons des applications multimédia telles que la synthèse d'images 3D le codage/décodage vidéo qui sont exécutés pour une longue durée et qui consomment un temps CPU élevé. Chaque tâche consomme un temps CPU et fournit un résultat de sortie. L'adaptation au niveau applicatif est faite en changeant les paramètres et le type de traitement pour fournir en sortie une qualité de service proportionnelle aux ressources consommées. Plus on accroit la consommation des ressources plus la qualité s'améliore. Par exemple, pour l'application de synthèse d'images 3D, on peut changer le nombre de polygones représentant l'objet. En effet, l'augmentation du nombre de polygones entraîne systématiquement une amélioration de QoS et vice versa. La modification du type d'algorithme d'ombrage utilisé (gouraud, plat, phong) modifie également la qualité observée puisque le modèle Gouraud par exemple permet de cacher l'apparence des polygones contrairement au modèle plat. Cette caractéristique (existante dans plusieurs autres applications multimédia comme MPEG, codage audio, etc…) peut être exploitée pour réduire la QoS au profit d'une réduction de la consommation de ressources (si par exemple les ressources d'énergie atteignent un niveau bas).

Le changement de l'un de ces paramètres entraîne une intervention au niveau de la couche système d'exploitation et parfois elle peut imposer une modification au niveau de l'architecture du système (si on change le type de traitement par exemple).

3.2 Adaptation niveau architectural

Une application peut avoir différentes implémentations. Une implémentation logicielle pure correspond à l'exécution de toutes les fonctions de l'application par le microprocesseur. Une implémentation mixte correspond à l'utilisation de composants matériels dédiés qui vont exécuter les tâches de l'application les plus complexes à la place du microprocesseur. Une implémentation mixte est généralement plus performante qu'une implémentation logicielle pure mais plus gourmande en ressources d'énergie. Comme les applications multimédia ont des besoins variables en terme capacité de calcul (du à leur variabilité), l'utilisation d'une seule implémentation peut causer des performances trop dégradées (cas où on utilise uniquement une implémentation logicielle) ou être trop gourmande (cas où on utilise uniquement une implémentation matérielle).

Afin de remédier à ce problème de choix de l'architecture, notre approche vise la possibilité de la modification de l'architecture du système au cours de son fonctionnement. Bien entendu, le changement du type de l'architecture hardware doit être signalé à la couche application pour qu'elle en tienne compte dans le code des applications exécutées. De même, la couche système d'exploitation doit être informée de ce changement pour qu'elle recalcule la nouvelle charge de travail affectée à chaque tâche.

Cette technique est plus performante que le changement dynamique de la tension d'alimentation pour diverses raisons. D'une part, cette technique peut être appliquée à tout type de processeur embarqué puisqu'elle n'exige pas qu'il supporte le changement dynamique de la tension d'alimentation. D'autre part, la technique DVS est de moins en moins utile compte tenu de la baisse des tensions d'alimentation des circuits actuels.

3.3 Modèle d'adaptation niveau système d'exploitation

Actuellement, les systèmes sur puce peuvent exécuter différentes applications simultanément. Afin de respecter la contrainte temps réel, il faut que toutes les tâches du système s'exécutent à leurs échéances. Le dépassement de l'échéance par l'une des tâches peut entraîner un retard de toutes les tâches ce qui perturbe le fonctionnement du système.

Nous travaillons avec des applications multimédia nous parlons donc de temps réel mou, par concequent les dépassements d'échéances sont acceptables mais dans une certaine limite.

Puisque le système exécute plusieurs applications à la fois, et que chaque application possède différents modes de fonctionnement (temps d'exécution différents) nous avons adopté la

notion d'affectation de budget de charge de travail (CPU_workload) à chaque tâche. En fait, l'exécution d'une tâche nécessite un certain nombre de cycles CPU (charge de travail). Afin de respecter la contrainte temps réel, nous allons allouer pour chaque tâche un nombre de cycles CPU suivant le nombre de tâches présentées sur le système. Lors de son exécution une tâche ne doit pas dépasser son budget pour terminer son traitement.

Cependant, une même tâche peut consommer différents nombres de cycles CPU selon différents paramètres (nombre et type de données, type de l'implémentation de la tâche HW/SW, …). C'est pourquoi il est nécessaire d'allouer à chaque fois à une tâche un nombre de cycles bien déterminé suivant les paramètres applicatifs et architecturaux. Cette réaffectation aura lieu dans deux cas. Le premier cas est lorsqu'il y a un changement au niveau architectural ou applicatif qui correspond à une modification de l'algorithme utilisé ou de l'implémentation. Dans sa nouvelle configuration, la tâche a besoin d'une nouvelle valeur du (CPU_workload) qu'il est nécessaire de recalculer et de réaffecter. Le second cas correspond à un dépassement d'échéance qui oblige le système d'adaptation à faire des ajustements qui peuvent imposer pour une tâche de changer d'implémentation. La détection du dépassement d'échéance est faite à travers la technique du « watchdog » qui sera expliquée dans la section(4).

La coordination de l'adaptation dans les trois couches hardware, système d'exploitation et application nous offre un modèle d'adaptation multicouche « cross layer » dont les capacités d'adaptation sont plus performantes que les adaptations monocouches comme nous l'avons montré au chapitre 1. Cette technique multicouche nécessite la coopération entre les différentes couches du système puisque le changement des paramètres de l'une peut entrainer des changements dans les autres couches. Spécifiquement, pour avoir une qualité de service pour une application donnée et qui consomme une quantité d'énergie bien fixe, nous avons besoin de configurer l'architecture adéquate dans la couche hardware, allouer un nombre de cycles processeur pour chaque tâche dans la couche système d'exploitation et modifier les paramètres applicatifs dans la couche application.

4 Adaptation multicouche

Le but du système d'adaptation est de satisfaire les consignes de l'utilisateur tout en tenant compte des données à traiter et des opportunités d'adaptation offertes par l'application et l'architecture cible. Les entrées pour le système d'adaptation sont :

- Les consignes de l'utilisateur : elles sont données par ce dernier ; la durée de vie « Ddv » souhaitée et le niveau de qualité de service minimale acceptable « QoSmin ».
- Les données à traiter : elles dépendent des contraintes imposées par l'environnement externe (type de données, type d'applications exécutées sur le système)

Cette section présente la conception de l'approche d'adaptation multi couche. Nous décrivons ainsi l'architecture et le mode de fonctionnement de cette approche.

4.1 Vue d'ensemble

L'approche d'adaptation proposée combine l'adaptation dans les trois couches, hardware, OS et application. La Figure 9 présente l'architecture globale de notre approche. Elle est composée essentiellement de :

- Un gestionnaire global qui coordonne entre les trois couches en se basant sur l'énergie disponible et les préférences de l'utilisateur (niveau minimal de la qualité de service et durée de vie souhaitée) pour donner la meilleure qualité de service possible.
- Un gestionnaire local qui coordonne entre la couche application et système d'exploitation afin de garantir le respect de la contrainte temps réel
- Un adaptateur de tâches qui permet d'ajuster les paramètres et les opérations de la tâche.
- Un adaptateur d'OS qui permet d'ajuster le nombre de cycles CPU et l'échéance affectée pour chaque tâche.
- Un adaptateur d'architecture qui s'occupe du changement de l'architecture du système
- Un moniteur de batterie qui donne une indication sur le niveau d'énergie restant dans la batterie.
- Une base de configuration qui contient des configurations pour chaque type d'application.

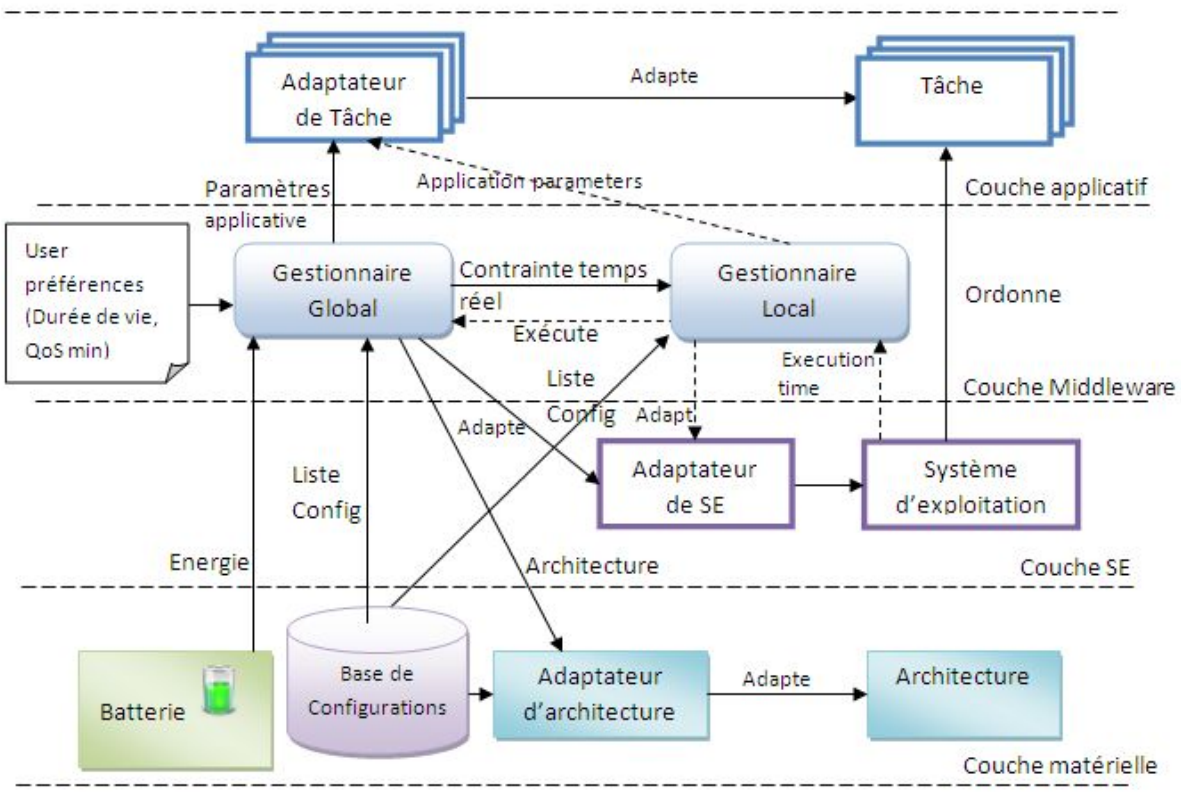

Figure 9: Approche d'adaptation

L'approche d'adaptation doit être assez performante et pas trop complexe. Le défi est donc de fournir la meilleure qualité possible pour un système tout en respectant les contraintes du système et sans être trop complexe pour ne pas consommer les ressources du système.

4.2 Gestionnaire global « GM »

Le gestionnaire global peut être activé par la tâche d'adaptation ou pour répondre à une demande du gestionnaire local. Dans de telles situations, le GM coordonne entre les trois couches (application, OS, hardware) pour choisir la meilleure configuration du système, à partir d'une base de configurations pré-caractérisées. La configuration choisie doit fournir à l'utilisateur la meilleure qualité de service possible tout en respectant les préférences de l'utilisateur et les contraintes du système (Ddv/Texe/QoS).

La Figure 10 montre le schéma du GM [Lou09c]. Ces entrées sont les préférences de l'utilisateur. Pour le moment, nous considérons deux paramètres, la durée de vie souhaitée et le niveau de qualité de service minimum accepté. Le gestionnaire global cherche à maximiser le niveau de qualité de service des applications multimédia exécutées tout en assurant les préférences de l'utilisateur.

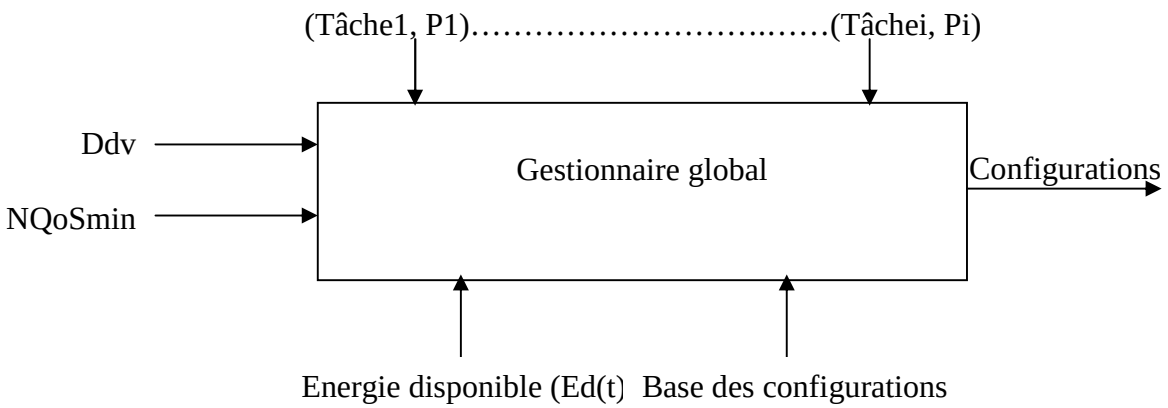

Figure 10: Le gestionnaire global

4.2.1 Formulation mathématique du problème

Plus formellement, on suppose que notre système exécute n tâches concurrentes "S1"={A_1,…..,A_k}. Chaque application « A_k » , 1<k<n, possède m différents modes de fonctionnement A^i_k ϵ "S2"={ A^1_k …… A^m_k }. Chaque mode A^i_k, correspond à une version algorithmique d'une application A_k. Chaque A^i_k peut avoir des configurations hardware/software différentes en fonction de la mise en œuvre de ses différentes fonctions (tâches): $A^i_{k,p}$, où p ϵ "S3".

Chaque $A^i_{k,p}$ est caractérisé par un temps d'exécution au pire cas noté $Texe\_A^i_{k,p}$ pour une période « P_k » et consomme au cours de cette période une énergie « Ec_k »

On cherche à fournir une qualité de service « QoS_k », pour une durée de vie « Ddv» et avec un niveau de QoS minimum « QoS_min_k». Sachant que la quantité d'énergie disponible dans la batterie est « Ed». Ce problème peut être représenté par les équations suivantes :

Maximiser F1 $= \sum_{k=1}^{n} QoS_k$ E3

Sous contraintes :

$QoS_k >= QoS\_min_k$ E4

$$\sum_{k=1}^{n} Ec_k * \frac{Ddv}{P_k} <= Ed$$ E5

$$\sum_{k=1}^{n} \frac{Texe\_A^i_{k,p}}{P_k} <= 1$$ E6

Les équations E3 et E4 permettent de fournir à l'utilisateur la meilleure qualité de service possible qui doit être supérieure au niveau de QoS minimum fixé par l'utilisateur pour chaque application.

L'équation E5 permet de garantir la contrainte de durée de vie. La quantité d'énergie consommée par toutes les tâches ne devra pas dépasser la quantité d'énergie disponible si le système fonctionne avec l'état actuel tout au long de la durée de vie.

L'équation E6 représente la condition suffisante d'ordonnançabilité du système selon l'algorithme d'ordonnancement EDF. Cette contrainte nécessite que le temps d'exécution de toutes les tâches ne doit pas dépasser 1.

Le problème de ce modèle est qu'il accepte la substitution d'une QoS d'une application par rapport à une autre ce qui peut conduire à avoir une tâche avec une excellente QoS et une autre tâche avec une faible QoS (vidéo excellente avec qualité audio médiocre par exemple). Ainsi, il est important d'homogénéiser la distribution de la QoS à toutes les applications du système. Ceci peut être assuré par une répartition équitable des ressources disponibles aux différentes applications.

Le Tableau 1 montre, avec un exemple simple, que deux solutions différentes avec des sorties de qualité de service très distinctes peuvent avoir les mêmes valeurs de la fonction objectif.

Tableau 1 : Résultat de la fonction objectif avec la fonction F1

Solutions	QoS application 1	QoS Application 2	QoS Application 3	Objective fonction
first solution	0	50	100	150
Second solution	50	50	50	150

Afin d'éviter ce problème et afin de garantir l'obtention d'une bonne qualité de service à chaque application notre fonction d'évaluation doit avoir la forme F2=Max ($\sum$ **Qos** $^{-}$ qui consiste à maximiser la fonction somme des qualités en exposant (1/k) (avec k un entier supérieur à 1).

Le Tableau 2 illustre un exemple du résultat de la fonction objectif en appliquant la deuxième formule.

Tableau 2 : Résultat de la fonction objectif avec la fonction F2

Solutions	QoS application 1	QoS Application 2	QoS Application 3	Objective Fonction (k=2)
First solution	0	50	100	17.07
Second solution ★	50	50	50	21.21

Pour démontrer que si nous avons *n* valeurs qui ont la même qualité de service totale, la recherche du maximum de la fonction somme des valeurs en exposant (1/k) (avec k un entier supérieur à 1) va nous donner un résultat où toutes les valeurs sont similaires. Nous avons utilisé la méthode de multiplicateur de Lagrange [Hif07] qui propose une stratégie pour trouver l'optimum d'une fonction sous contraintes. Néanmoins afin de vérifier que l'optimum trouvé est un maximum global nous avons appliqué la méthode Hessienne [Ste10]. Une démonstration complète de cette formule existe dans [lin10a].

4.2.2 Quantification de la consommation en énergie électrique

Nous considérons un système formé par un {CPU, mémoire, des accélérateurs matériels} Figure 11 :

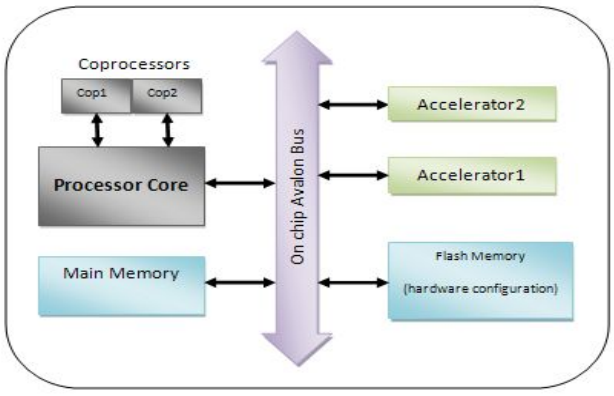

Figure 11 : Architecture d'un SoC

La consommation du système dépend de :

- La puissance statique du processeur et de la mémoire associée. Cette valeur est notée Pow_idl
- L'impact de chaque accélérateur sur la puissance statique noté Pow_conf
- La puissance consommée dynamiquement du système qui est due à l'exécution de chaque application. Cette valeur est notée Pow_app

La Figure 12 montre ces différentes puissances

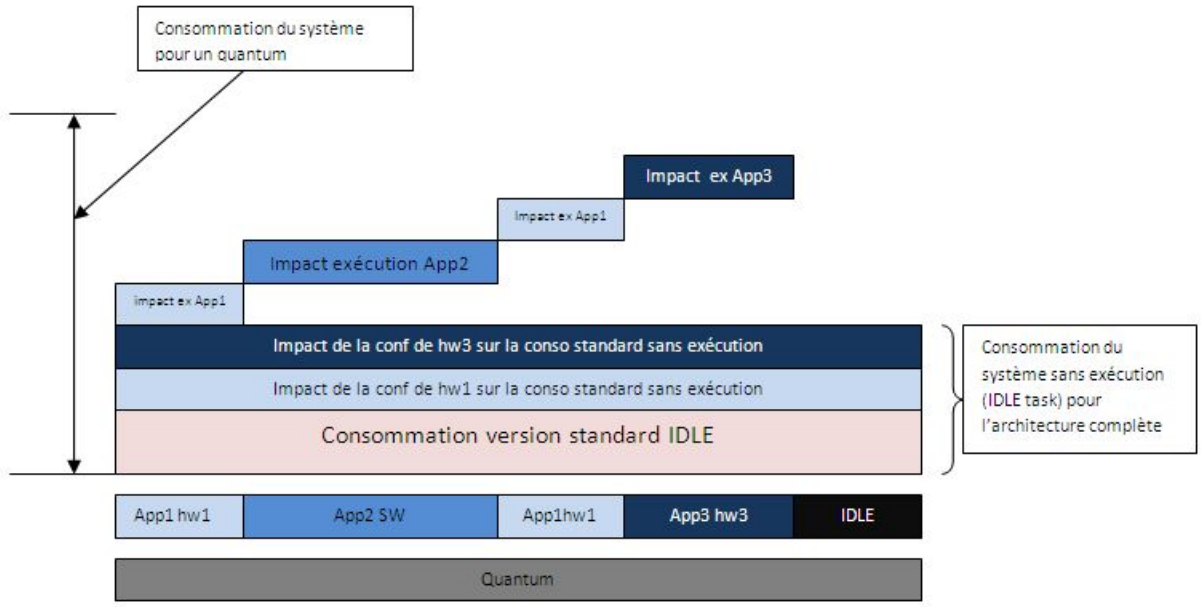

Figure 12 : Modélisation de la consommation

Ainsi avec ce modèle de consommation, l'équation E5 peut être reformulée selon l'équation E7

$$Pow\_i \quad \text{lx} \quad \sum_{i\ 1} Pow\_con\ \_i \times \quad \sum_{i\ 1} Pow\_ \quad \_i \times \frac{}{Pi} \times \quad i \qquad \text{E7}$$

Avec le budget affecté pour un quantum $Beq = \frac{}{v}$ et h est l'hyper-period du système

Donc notre problème d'optimisation peut être formulé de la manière suivante :

$$\text{Maximiser } (\sum_{1} \text{Qos}^{-} \qquad \text{E3}$$

Sous contraintes:

$$\begin{cases} Qi \quad QoS\_ \quad in & \text{E4} \\ Pow\_i \quad \text{lx} \quad \sum_{i\ 1} Pow\_con\ \_i \times \quad \sum_{i\ 1} Pow\_ \quad \_i \times \frac{}{Pi} \times \quad i & \text{E7} \\ \sum_{i=1}^{n} \frac{\text{Exi}}{\text{Pi}} <= 1 & \text{E6} \end{cases}$$

4.2.3 Recherche de la solution

Le gestionnaire global devra sélectionner la configuration adéquate pour chaque tâche (architecture+paramètres applicatifs) à partir de la base de configuration qui devra contenir

des informations relatives à chaque configuration telle que les niveaux de QoS, le nombre de cycles CPU requis, la quantité d'énergie consommée pour une période. Donc afin de choisir la meilleure configuration de chaque tâche, le GM se trouve devant un problème NP-complet, puisqu'il devra extraire toutes les combinaisons possibles qui peuvent répondre aux contraintes déjà citées. Par exemple, si on considère que notre système exécute trois tâches concurrentes et que l'on a « n » configurations possibles pour chaque tâche les GM devront vérifier n^3 solutions possibles pour extraire toutes les combinaisons possibles afin de répondre aux exigences du système.

Partant du fait que la tâche d'adaptation doit avoir un overhead minimum et pour réduire la complexité de la tâche d'adaptation, nous avons étudié différentes méthodes d'optimisation de résolution de ce type de problème.

4.2.4 Présentation des méthodes d'optimisation

L'optimisation est un ensemble de techniques permettant de trouver les valeurs des variables qui rendent optimale une fonction de réponse, appelée aussi fonction objectif.

Il existe deux classes d'algorithmes d'optimisation qui sont les algorithmes exacts et les algorithmes approchés (Figure 13).

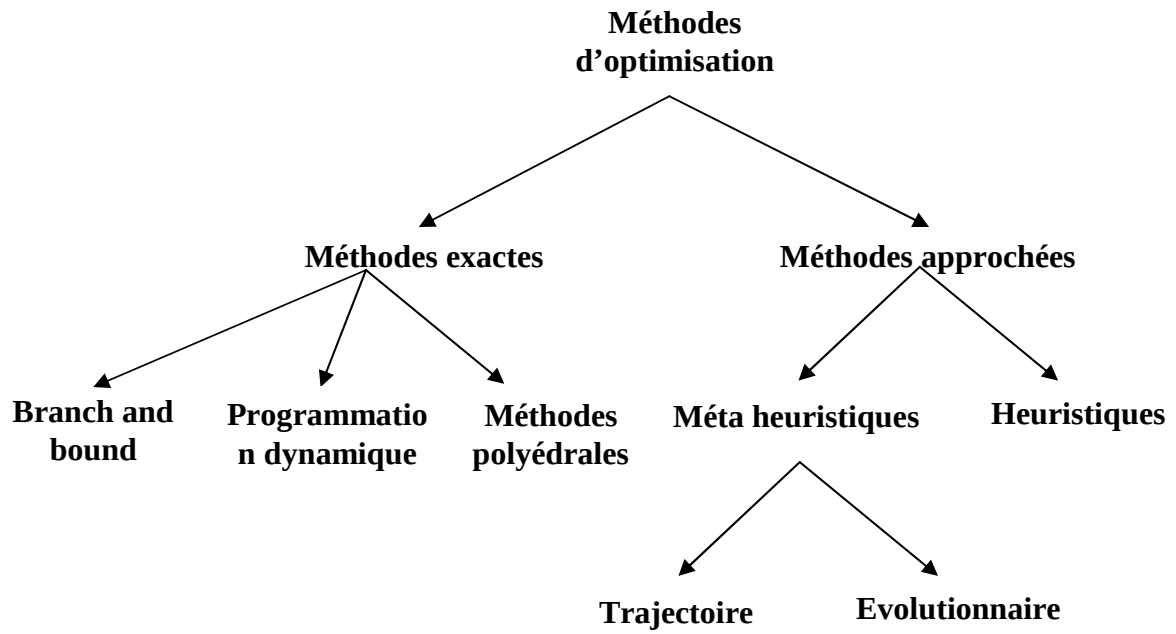

Figure 13:Classification des algorithmes d'optimisation

4.2.4.1 Méthodes exactes

Diverses méthodes de résolution exactes existent dans la littérature. Elles se caractérisent par le fait qu'elles permettent d'obtenir une ou plusieurs solutions dont l'optimalité est garantie.

Parmi ces méthodes on cite :

- la programmation dynamique
- les méthodes polyédrales
- le Branch & Bound
- la méthode de recherche par énumération explicite de toutes les solutions

Ces méthodes permettent de trouver des solutions optimales pour des problèmes de petite taille. Mais malgré les progrès réalisés en termes de technologie, le temps de calcul nécessaire pour trouver une solution risque d'être très grand puisqu'il dépend de la taille du problème. Les méthodes exactes rencontrent généralement des difficultés avec les problèmes de taille importante dans notre travail on va opter pour la dernière méthode « full enumeration».

Cependant dans certains cas nous pouvons nous contenter des solutions de bonne qualité, sans garantie d'optimalité, mais au profit d'un temps de calcul réduit. Nous utilisons pour cela une méthode approximative avec l'inconvénient de ne disposer en retour d'aucune information sur la qualité des solutions obtenues.

4.2.4.2 Les méthodes approximatives

Dans certaines situations, telles que la limite des ressources disponibles pour le système, cas de la plupart des systèmes embarqués actuels, il est nécessaire de disposer d'une solution qui permette de fournir une bonne qualité (assez proche de l'optimale). Nous parlons ainsi des approches approximatives.

Dans ce cas l'optimalité de la solution ne sera pas garantie, ni même l'écart avec la valeur optimale. Cependant, le temps nécessaire pour obtenir la solution sera beaucoup plus faible et pourra même être fixé. Typiquement ce type de méthodes, est particulièrement utile pour les problèmes nécessitant une solution dans un laps de temps très court ou pour résoudre des problèmes difficiles sur des instances numériques de grande taille.

Parmi ces méthodes, on distingue les heuristiques qui concernent un problème bien spécifique

et les méta-heuristiques qui sont plus puissantes et qui peuvent résoudre un grand nombre de problèmes.

- ***Les heuristiques***

Dans le but d'améliorer le fonctionnement d'un algorithme dans sa recherche dans l'espace des solutions d'un problème donné, le recours à une méthode *heuristique* permet d'améliorer les performances dans le processus de recherche des solutions optimales.
Feignebaum et Feldman définissent une **heuristique** comme une règle d'estimation, une stratégie, une astuce, une simplification, ou toute autre sorte de système qui limite la recherche des solutions dans l'espace des configurations possibles[Bap06].

- ***Les métaheuristiques***

Les métaheuristiques se placent à un niveau plus général encore, et peuvent être utilisées dans toutes les situations où le concepteur ne connaît pas d'heuristique efficace pour résoudre un problème donné.
En 1996, I.H. Osman et G. Laporte définissaient la méta-heuristique comme « un processus itératif qui subordonne et qui guide une heuristique, en combinant intelligemment plusieurs concepts pour explorer et exploiter tout l'espace de recherche. Des stratégies d'apprentissage sont utilisées pour structurer l'information afin de trouver efficacement des solutions optimales, ou presque-optimales » [Bap06].

Les méta-heuristiques sont souvent inspirées par des systèmes naturels, qu'ils soient pris en physique (cas du recuit simulé), en biologie de l'évolution (cas des algorithmes génétiques) ou encore en éthologie (cas des algorithmes de colonies de fourmis ou de l'optimisation par essaims particulaires).

- **Méta-heuristique trajectoire**

Comme point de départ, ces algorithmes partent d'une solution initiale proposée par le concepteur ou aléatoirement et commencent des itérations dans le but de l'améliorer ; ce qui construit une trajectoire des solutions choisies. Dans cette catégorie d'algorithmes on peut citer:

- la méthode Tabou
- la méthode de descente

- le recuit simulé
- la recherche par voisinage variable

- **Méta-heuristique évolutionnaire**

Contrairement à la méthode trajectoire, celle-ci travaille avec un ensemble de solutions simultanément, qui évoluent au cours du temps : ce qui permet en conséquence une amélioration de l'espace des solutions. Dans cette seconde catégorie, on recense :

- l'optimisation par essaim particulaire
- les algorithmes génétiques
- les algorithmes à estimation de distribution
- les algorithmes par colonies de fourmis
- le path relinking (ou chemin de liaison)

Dans la section suivante nous nous intéressons à la description détaillée du mode de fonctionnement d'un algorithme de chaque catégorie l'algorithme génétique et recuit simulé.

4.2.5 Algorithme génétique

4.2.5.1 Définition

Les algorithmes génétiques font partis des algorithmes d'optimisations évolutionnaires. L'utilisation de la théorie de l'évolution comme un modèle informatique pour trouver une solution optimale peut être justifiée par le fait qu'elle permet de trouver une solution parmi un grand nombre de possibilités dans un délai raisonnable. De ce fait, les algorithmes génétiques ont été inspirés du concept de la sélection naturelle développée par Charles Darwin et des méthodes de combinaison des gènes introduites par Mendel pour traiter des problèmes d'optimisation [Lay09].

Les premiers travaux réalisés dans ce domaine ont débuté dans les années cinquante, lorsque certains biologistes américains ont simulé des structures biologiques sur un ordinateur, puis dans les années soixante, sur la base des travaux antérieurs, John Holland a développé les principes de base des algorithmes génétiques dans le cadre de l'optimisation mathématique [Lac04].

4.2.5.2 Caractéristiques principales

Puisque ces algorithmes sont inspirés de la génétique classique, le vocabulaire employé est le même que celui de la génétique. Ainsi pour décrire un algorithme génétique les termes utilisés sont :

- **Gène :** Le plus petit élément d'une solution potentielle au problème posé.
- **Chromosome / Individu:** un chromosome (parfois aussi appelé un génome) est un ensemble fini de gènes. Il définit une proposition de solution au problème que l'algorithme génétique essaie de résoudre. Dans la plupart des cas un individu est présenté par un seul chromosome.
- **Population :** elle est formée par l'ensemble des individus utilisés par l'algorithme génétique.
- **Génération :** c'est l'évolution de la population à un instant t donné.
- **Fonction de performance (fitness) :** c'est un type particulier de fonction objectif qui prescrit l'optimalité d'une solution. Elle est utilisée pour calculer le coût d'un point de l'espace de recherche.

4.2.5.3 Principe

Pour un problème d'optimisation, les étapes de l'algorithme génétique sont les suivantes (Figure 14) :

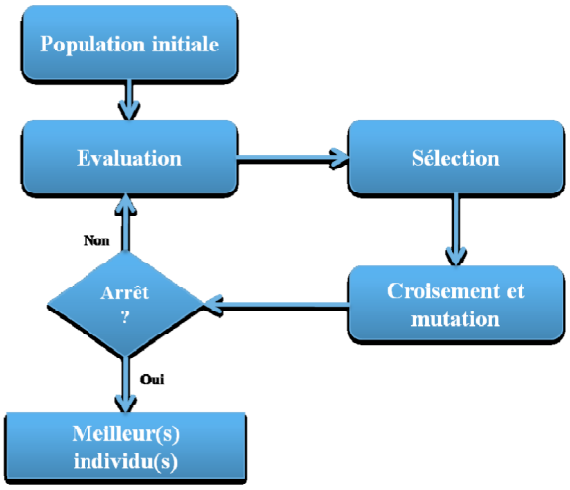

Figure 14: Principe de base d'un algorithme génétique

1. Générer aléatoirement une population de n chromosomes x
2. Évaluer l'adaptabilité f(x) de chaque chromosome
3. Créer une nouvelle population
    a. Sélectionner 2 parents chromosomes
    b. Croiser les 2 parents pour obtenir 2 enfants
    c. Muter les enfants obtenus
    d. Placer les enfants dans la population
4. Itérer à partir de l'étape (2) jusqu'à ce qu'à attendre le critère d'arrêt [Lac04].

Le critère d'arrêt peut être choisi :

- Aléatoirement : le nombre d'itérations qui correspond au nombre de générations.
- Basé sur le critère de convergence : valeur de fitness incluse dans un intervalle désigné au paravent.

Dans notre cas le critère d'arrêt est atteint après un nombre de générations choisies d'avance par l'utilisateur du système selon la précision attendue.

4.2.6 Algorithme du recuit simulé

4.2.6.1 Historique

La structure complexe de l'espace des configurations d'un problème d'optimisation difficile a conduit trois chercheurs de la société IBM S.Kirkpatrich, C.D.Gelatt et M.P. Vecchi à proposer en 1983 une nouvelle méthode itérative en s'inspirant de la technique expérimentale du recuit utilisée par les métallurgistes pour simuler l'évolution d'un système physique vers son équilibre thermodynamique [Sia03].

La technique du recuit consiste à chauffer préalablement le matériau pour lui conférer une énergie élevée puis à le refroidir lentement en marquant des paliers de température de durée suffisante. Cette stratégie de baisse contrôlée de la température conduit à un état solide stable correspondant à l'optimum absolu de l'énergie.

4.2.6.2 Définition

Le recuit simulé est souvent présenté comme la première méta-heuristique qui a mis en œuvre une stratégie d'évitement des minima locaux [Bap06]. Cette technique est l'un des exemples typiques des méthodes de trajectoire à base de solution unique qui construit une trajectoire dans l'espace des solutions en tentant de se diriger vers des solutions optimales.

L'idée d'utiliser la technique du recuit consiste à introduire en optimisation un paramètre de contrôle qui joue le rôle de la température. La température du système à optimiser doit avoir le même effet que la température du système physique : elle doit conditionner le nombre d'états accessibles et conduire vers l'état optimal si elle est abaissée de façon lente et bien contrôlée.

4.2.6.3 Principe

En pratique, cette technique exploite l'algorithme de Metropolis, qui permet de décrire à une certaine température T le comportement d'un système en équilibre thermodynamique.

Pour une procédure de maximisation de la fonction « objectif », le principe est le suivant :

- En partant d'une configuration donnée, le système subit une modification élémentaire si cette transformation a pour effet d'améliorer la fonction objectif du système, elle est acceptée, si elle provoque au contraire une diminution ΔE de la fonction objectif elle peut être acceptée tout de même avec la probabilité e$\exp(\Delta E / T)$.
- En appliquant itérativement ce procédé tout en gardant la température constante, l'équilibre thermodynamique sera atteint concrètement au bout d'un nombre suffisant de modifications. La température est alors abaissée avant d'effectuer une nouvelle série de transformations. La loi de décroissance par paliers de la température est souvent empirique tout comme le critère d'arrêt du programme [Sia03].

La Figure 15 présente l'organigramme de l'algorithme du recuit simulé.

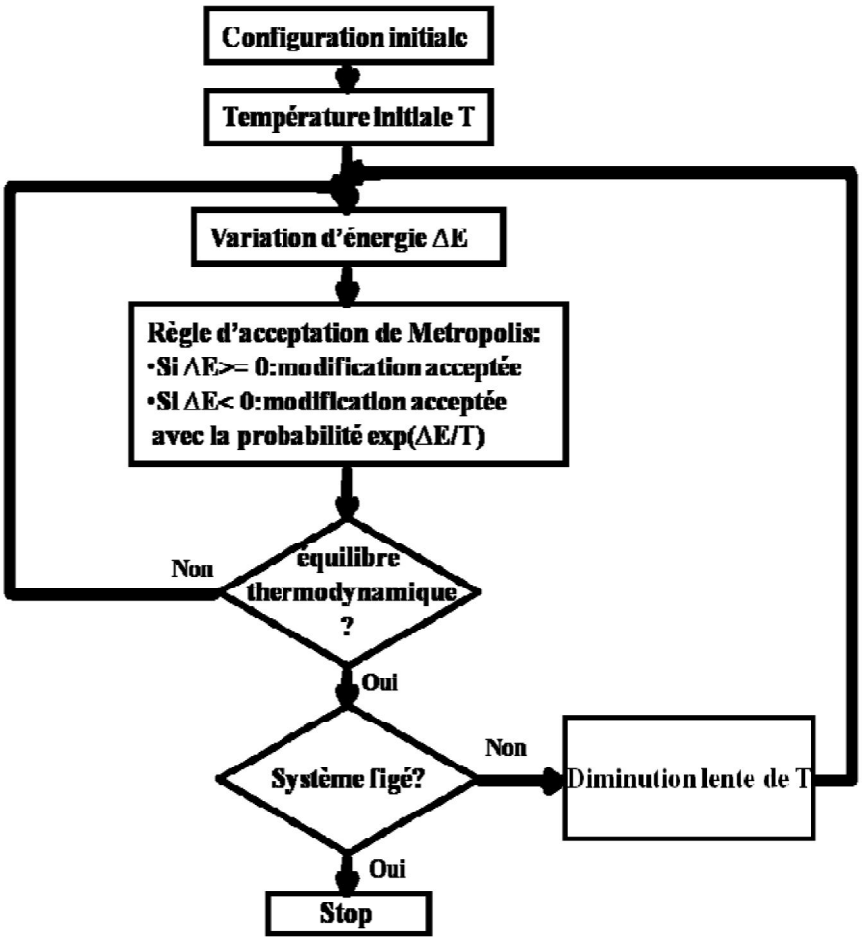

Figure 15: Organigramme de l'algorithme du recuit simulé

4.2.6.4 Paramétrage

- **Solution initiale :** La solution initiale peut être choisie au hasard dans l'espace des solutions possibles ou par un expert du problème.

- **Mémoire :** La méthode du recuit simulé est une méthode sans mémoire. En effet, l'algorithme pourrait converger vers une certaine solution **S** en ayant visité auparavant une solution **S'** de qualité supérieure. C'est pourquoi afin de l'améliorer on peut ajouter une mémoire à long terme qui stocke les meilleures solutions rencontrées.

- **Température initiale :** Son choix dépend de la qualité de la configuration de départ. Si cette configuration est choisie aléatoirement, il faut une température relativement élevée. Si au contraire, la solution de départ est déjà assez bonne, parce qu'elle a été choisie par un expert du problème considéré par exemple, une température initiale assez basse sera suffisante.

- **Décroissance de la température :** la température suit souvent une loi géométrique décroissante : $T_{k+1} = T_k \cdot \alpha$, avec α un nombre réel appartenant à l'intervalle]0..1[.

- **Arrêt du programme :** lorsque la température a atteint une valeur négligeable ou bien lorsque aucune solution n'a été accepté au cours du palier : par exemple, lorsque trois paliers successifs de température ont été descendus sans qu'aucune solution nouvelle n'ait pu être trouvée.

4.3 Le gestionnaire local

Dans ce qui précède, nous avons présenté le principe de fonctionnement du gestionnaire global dont le rôle est de choisir une configuration pour le système qui lui permet de respecter ses contraintes. Mais ce dernier ne possède pas les techniques pour contrôler leur respect. Cependant, nous travaillons dans un système multimédia temps réel mou qui permet les dépassements des échéances dans une certaine limite. Nous avons proposé une technique de contrôle du respect de la contrainte temps réel qui se base sur l'utilisation d'un gestionnaire local.

4.3.1 Principe de fonctionnement

Notre technique de contrôle intervient dans les deux couches application et système d'exploitation. Elle se base sur le principe de watchdog. Il s'agit dans notre cas d'un logiciel permettant de s'assurer que le système ne reste pas bloqué dans une situation qui altère son bon fonctionnement suite à la violation de ses contraintes. C'est une protection destinée à reconfigurer le système si une action définie n'est pas exécutée dans un délai imparti. La Figure 16 présente un schéma qui décrit d'une manière générale la technique de contrôle.

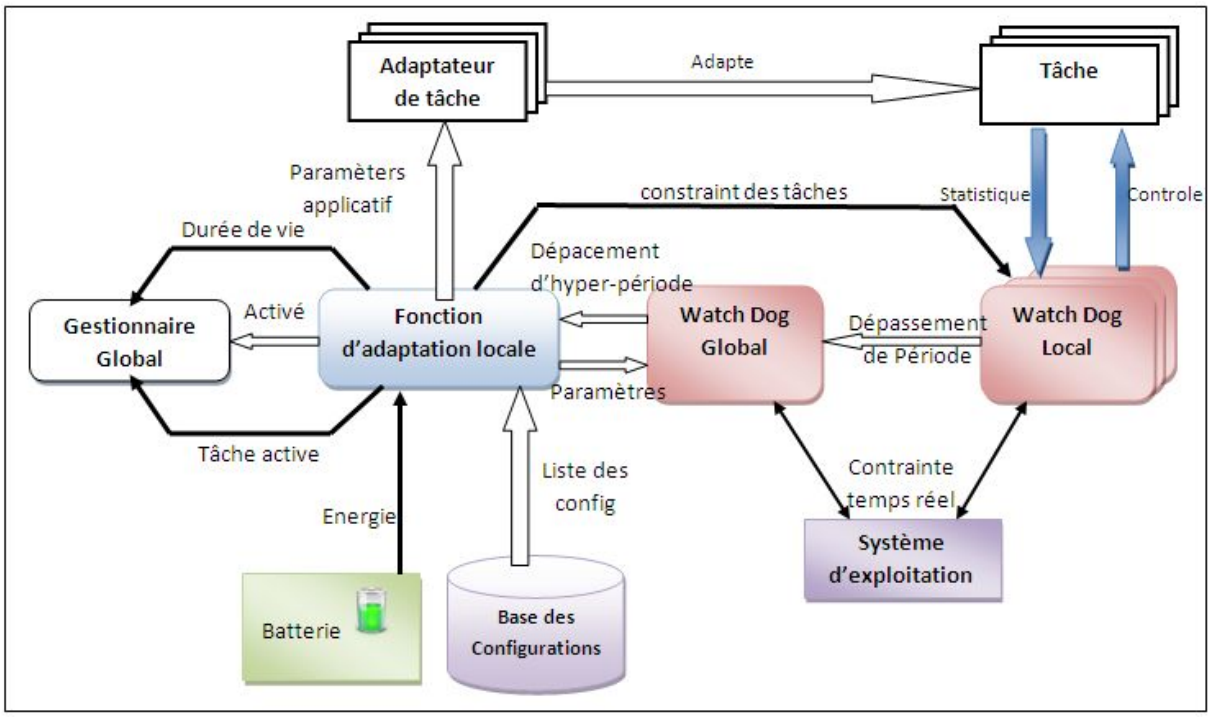

Figure 16:Schéma de principe du gestionnaire local

Nous avons défini deux types de watchdog : un watchdog local responsable sur le contrôle d'exécution au niveau tâche, et un watchdog global chargé du contrôle d'exécution au niveau système sur toute l'hyper-période (HP), le plus petit commun multiple de toutes les périodes. En effet, afin de respecter les contraintes temporelles, toutes les tâches doivent être exécutées à chaque période. Et dans l'hyper-période h du système, une tâche Ti de période Pi doit être exécutée ni = h/Pi fois.

Cependant, un dépassement d'HP peut toujours avoir lieu et c'est au watchdog global de le gérer.

On affecte à chaque tâche un watchdog local responsable de contrôler son exécution et maintenir son état pour chacune de ses périodes. Il est créé au moment de la création de la tâche, et supprimé lorsqu'elle l'est aussi. En cas de détection de dépassement d'échéance, le watchdog local alerte le watchdog global qui enregistre ce dépassement. A l'expiration de HP, le watchdog global analyse la liste des violations d'échéance pour voir si elles ont une influence sur le fonctionnement global du système. S'il n'y a pas d'influence, le système continue à fonctionner avec les mêmes paramètres applicatifs et d'OS. Mais s'il détecte un dépassement d'HP, il en déterminera alors l'origine.

Ensuite, il fera l'appel à une fonction d'adaptation qui se chargera de prendre une décision d'adaptation selon le dépassement. Cette fonction choisit de nouveaux paramètres applicatifs

pour la tâche correspondante afin de résoudre le problème de dépassement. Cette recherche est beaucoup moins complexe que celle du gestionnaire global puisqu'elle ne prend en considération que les configurations de la tâche source du dépassement. Donc la configuration choisie est celle qui a les mêmes attributs architecturaux de la configuration actuelle avec un temps d'exécution inférieur à la valeur actuelle. Une fois une nouvelle configuration est choisie le gestionnaire local envoi les nouveaux paramètres aux adaptateurs des tâches et de l'OS pour qu'ils modifient leurs paramètres suivant les caractéristiques de la nouvelle configuration choisie. Dans le cas où le gestionnaire local ne trouve pas de solution pour résoudre le problème, il fait appel au gestionnaire global pour reconfigurer la totalité du système.

4.3.2 Choix de l'algorithme d'ordonnancement

Comme nous l'avons déjà mentionné, l'ordonnancement a pour rôle d'allouer le processeur aux diverses tâches. L'ordonnanceur est une procédure de service du système d'exploitation. Au cours de l'exécution à chaque laps de temps le système d'exploitation détermine si le système a besoin d'un ré-ordonnancement. Si c'est le cas, le système invoquera son ordonnanceur pour choisir une nouvelle tâche pour l'exécuter.

Nous donnons par la suite un bref rappel sur les algorithmes d'ordonnancement et nous expliquons la solution retenue.

4.3.2.1 Ordonnanceurs préemptifs

Un ordonnanceur est dit préemptif ou « avec réquisition », si l'exécution du processus courant peut être arrêtée par l'ordonnanceur (suite à une interruption) pour laisser le processeur volontairement à une autre tâche. Il est dit non préemptif ou « sans réquisition » le cas contraire. [Dan04]

Dans le cas d'un système d'exploitation non préemptif, une commutation de contexte ne peut avoir lieu que si la tâche courante termine son exécution ou passe à l'état bloquée.

4.3.2.2 Lois d'ordonnancements classiques

Les politiques d'ordonnancement déterminent la prochaine tache qui sera mise dans l'état « courant ». Les politiques d'ordonnancement les plus répandues sont :

- premier arrivé, premier servi « PAPS » (ou FIFO)

- le tour de rôle ou tourniquet (round robin, circulaire),
- l'ordonnancement par priorité.

Dans la politique FIFO, les tâches « prêtes » sont placées dans une file FIFO gérée suivant la politique premier arrivé premier servi. Une tâche qui passe à l'état prêt s'insère en fin de file, et lorsque la tâche courante libère le processeur, c'est la tâche qui est en tête de file qui devient la tâche courante.

Dans la politique de tourniquet, les tâches sont placées dans une file FIFO et sont activées périodiquement. La période d'activation s'appelle un quantum de temps. A la fin d'un quantum, la tâche qui est en tête de file est activée.

4.3.3 Les algorithmes d'ordonnancement pour les systèmes temps réel

Pour les systèmes temps réel on distingue deux types de politiques d'ordonnancement :

- ordonnancement hors ligne *(of line)*: l'ordonnancement est pris avant l'exécution du système.
- ordonnancement en ligne *(on line) :* il prend les décisions d'ordonnancement durant l'exécution du système.

4.3.3.1 Les ordonnanceurs sous contrainte de temps

4.3.3.1.1 Description

L'ordonnanceur est une pièce fondamentale d'un système temps réel. Pour chaque processeur, l'ordonnanceur est en charge de définir la séquence d'exécution des processus, qui est une séquence infinie d'éléments du type : (date, identifiant travail) tout au long de l'évolution du système. Donc le système d'exploitation doit disposer des mécanismes nécessaires pour la gestion du temps et ce pour prendre les bonnes décisions aux bons moments.

4.3.3.1.2 Ordonnanceurs à priorités simples

Dans cette partie, nous définissons les algorithmes d'ordonnancement à priorité fixe.

Les ordonnancements à priorités les plus courants sont les suivants :

- RM (pour Rate Monotonic) : ordonnancement à priorité statique pour les tâches périodiques/sporadiques avec échéance relative égale à la période/délai d'inter-arrivée. Une tâche est d'autant plus prioritaire que sa période est petite.

- DM (pour Deadline Monotonic) : ordonnancement à priorité statique pour les tâches sporadiques. Une tâche est d'autant plus prioritaire que son échéance relative est petite. DM équivaut à RM quand l'échéance relative est égale à la période.
- EDF (pour Earliest Deadline First) : ordonnancement à priorité dynamique. Une tâche est d'autant plus prioritaire que sa date d'échéance est plus proche.
- LLF (pour Least Laxity First) : ordonnancement à priorité dynamique. Les tâches sont d'autant plus prioritaires que leur laxité est faible à la date courante.

4.3.3.1.3 Propriétés des ordonnancements

- EDF et LLF sont optimaux parmi les ordonnancements préemptifs non oisifs [LL73]. EDF est également optimal relativement à la minimisation du retard maximal des tâches.
- EDF est optimal parmi les ordonnancements non-préemptifs non oisifs [Bru06].
- RM est optimal parmi les ordonnancements préemptifs non oisifs à priorité statique, pour les tâches périodiques non-concrètes/synchrones ou sporadiques, lorsque l'échéance relative est égale à la période (ou au délai d'inter-arrivée dans le cas sporadique) [LL73].
- DM est optimal parmi les ordonnancements préemptifs non oisifs à priorité statique, pour les tâches périodiques non-concrètes/synchrones ou sporadiques à échéance relative inferieure à la période.
- Dans les cas où les tâches périodiques concrètes ne sont jamais synchrones, ni RM, ni DM ne sont optimaux parmi les ordonnancements préemptifs non oisifs à priorité statique. Dans [Aud04], on donne l'adéquation optimale des priorités pour l'ordonnancement préemptif non oisif à priorité statique pour le cas général (i.e. qui reste valable quand les échéances relatives et les périodes ne sont pas reliées).
- DM est optimal parmi les ordonnancements non-préemptifs non oisifs à priorité statique, pour les tâches périodiques non-corsètes/synchrones ou sporadiques à échéance relative inferieure à la période [Bat98]. Pour l'ordonnancement à priorité dynamique, les démonstrations utilisent le fait que si on dispose d'un ordonnancement faisable, il faut alors faire intervertir 2 tâches pour le respecter l'ordre d'ordonnancement EDF ou LLF conserve la faisabilité de l'ordonnancement. Le même type de démonstration est utilisé avec les priorités des tâches pour démontrer l'optimalité de RM/DM.

4.3.4 Choix de l'ordonnanceur

La méthode d'adaptation nécessite la disponibilité d'ordonnanceurs à caractéristiques spécifiques. A titre d'exemple un ordonnanceur à priorité fixe ne peut être utilisé. En effet, si

on utilise cet ordonnanceur avec les budgets de temps affectés par le GM on peut rencontrer des cas là ou l'une des tâches ne peut pas terminer son exécution au cours de l'hyper période. Le graphe de la Figure 17 illustre l'un de ces cas.

Considérons un système qui exécute trois tâches T1 (période=10, priorité=1, Texe=3), T2(période=10,priorité=2,Texe= 3) et T3 (période=15,priorité=1,Texe=6). Le graphe d'exécution des différentes tâches sur un ordonnanceur à base de priorité est illustré par la Figure 17 :

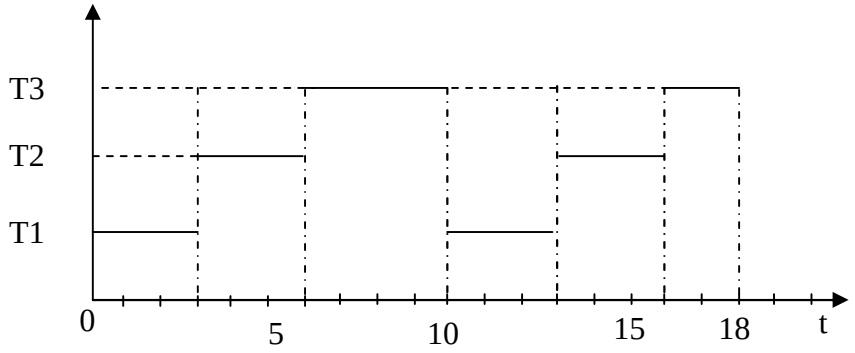

Figure 17: Graphe de séquences avec MicroC

On remarque bien que la tâche T3 a dépassé sa période(t=18) et ceci est dû au fait que les tâches T1 et T2 sont plus prioritaires ce qui entraîne son interruption. On est donc amené à chercher un ordonnanceur adéquat qui permet de garantir l'exécution des différentes tâches dans leur propre période. Nous avons choisi de travailler avec l'ordonnanceur EDF (earliest deadline first) qui se base sur l'affectation dynamique des priorités puisqu'il a une condition (E6) nécessaire et suffisante pour que le système soit ordonnançable.

Le système ne peut être ordonnançable que si $U = \sum_{i\ 1} \frac{Ci}{i}$ 1 E8

Avec Ci et Ti sont respectivement le temps d'exécution et la période de la tâche i.

Corollaire : EDF est un algorithme optimal

Si U>1 aucun algorithme n'est capable d'ordonnancer l'ensemble des tâches

Si U<1 l'ensemble des tâches est ordonnançable par EDF

Reprenons le même exemple du graphe suivant et considérons que la valeur du deadline de chaque tâche est égale à la période. Le graphe d'exécution des différentes tâches sur un système d'exploitation qui se base sur un ordonnanceur de type EDF est illustré par la Figure 18.

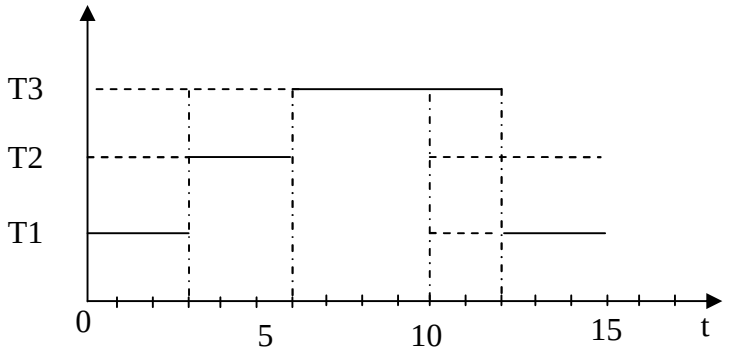

Figure 18: Graphe de séquences avec EDF

On remarque bien que toutes les tâches terminent leur exécution avant la fin de la période. L'ordonnanceur EDF répond bien à nos besoins.

5 Etape de caractérisation des configurations

Cette étape permet à partir d'une analyse ou à travers des outils de construire la base des configurations. Cette étape se fait hors ligne. Nous présentons dans cette partie les méthodes et les outils utilisés dans notre travail afin de mettre en place la base des configurations.

La construction de la base des configurations nécessite deux principales étapes. La première étape consiste d'abord à la détermination de l'ensemble des tâches candidates à une implémentation HW ensuite à la mise en place des différentes architectures possibles du système. La deuxième étape consiste à caractériser les différentes configurations possibles du système en agissant sur les paramètres de l'application.

5.1 Mise en place des configurations

Le choix du nombre de configurations du système dépend d'un compromis précision/complexité. En effet, la disponibilité d'un nombre important de configurations permet d'avoir une adaptation fine qui permet de mieux ajuster les contraintes. Cependant, cette finesse d'adaptation est obtenue au dépend d'une complexité plus importante du système d'adaptation (due à l'augmentation de l'espace occupée pour la sauvegarde des configurations) et un coût plus élevé du système de gestion des contraintes (complexité plus grande de l'algorithme de recherche de la solution).

5.2 Partitionnement logiciel/matériel

Dans notre travail nous avons opté à une méthode de partitionnement logiciel/matériel qui se base sur :

- l'utilisation des résultats fournis par le profilage de l'application sur la plateforme de

travail et l'utilisation de l'outil design trotter. Cette approche est représentée par la Figure 19 [Lou09b].

- l'outil Design Trotter développé au laboratoire Lab-STICC est étendu dans le cadre de la thèse de nader Ben Amor [Ben05].

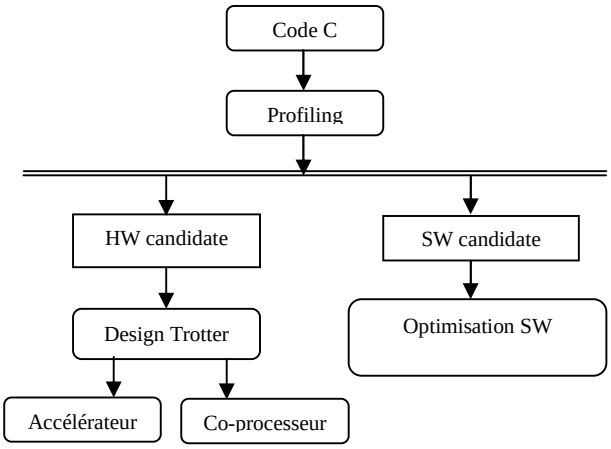

Figure 19: Approche de partitionnement SW/HW

5.2.1 Profilage de l'application :(Profiling)

Cette étape permet de donner au concepteur des statistiques sur les différentes fonctions exécutées dans un programme (temps d'exécution, pourcentage d'occupation du processeur, nombre d'itération, l'occupation mémoire, etc). Cette technique peut donner une idée sur les chemins suivis pour l'exécution de l'application.

Dans notre travail nous avons utilisé cette technique pour comparer le temps d'exécution et le nombre d'itérations des différentes fonctions qui constituent l'application. Les fonctions qui prennent plus de temps d'exécution seront favorisées pour une implémentation hardware.

5.2.2 Analyse par design trotter

Les entrées de l'environnement Design Trotter se présentent sous la forme de fonctions décrites en langage de haut niveau (en langage C). L'environnement Design Trotter fournit en sortie différents types d'estimations et d'informations visant à guider le concepteur de systèmes embarqués. Parmi celles-ci nous pouvons citer l'estimation système par métriques [Mou03b].

L'objectif des métriques est d'analyser les fonctions de l'application afin d'en déterminer deux paramètres : d'une part leur orientation traitement, Contrôle ou Mémoire (à l'aide des

métriques MOM et COM) et d'autre part leur criticité (parallélisme moyen disponible) à l'aide du métrique gamma. La Figure 20 montre l'environnement Design Trotter [Ben07].

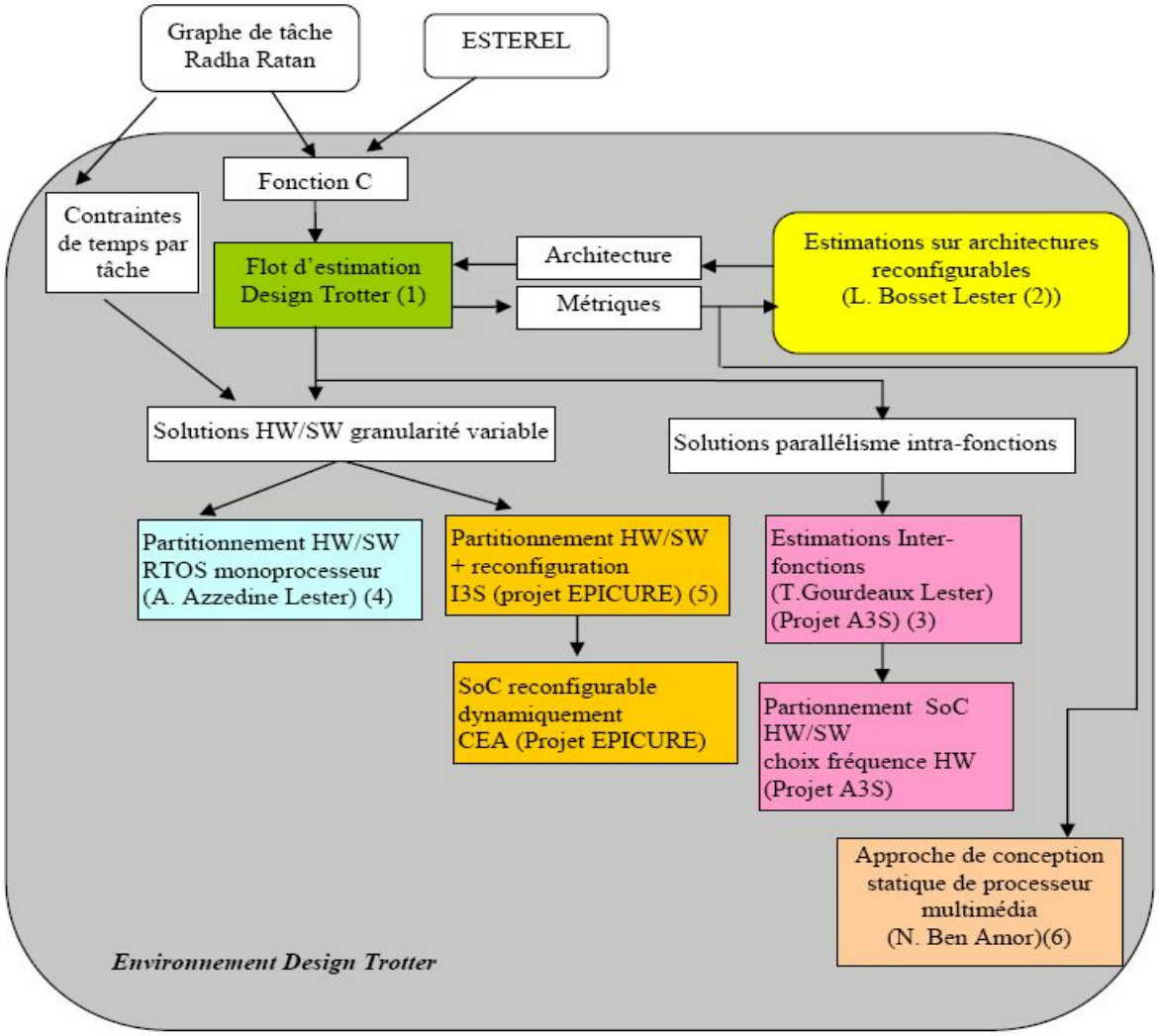

Figure 20: Environnement Design Trotter

Ces métriques sont calculées pour chaque fonction de l'application et pour chaque niveau hiérarchique de graphe. Par la suite en analysant ces résultats numériques on peut classer les différentes fonctions qui constituent notre application selon leurs comportements. Par exemple, d'une part, les fonctions orientées contrôlent avec peu d'occasions de parallélisme et sont donc un candidat prometteur d'une implémentation logicielle. D'autre part, les fonctions à parallélisme élevé sont candidats pour les implémentations matérielles.

En se basant sur l'information issue de la première étape, on construit deux classes de fonctions. Les premières fonctions incluent des candidats pour l'implémentation matérielle, elles peuvent être par exemple des fonctions flot de données avec des opérations de test (avec une valeur petite de COM et une valeur grande de γ) et un temps d'exécution élevé. Cela va

être additionné comme un accélérateur matériel au cœur du processeur embarqué pour obtenir une architecture à coût réduit. Ces modèles incluent des modules utilisés qui couvrent plusieurs applications multimédia comme la transformation d'images (DCT, le transformateur d'ondelettes DWT, etc.) et des algorithmes classiques de traitement d'image comme le filtrage, et aussi les fonctions de traitement de vidéo comme l'estimation de mouvement.

La deuxième classe contient des fonctions logicielles, elle peut être par exemple des fonctions avec beaucoup de contrôle et peu de parallélisme (des valeurs grandes de COM et petites de γ) et un temps d'exécution bas.

5.3 Caractérisation des configurations

Elle consiste à déterminer pour chaque configuration son temps d'exécution au pire cas, son énergie consommée et son niveau de QoS.

5.3.1 Calcul de Texe

Pour caractériser les configurations du système nous avons besoin de calculer le temps d'exécution de l'application au pire cas (Worst Case Execution Time WCET). Différents travaux ont été menés pour l'estimation de cette valeur. Dans notre travail nous avons utilisé une mesure directe sur la carte vue la précision de cette méthode. En effet, une fois les différentes configurations matérielles sont mises en place on peut les utiliser pour faire les mesures et ce à travers un « timer » hardware. Bien entendu, il faut initialiser les paramètres de l'algorithme de façon qu'il suive le plus long chemin possible.

Le timer est un composant hardware qui joue le rôle d'un décompteur qui se décrémente à chaque n ticks d'horloge qui est appelé période. Elle est fixée lors de la conception de la partie hardware. Pour mesurer le temps d'exécution d'une tâche, on lit alors, la valeur du timer avant et après son exécution. Le temps pris par cette période est la différence entre les deux valeurs multipliées par la période du timer. [Lou09a]

5.3.2 Mesure de la consommation

Cette étape peut être déterminée par simulation ou par mesure directe. La 1ère solution utilise l'environnement de simulation relatif à la cible choisie (par exemple Quartus « Power estimator » si le processeur NIOS est choisi). Cette solution exige des temps de simulation relativement longs. De plus, les valeurs fournies ne sont pas exactes. La deuxième solution consiste à mesurer la puissance consommée Pconso par mesure directe. Cette deuxième

méthode est plus rapide et plus précise mais elle exige un matériel bien spécifique avec une haute précision.

5.3.3 Quantification de la QoS

Un modèle doit être défini pour quantifier la QoS de chaque configuration connaissant ses attributs applicatifs. Il permet de caractériser la qualité de service fournie à l'utilisateur en fonction des paramètres liés à l'application multimédia.

6 Conclusion

La notion d'adaptation est devenue indispensable dans la conception des systèmes embarqués autonomes. Cette adaptation peut intervenir dans les trois couches du système afin de bénéficier au maximum de son apport. Dans ce contexte nous avons décrit dans ce chapitre, un système d'adaptation à contraintes multiples dédié aux systèmes embarqués. Ce système permet de satisfaire les trois contraintes Ddv, Texe et QoS. Afin de mettre en place ce système, deux étapes sont nécessaires. La première étape est une étape « hors ligne » qui permet la construction de la table des configurations et sa caractérisation. Pour cela, une analyse préliminaire de l'application doit être effectuée afin de déterminer les attributs applicatifs susceptibles d'influencer le compromis Ddv, Texe et QoS. La deuxième étape est une étape en ligne formée par deux activités : l'activité d'observation et celle d'adaptation. L'activité d'observation permet le suivi des paramètres Ddv, Texe et QoS. L'activité d'adaptation permet d'adapter le système en choisissant la configuration adéquate de la table des configurations.

L'approche proposée est générique puisqu'elle est indépendante de l'application et de la plateforme de prototypage. Afin de la mettre en place pour pouvoir évaluer ses performances il est judicieux de passer par un environnement de conception logiciel/matériel et une application cible. Dans le chapitre suivant nous entamons une étude de cas pour valider l'approche proposée. Nous décrivons ainsi les éléments principaux de l'application et de l'environnement choisis.

CHAPITRE 4 : Etude de cas & Environnement de validation

1 Introduction

La validation est une étape primordiale dans la réalisation de n'importe quel projet. Cette étape permet d'une part de valider et de combler les défaillances de la partie théorique étudiée et d'autre part elle permet de mettre en évidence les performances des travaux réalisés. Afin de valider l'approche proposée nous sommes amenés à choisir un environnement de prototypage logiciel/matériel et une application cible. Ceci est le thème du présent chapitre. Nous présentons dans sa première partie l'application de synthèse d'images 3D connue sous le nom du pipeline graphique 3D ou moteur 3D. Cette application répond bien à nos besoins puisqu'elle présente des paramètres applicatifs modifiables selon le compromis (temps d'exécution, qualité de l'objet fourni).

La deuxième partie de ce chapitre est consacrée à la présentation de l'environnement de conception des systèmes sur puce. Il s'agit de l'environnement QUARTUS qui permet de concevoir des systèmes sur puces sur les circuits FPGA de la société ALTERA et qui intègre aussi le système d'exploitation MicroC_OS-II.

2 Aperçu sur l'application synthèse d'image 3D

La chaîne de production d'une image 3D est appelée pipeline graphique. Elle est formée par l'ensemble des opérations nécessaires pour afficher un objet 3D regardé depuis une position et avec une orientation donnée. A chaque fois que l'état de la scène change (c'est-à-dire à chaque mouvement de la caméra), elle doit être redessinée. Pour cela, la description des objets de la scène doit être traduite en points 2D à l'écran.

Les éléments d'entrée de ce pipeline sont des triangles qui sont plus pratiques que les quadrilatères ou autres polygones pour les calculs. En effet, ils ne peuvent être ni vrillés (dont les sommets ne sont pas coplanaires) ni concaves. La plupart des moteurs 3D effectuent une triangulation des différentes faces avant de les envoyer au pipeline graphique.

2.1 Différentes étapes du pipeline

Tout objet graphique complexe est d'abord transformé en un ensemble de triangles. La Figure 21 montre deux exemples de cette transformation.

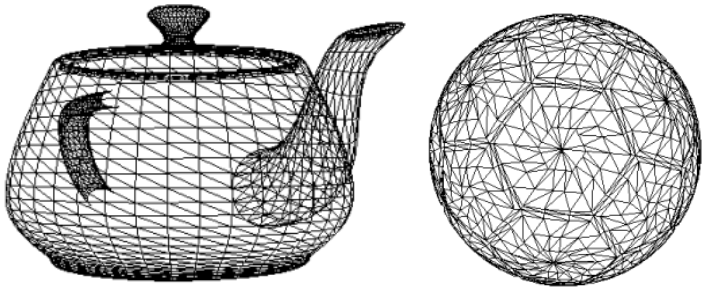

Figure 21: Objets transformés en un ensemble de triangles

Plusieurs éditeurs 3D permettent d'exporter un objet dans un fichier ASCII, comme le cas de 3D Studio. Ce fichier contient le nom de l'objet, le nombre de sommets et le nombre de faces. Les sommets sont décrits avec leurs composantes x, y et z. Une fois les sommets initialisés, les faces seront énumérées avec leurs trois sommets. Chacun des triangles subit alors les différentes étapes du pipeline graphique. La Figure 22 représente ces étapes.

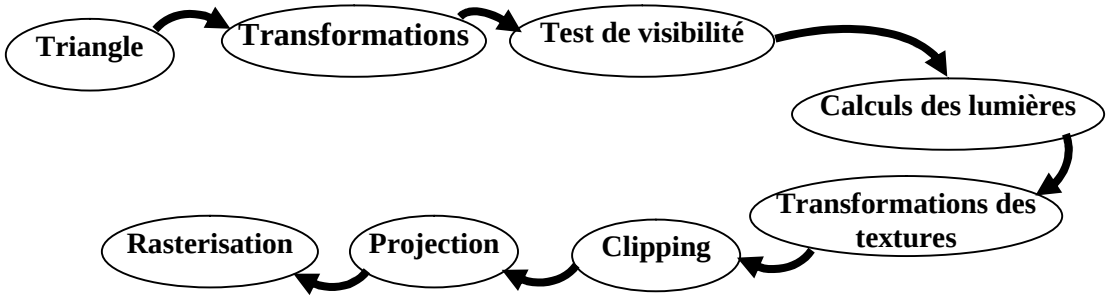

Figure 22: Les différentes étapes du pipeline [lou04]

2.1.1 Transformations

Les transformations sont des translations, des rotations et des homothéties des triangles pour les convertir de leurs repères locaux au repère global de la scène puis dans le repère de la caméra. Pour ce faire, la position et l'orientation de l'utilisateur sont utilisées. En fait, ce sont des opérations matricielles donnant le transformé d'un point par une ou plusieurs des matrices de transformations.

2.1.2 Test de visibilité

Le test de visibilité d'un triangle se fait grâce à son vecteur normal donnant sa face avant. Si l'angle formé entre le vecteur normal N et le vecteur de vision V (allant de la face à l'œil) est aiguë alors la face sera visible. Sinon elle sera invisible. Ce calcul se fait en utilisant le

produit scalaire entre N et V. Si celui-ci est négatif alors cela signifiera que la face est invisible.

Dans la suite, on détaille la détermination des fonctions de « calcul de normale » et « sa normalisation » parce qu'on aura besoin dans la phase de réalisation des accélérateurs hardwares.

2.1.2.1 Détermination de la normale à une face

Pour calculer la normale à une face triangulaire, on prend les trois vecteurs délimiteurs du triangle, dans le sens des aiguilles d'une montre. La Figure 23 illustre cette étape. On soustrait celui du milieu des 2 autres, et on obtient 2 vecteurs dont le produit vectoriel est la normale à la face.

Soit les trois vecteurs $\mathbf{V_1}: \begin{pmatrix} x_a \\ y_a \\ z_a \end{pmatrix}$, $\mathbf{V_2}: \begin{pmatrix} x_b \\ y_b \\ z_b \end{pmatrix}$ et $\mathbf{V_3}: \begin{pmatrix} x_c \\ y_c \\ z_c \end{pmatrix}$, on détermine $\mathbf{V_{n1}}$ et $\mathbf{V_{n2}}$ tel que :

$$\mathbf{V_{n1}}: \begin{cases} V_{n1.x} = x_a - x_b \\ V_{n1.y} = y_a - y_b \\ V_{n1.z} = z_a - z_b \end{cases} \qquad \text{E9}$$

$$\mathbf{V_{n2}}: \begin{cases} V_{n2.x} = x_c - x_b \\ V_{n2.y} = y_c - y_b \\ V_{n2.z} = z_c - z_b \end{cases} \qquad \text{E10}$$

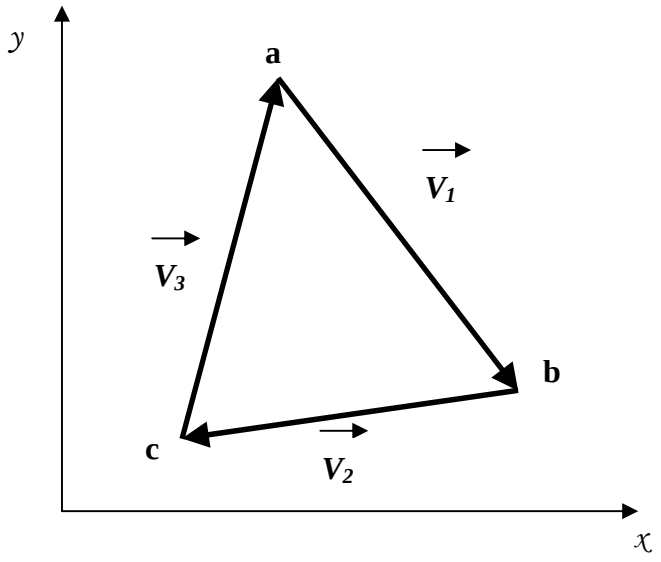

Figure 23 : Détermination de la normale

La normale de la face est **N (Nx,Ny,Nz)** tel que :

$$\begin{cases} N_x = V_{n1.y} \times V_{n2.z} - V_{n2.y} \times V_{n1.z} \\ N_y = V_{n1.z} \times V_{n2.x} - V_{n2.z} \times V_{n1.x} \\ N_z = V_{n1.x} \times V_{n2.y} - V_{n2.x} \times V_{n1.y} \end{cases} \quad \text{E11}$$

2.1.2.2 Normalisation

La normalisation est une étape très importante dans la détermination du signe du cosinus de l'angle entre deux vecteurs (lumière et normale) pour le calcul de l'intensité de la couleur. En effet, un produit scalaire entre deux vecteurs normalisés donne directement ce cosinus.

La normalisation du vecteur normal N(Nx,Ny,Nz) est donnée par le système d'équation E12.

$$\begin{cases} N_{xn} = \dfrac{N_x}{\sqrt{N_x^2 + N_y^2 + N_z^2}} \\ N_{yn} = \dfrac{N_y}{\sqrt{N_x^2 + N_y^2 + N_z^2}} \\ N_{zn} = \dfrac{N_z}{\sqrt{N_x^2 + N_y^2 + N_z^2}} \end{cases} \quad \text{E12}$$

2.1.3 Calculs des lumières

Les modèles d'illumination, ci-dessous, prenant en compte seulement les interactions entre une surface et la source lumineuse, servent à déterminer l'intensité de la couleur en un point.
L'intensité de la lumière est donnée par l'équation E13
I=lumière ambiante + lumière diffuse+lumière spéculaire [lou04] E13

2.1.3.1 Lumière ambiante

La lumière ambiante correspond au modèle le plus simple. On considère qu'il existe une source lumineuse présente partout éclairant de manière égale (constante) dans toutes les directions. Le calcul de l'intensité de la lumière ambiante est donné par l'équation E14.

$$I_{amb} = \boldsymbol{\rho}_a I_a \quad \text{E14}$$

I_{amb} : intensité de la lumière ambiante.

ρ_a: est le facteur qui détermine la quantité de lumière ambiante réfléchie par la surface et en fonction des propriétés matérielles de la surface ($0 \leq \rho a \leq 1$).

I_a: l'intensité de la lumière.

2.1.3.2 Lumière due à une réflexion diffuse

Dans le modèle de réflexion diffuse, l'intensité en un point d'une surface dépend de l'angle formé entre le rayon de lumière qui touche le point de la surface et la normale à cette surface. Plus l'angle formé entre le rayon de lumière et la normale au plan est faible, plus l'intensité lumineuse réfléchie visible par l'observateur est forte. L'équation E 15 donne cette intensité.

$$I_{diff} = \boldsymbol{\rho}_d\, I_l\, ||\mathbf{Np}||\, ||\mathbf{Lp}|| \cos\theta = \boldsymbol{\rho}_d\, I_l\, (N_{px}.L_{px} + N_{py}.L_{py} + N_{pz}.L_{pz}) \qquad \text{E 15}$$

Idiff : intensité de la lumière due à une réflexion diffuse.

ρd : coefficient de réflexion diffuse de la surface $0 \leq \rho d \leq 1$

Il: l'intensité de la source lumineuse.

Np (Npx,Npy,Npz) :la normale à la surface à un point P.

Lp(Lpx,Lpy,Lpz) : la direction du point P à la source lumineuse.

θ: l'angle formé entre Np et Lp.

2.1.3.3 Lumière due à une réflexion spéculaire

Le modèle de réflexion spéculaire se différencie du modèle de diffusion par le fait de faire intervenir le point d'observation. Dans ce modèle les rayons de lumière sont réfléchis par symétrie par rapport à la normale à la surface. Ce modèle correspond aux propriétés de "miroir" des objets. L'équation E16 calcule l'intensité de la lumière due à une réflexion spéculaire.

$$I_{sp} = \boldsymbol{\rho}_s\, I_l\, (||\mathbf{Np}||\, ||\mathbf{Lp}|| \cos\theta)^n \qquad \text{E16}$$

Isp : intensité de la lumière due à une réflexion spéculaire.

ρs : coefficient de réflexion spéculaire $0 \leq \rho s \leq 1$

n : coefficient de la diffusion autour du rayon réfléchi.

2.1.4 Transformations des textures

Cette étape permet de transformer les textures avant qu'elles ne soient appliquées au triangle dans l'étape de Rastérisation. Ce sont des transformations 2D sur les images qui sont un cas simplifié des transformations 3D précédentes. Si aucune texture ne doit être appliquée au triangle, cette étape sera rejetée. Dans notre travail on n'a pas tenu compte de cette étape puisqu'on n'utilise pas de textures dans nos objets.

2.1.5 Clipping (fenêtrage)

Dans cette étape on élimine les triangles qui ne font pas partie du volume de vue (dans notre cas l'écran de l'ordinateur) et on découpe ceux en partie visible selon leurs intersections avec le volume de vue. La Figure 24 présente un exemple de fenêtrage d'une figure.

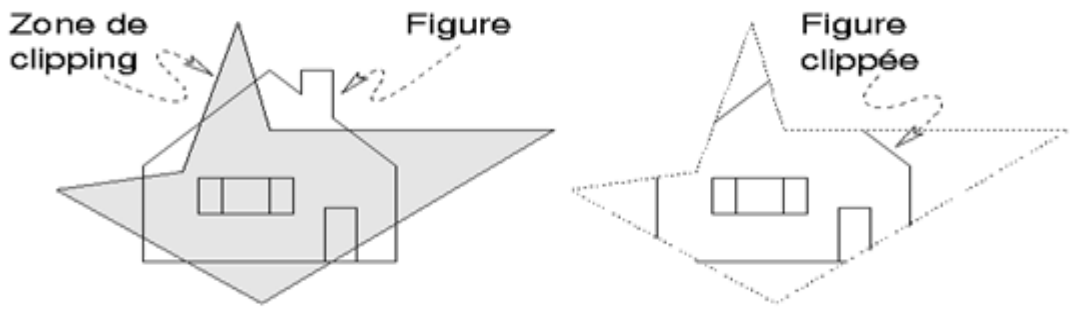

Figure 24: Clipping d'une figure

2.1.6 Projection

La projection est la transformation qui permet de donner la position du point image sur le plan à partir d'un point dans l'espace. Il s'agit donc de déterminer ce que l'on doit dessiner sur le plan de l'écran pour que l'observateur voie la même chose sur le plan comme s'il observait vraiment l'objet. La Figure 25 représente la projection d'un objet sur un écran.

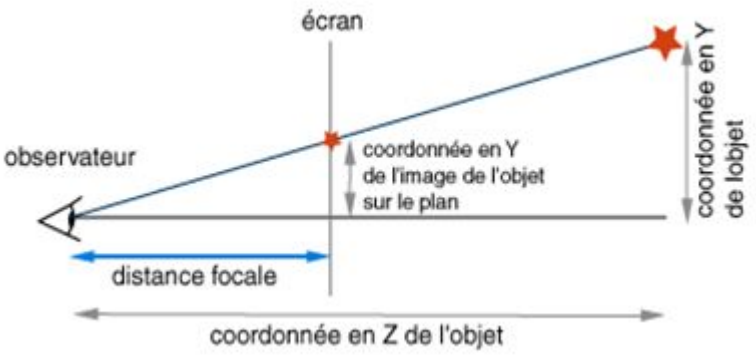

Figure 25: Projection

Les coordonnées de l'objet en question sont données par le système d'équation E17.

$$\begin{cases} C_{imagex} = -\dfrac{D_F \times C_x}{C_z} \\ C_{imagey} = -\dfrac{D_F \times C_y}{C_z} \end{cases} \qquad \text{E17}$$

- C_{imagex} : coordonnée en x de l'image de l'objet
- C_{imagey} : coordonnée en y de l'image de l'objet
- D_F : distance focale.
- C_x : coordonnée suivant x de l'objet.
- C_y : coordonnée suivant y de l'objet.
- C_z : coordonnée suivant z de l'objet.

2.1.7 Rastérisation

La rastérisation est l'étape transformant les formes géométriques 3D en des pixels sur l'écran tout en donnant un aspect réel à l'objet 3D en question.

La solution la plus simple pour effectuer le rendu d'une surface consiste à calculer l'illumination en chaque point visible de la surface. Cette méthode est très coûteuse en temps de calcul. Dans cette partie, nous allons voir les différentes méthodes permettant de diminuer le coût en temps en n'effectuant le calcul d'illumination que pour un nombre limité de points.

2.1.7.1 Ombrage plat

La méthode d'ombrage la plus simple pour les facettes polygonales est l'ombrage plat. Elle consiste à calculer l'intensité de couleurs pour un seul point de la surface que l'on veut représenter. Ensuite, on applique la même intensité pour toute la surface. La Figure 26 montre une application d'ombrage plat sur un triangle.

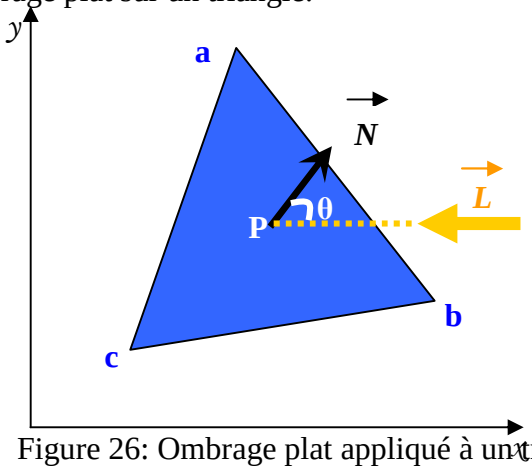

Figure 26: Ombrage plat appliqué à un triangle

Soit P un point de la surface, l'intensité de couleurs est calculée par l'équation de Lambert (E18).

$$I_P = I_{amb} + I_{diff} + I_{spec} \qquad E18$$

I_P : intensité de couleurs au point P.

I_{amb}, I_{diff} et I_{spec} sont respectivement définis dans (E14, E 15 et E16).

Si Np et Lp sont unitaires, à partir de l'équation E18, on obtient une nouvelle équation de Lambert (E19) plus simple.

$$I_P = I_{amb} + \boldsymbol{\rho}_d I_l \cos\theta + \boldsymbol{\rho}_s I_l \cos\theta \qquad E19$$

Si $\cos\theta < 0$, l'intensité résultante sera la valeur de l'intensité de la lumière ambiante, sinon on la calcule selon l'équation de Lambert.

Une fois l'intensité de couleurs calculée, on l'applique sur toute la surface du polygone en utilisant un algorithme de balayage de lignes. Le résultat d'ombrage plat appliqué sur une sphère est représenté par la Figure 27.

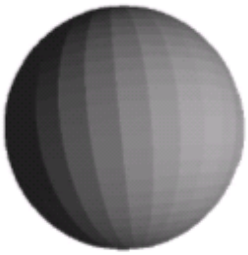

Figure 27: Ombrage plat appliqué à une sphère

Cette méthode a tendance à trop faire ressortir les polygones qui représentent un objet comme le montre la Figure 27.

2.1.7.2 Ombrage de Gouraud

La méthode développée par Gouraud élimine les discontinuités d'intensité de couleurs sur une facette polygonale par interpolation des valeurs d'intensité aux sommets de la facette. Cette technique permet d'avoir un lissage qui "gomme" les frontières entre les polygones que l'on obtient avec un ombrage plat.

La Figure 28 présente une application de l'ombrage de Gouraud à un triangle ainsi que les points clefs utilisés dans l'algorithme. Cet algorithme nécessite un calcul des intensités de

couleurs dans les trois sommets du triangle (I1, I2 et I3), une interpolation de l'intensité de couleurs entre les sommets et une interpolation horizontale (selon l'axe des x) de l'intensité de couleurs.

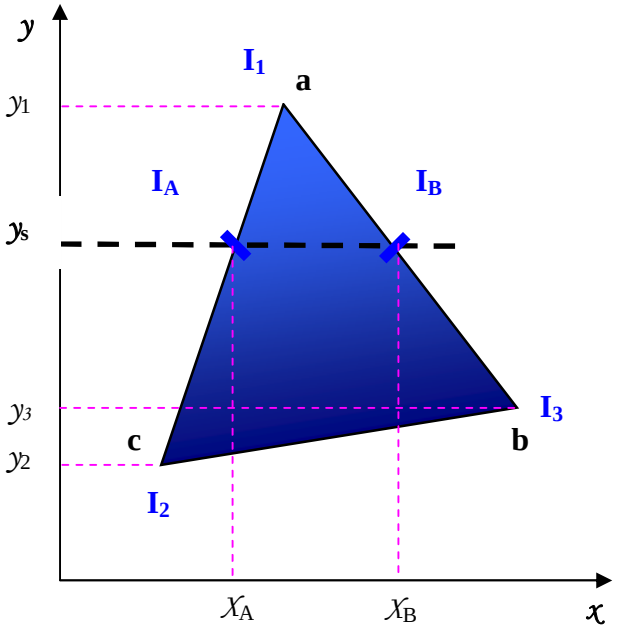

Figure 28: Ombrage de Gouraud appliqué à un triangle

Calcul de l'intensité de couleurs dans les sommets

Le calcul de l'intensité de couleurs dans un sommet du triangle nécessite d'abord la détermination des normales aux sommets. La normale à un sommet est la moyenne des normales des faces partageant ce sommet. Par la suite, l'intensité est calculée à l'aide de l'algorithme d'ombrage plat.

Interpolation de l'intensité de couleurs entre les sommets

Dans cette étape, on considère une ligne de balayage (ys) qui fait le parcourt du triangle en commençant par le sommet (a). A chaque déplacement vertical de (ys), on calcule les couleurs aux extrémités IA et IB respectivement selon les équations E20 et E21.

$$I_A = \frac{(y_s - y_2) \times I_1 + (y_1 - y_s) \times I_2}{y_1 - y_2} \qquad \text{E20}$$

$$I_B = \frac{(y_s - y_3) \times I_1 + (y_1 - y_s) \times I_3}{y_1 - y_3} \qquad \text{E21}$$

Interpolation horizontale de l'intensité de couleurs

Pour avoir une interpolation horizontale de l'intensité de couleurs entre IA et IB, en faisant le parcourt de la ligne de balayage pixel par pixel, on incrémente l'intensité de couleurs par (ΔIS) selon l'équation E22.

$$\Delta I_S = \frac{I_B - I_A}{x_B - x_A} \qquad \text{E22}$$

La Figure 29 montre une sphère en utilisant l'ombrage de Gouraud.

Figure 29 : Ombrage de Gouraud appliqué à une sphère

2.2 Graphe de tâches de l'application 3D

Il faut dire que nous avons un code original (qui a des limites qu'il faut citer) qui utilise des fichiers ascii en entrée, et qui génère sur « écran un seul objet 3D animé avec l'algo plat et gouraud selon le modèle de couleur à spécifier

Nous avons réalisé le graphe de tâches associé à cette application. Il est illustré par la Figure 30.

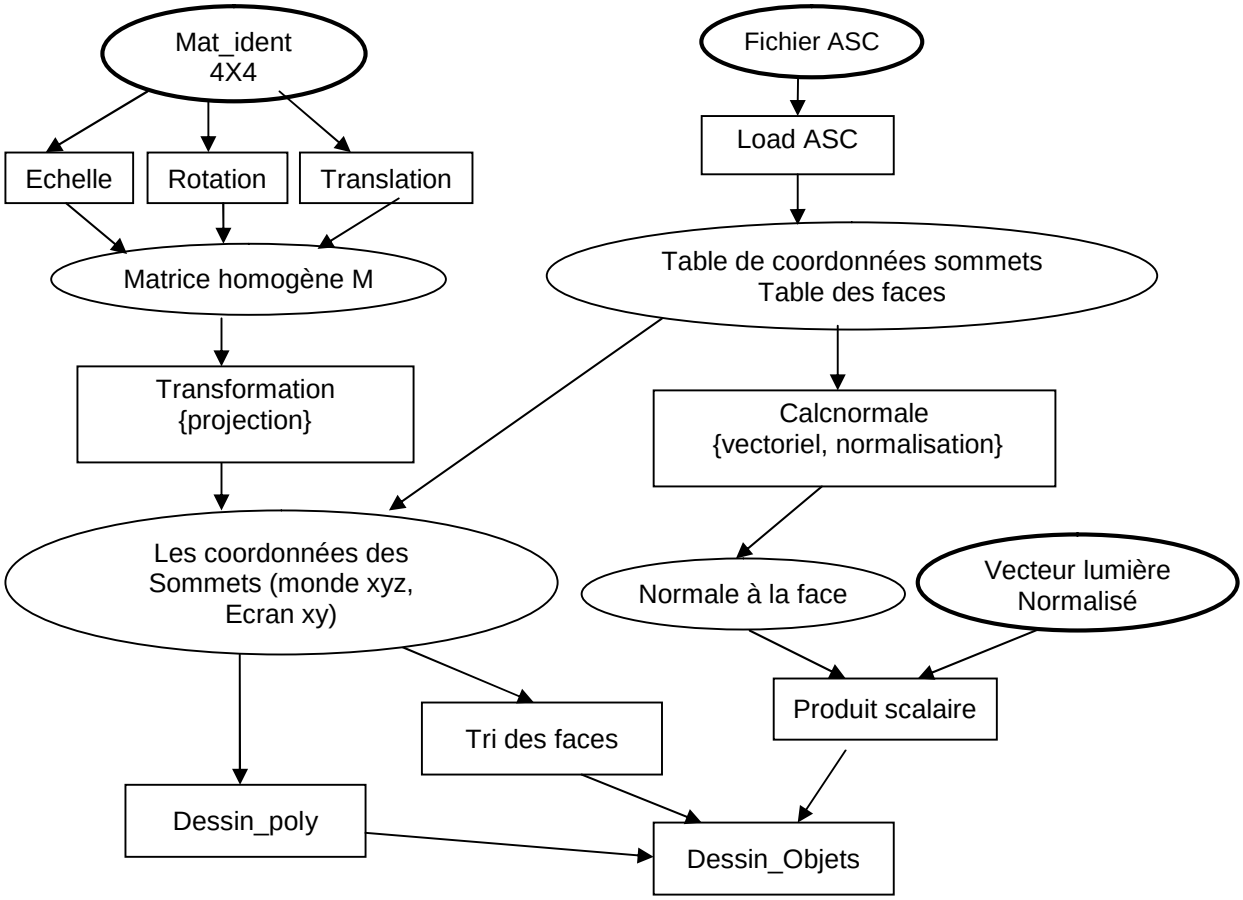

Figure 30: Graphe de tâche de l'application synthèse 3D

2.3 Modification de l'application « synthèse 3D »

Afin de pouvoir appliquer les décisions d'adaptation (algorithmique) du module « adaptateur » il est nécessaire que le code original soit modifié pour qu'il puisse:

- ✓ générer un nombre d'objets 3D différents sur l'écran
- ✓ varier le nombre de polygones représentant un objet 3D.
- ✓ varier le type de d'ombrage de l'objet (Plat ou Gouraud).

Le principe consiste à dégrader la qualité (en réduisant le nombre de polygones ou l'algorithme d'ombrage) si nécessaire au profil de la préservation de l'autonomie du système.

2.3.1 Construction des configurations logicielles

Comme première étape, nous avons créé différentes configurations $\{A^{i}_{k,im}\}$ d'objets 3D. Dans un premier temps, ces configurations sont purement logicielles (Im ={logicielles}) avec deux paramètres applicatifs d'adaptation :

- Le nombre variable de triangles
- Le type d'algorithmes d'ombrage.

o ***Variation du nombre de triangles***

Pour cela, on a utilisé l'éditeur Autodesk 3Ds max 2010. Cet outil permet la modification du nombre de polygones constituant l'objet. En effet, l'augmentation du nombre de polygones entraine l'amélioration de la qualité et vice versa. La Figure 31 montre l'effet de la modification du nombre de polygones sur la qualité de l'objet 3D.

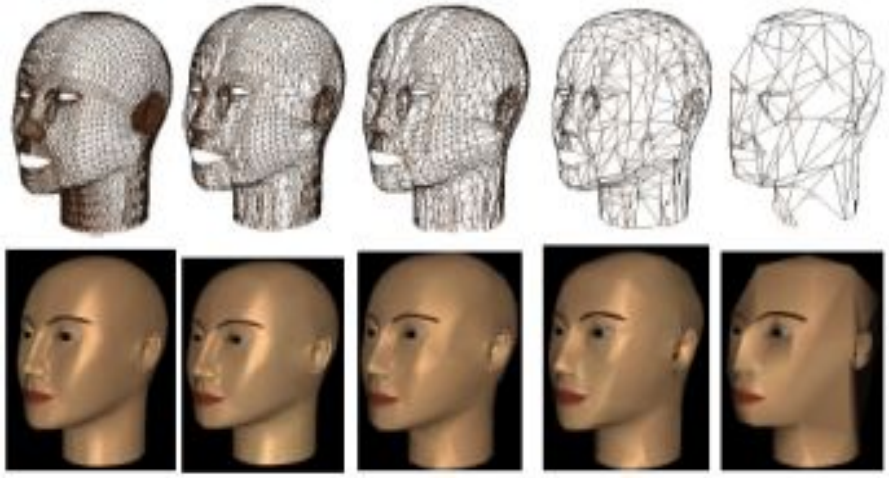

Figure 31 : Objets 3D avec qualités différentes

Les objets crées sont ensuite enregistrés sous forme de fichier d'extension «.3Ds ». Pour les exploiter par l'application synthèse 3D, ces fichiers doivent être transformés en fichiers ASCII d'extension « .asc ». Pour cela nous avons utilisé l'outil *Wcvt2pov* pour la réalisation de cette conversion [Ben07].

Le fichier ASCII créé contient le nom de l'objet, le nombre de sommets et le nombre de faces. Les sommets sont décrits avec leurs composantes x, y et z. Une fois les sommets initialisés, les faces seront énumérées avec leurs trois sommets. Ce fichier sera ensuite fourni comme entrée pour l'application « synthèse 3D ».

o ***Modification du type d'ombrage***

Nous avons modifié le code de l'application 3D afin d'être capables de traiter deux types d'algorithmes d'ombrage qui sont l'algorithme plat et l'algorithme Gouraud avec un nombre de polygones qui peut varier au cours du fonctionnement du système. Bien entendu, nous

avons modifié la plupart des structures de données utilisées afin de pouvoir modifier le type d'ombrage et le nombre de polygones de l'objet au cours du fonctionnement du système.

2.3.2 Développement d'une version multi-applications

Pour tester notre technique d'adaptation, nous devons traiter plusieurs applications multimédia simultanément. Pour cela, nous avons modifié le code principal de l'application 3D qui fait le traitement d'un seul objet 3D en un code qui peut manipuler plusieurs objets à la fois dans la même scène. On suppose que chaque objet 3D constitue une application indépendante et que chaque application représente une seule tâche qui s'exécute indépendamment des autres tâches (une tâche ne peut pas utiliser les accélérateurs marériels des autres).

2.4 Intégration des services de MicroC/OS-II

Afin d'exécuter le code de la synthèse 3D sur MicroC/OS, nous devons les répartir en tâches qui coopèrent pour générer une scène d'objets 3D choisis par l'utilisateur. La Figure 32 présente le principe utilisé.

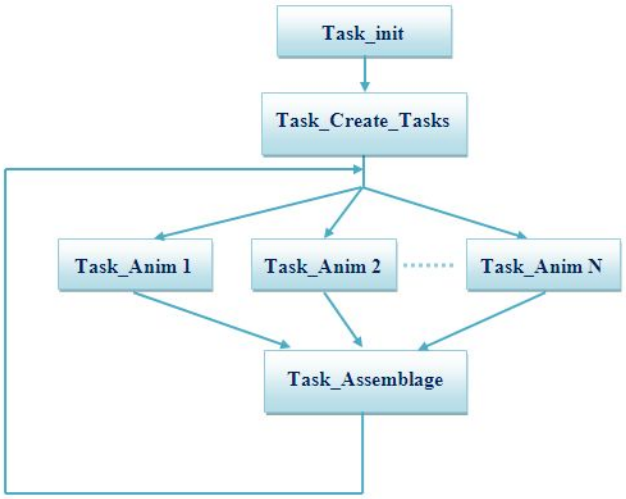

Figure 32 : Scénario du fonctionnement de l'application 3D avec les services MicroC/OS

Tout d'abord nous avons défini une tâche initiale « task_init » qui permet de :

- spécifier le nombre d'objets N à afficher dans la scène.

- Faire l'appel des fonctions indépendantes du traitement de l'animation des objets :
  - ✓ Preparepal () pour la préparation de la palette des couleurs.
  - ✓ Precalc () pour le calcul des tableaux de sinus et cosinus.
  - ✓ Normalise () pour la normalisation du vecteur de lumière).
- Ensuite selon le nombre d'objets choisis dans la tâche précédente nous faisons la création des tâches d'animation. Chaque tâche « Task_Anim i » (avec $0< i \leq N$) manipule l'animation d'un objet 3D de la scène.
- Une fois que toutes les tâches d'animation ont terminé leurs traitements, nous faisons l'appel de la tâche « Task_Assemblage » qui permet l'assemblage de tous les objets dans une même scène. Après l'exécution de la tâche d'assemblage nous revenons à l'exécution des tâches d'animation et ainsi de suite.

3 Environnement de conception des configurations mixtes

Divers types d'environnement de conception de système sur puce existent. Dans notre travail nous avons choisi celui fourni par Altera. L'inconvénient de cet environnement est qu'il ne permet pas de faire de la reconfiguration dynamique. Pour remédier à ce problème toutes les configurations hardware seront embarquées et le système switch entre elles suivant les besoins du système. Bien entendu nous supposons que les autres configurations n'existent pas dans le système.

Ce choix à été fait vu la disponibilité ainsi que la maitrise qu'on a développé de se type de plateforme dans notre laboratoire. Par ailleurs, la contribution de la thèse est de traiter le problème d'adaptation et non pas la reconfiguration dynamique qui est traitée dans d'autres travaux (ex. thèse de Linfeng YE, dans le laboratoire Lab-STICC [Lin10b]).

Cet environnement de développement comporte essentiellement :

- un environnement de conception hardware/software formé par un ensemble de logiciels :
  - Quartus : compilation et simulation du hardware

- Programmer : configuration de l'FPGA par la partie hardware
- SOPC Builder : conception de la partie hardware par assemblage d'IP
- Nios IDE : implémentation et exécution de la partie software par émulation ou sur la carte

- une carte pour le prototypage du système.
  - L'FPGA utilisée est le (Stratix III)
  - Le processeur Nios-II
  - Un cable JTAG pour la configuration de l'FPGA et l'exécution de la partie software.

La Figure 33 montre le schéma de la carte de développement utilisée

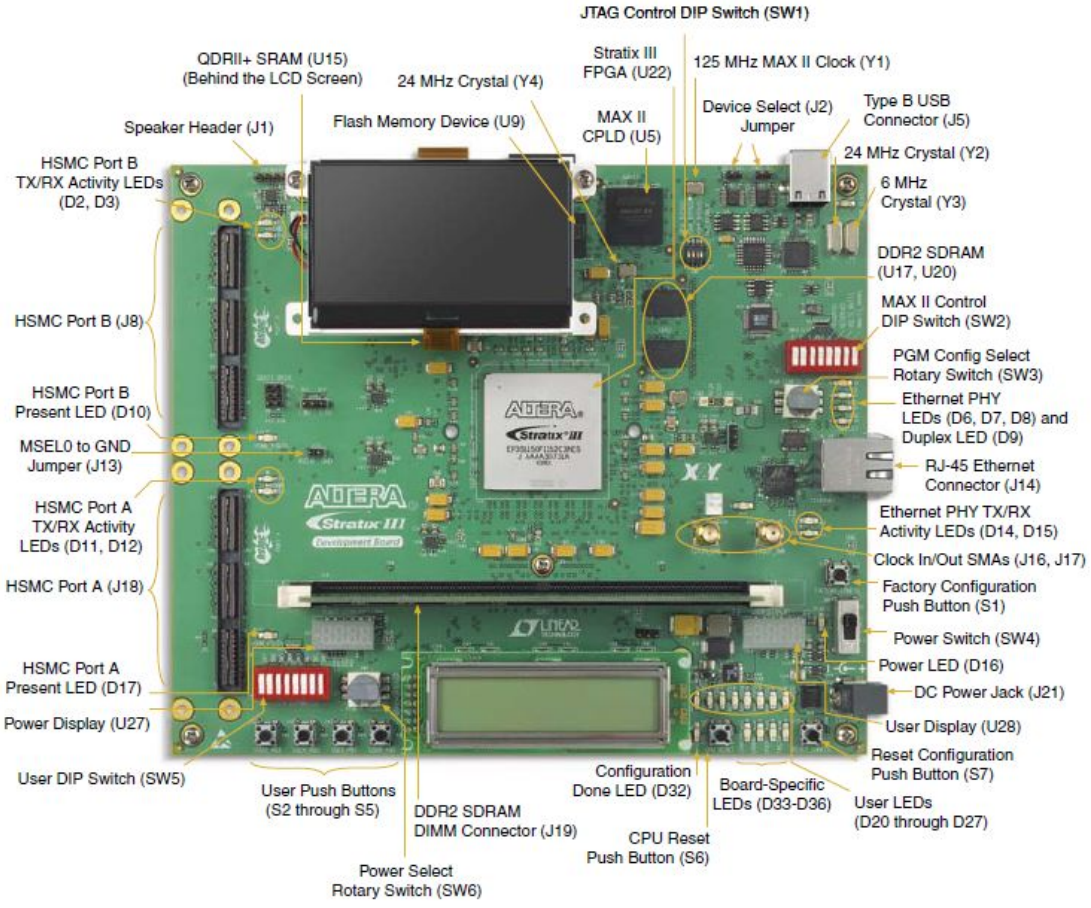

Figure 33 : Schéma de la carte de développement Stratix III

3.1 Le processeur embarqué NIOS

Le processeur embarqué NIOS est un processeur RISC (Reduced *Instruction Set Computer*), délivré sous forme d'un Soft Core (une IP disponible dans l'environnement SoPC Builder), dédié à la famille d'Altera. L'architecture du NIOS II peut être choisie selon les besoins de l'application :

- Possibilité de réduire ou (enrichir) les périphériques supportés par ce processeur.
- Ajout d'instructions spécialisées (au maximum 256 instructions)
- Pouvoir de supporter des composants faits par le concepteur

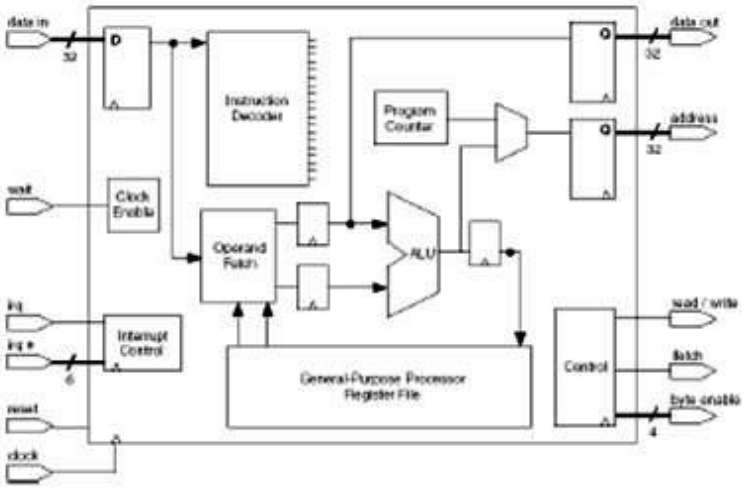

Figure 34: CPU NIOS

Ce processeur possède une largeur de bus de 32 bits, au maximum 6 niveaux de pipeline Figure 34. Sa fréquence de fonctionnement est de 100MHZ.

Le NIOS II utilise le bus Avalon Dans la suite nous nous intéressons à étudier le bus Avalon. Cette étude nous aidera au fur et à mesure, de notre travail à l'ajout des accélérateurs à notre architecture hardware. [Alt11]

3.2 Etude du bus Avalon

L'Avalon est un bus simple conçu pour connecter les processeurs embarqués et les périphériques dans un système sur un composant programmable (SOPC).

Les transactions de base du bus Avalon peuvent utiliser une largeur variable de données (8, 16, ou 32 bits) entre un périphérique maître et un périphérique esclave. Une fois un transfert est accompli, le bus est immédiatement disponible sur le prochain coup d'horloge pour une autre transaction entre la même paire maître-esclave ou entre des maîtres et des esclaves

indépendants. Le bus Avalon supporte également les dispositifs avancés de communication tels que les périphériques latents, les périphériques de transferts. Ces modes avancés de transfert permettent à plusieurs unités de données d'être transférées entre les périphériques pendant une simple transaction du bus.

3.2.1 Caractéristiques

La Figure 35 montre la décomposition du bus Avalon. Il comprend un décodeur d'adresse, un multiplexeur de données, un générateur de cycles d'attente et un contrôleur d'interruption.

Le bus Avalon supporte plusieurs maîtres du bus. Cette architecture multi-maîtres fournit une grande flexibilité dans la construction des systèmes SOPC et est favorable aux périphériques de large bande passante. Par exemple, un périphérique maître peut exécuter des transferts par accès direct en mémoire (DMA), sans exiger un processeur pour transférer des données à partir du périphérique à la mémoire [Alt02].

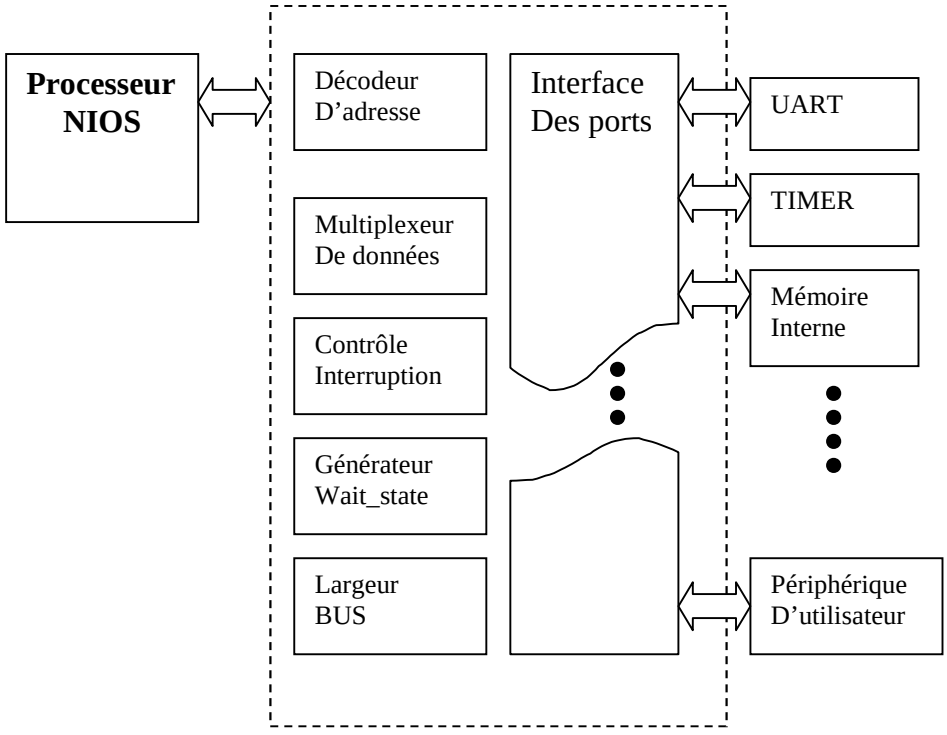

Figure 35: Architecture de bus Avalon

Cette interface est générée de façon transparente par le SOPC Builder ce qui permet la création rapide d'un système complet fonctionnel intégrant le NIOS avec ses périphériques.

L'ajout d'un périphérique sur le bus dépend de son type maître ou esclave.

Un périphérique maître est un périphérique qui peut initialiser les transferts. Il a au minimum un port master qui est connecté au bus Avalon. Il peut avoir un port esclave qui permet au périphérique de recevoir des transferts initialisés par les autres maîtres.

Un périphérique esclave est un périphérique qui accepte les transferts mais il ne peut pas les initialiser. Par exemple une mémoire, un UART.

3.2.2 Modes de transfert.

Le bus Avalon offre différents modes de transferts que nous détaillons par la suite :

- **Transfert simple**

Un transfert est une opération de lecture ou d'écriture qui peut prendre un cycle s'il s'agit d'un transfert sans « wait state » pour les périphériques synchrones ou plus d'un cycle pour les périphériques asynchrones qui nécessitent wait state.

- **Transfert avec latence**

Le bus Avalon supporte les transferts de données avec latence. Dans ce cas le maître peut initialiser un transfert de lecture, passe à un autre transfert et retourne pour recevoir les données plus tard.

- **Transfert en mode Streaming**

Ce mode de transfert permet de créer un canal entre le « streaming maître » et le « streaming esclave » pour exécuter successivement des transferts de données. Ce mode de transfert peut utiliser le « setup time » ou le « hold time » ou « waite states ». Ce mode de transmission maximise le débit entre le maître et l'esclave.

3.3 Flot de conception de l'environnement d'Altera

La Figure 36 présente les étapes nécessaires pour le prototypage d'un SoC en utilisant le kit de développement d'Altera.

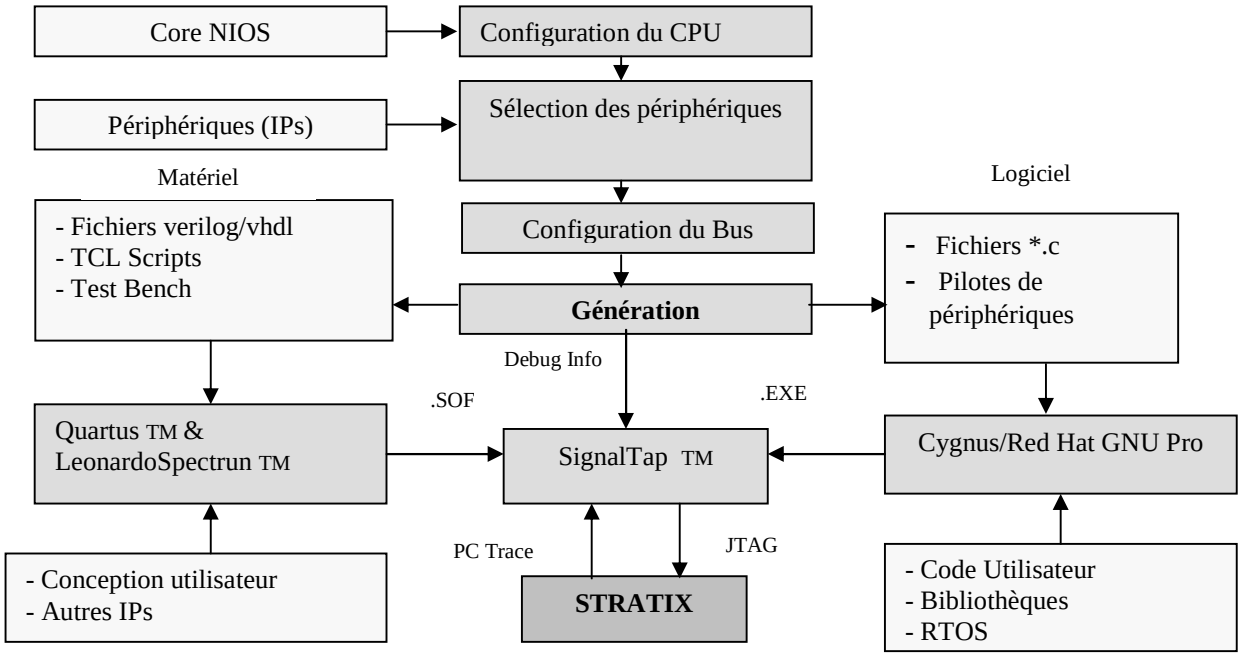

Figure 36: Flot de conception logiciel et matériel

4 Etude du système d'exploitation temps réel : MicroC/OS-II

MicroC/OS-II, conçu et mis à point par Jean J. Labrosse, est un noyau temps réel permettant d'effectuer une exécution de plusieurs tâches sur un microprocesseur ou un microcontrôleur [Lab02].

Ce noyau temps réel est maintenant disponible sur un grand nombre de processeurs, et il peut intégrer des protocoles standards comme TCP/IP (µC/IP) pour assurer une connectivité IP sur une liaison série. Les différentes versions de MicroC/OS-II sont portées sur des systèmes différents : Motorola famille 680x0, 68HC11/16, Power PC 860, Intel 80x86, Philips XA, etc.

4.1 Capacités et caractéristiques

Les caractéristiques essentielles de ce noyau sont les suivantes :

- Ouvert, code source disponible [Lab02],
- Portable, ROMable donc Encapsulable dans un produit,
- Fiable et robuste,
- Aux fonctionnalités ajustables,
- Multitâches et préemptif (l'Ordonnanceur de ce noyau contient seulement quatre lignes simples de code C),

- Interruptible : traitement des interruptions Par les ISR,
- Il permet de gérer 63 tâches où chaque degré de priorité correspond à une seule tâche, c'est-à-dire que deux tâches ne peuvent pas avoir le même degré de priorité.
- Changement de priorité des tâches (inversion et héritage de priorités).
- Fonction d'attente de tâche,
- Occupation optimale dans la mémoire: 2 Koctets taille du code,
- Création et gestion des sémaphores, des mutex, des mails box, des queues de messages et des drapeaux d'événements,
- Le temps d'exécution pour la plupart des services fournis par µC/OS-II est constant et « déterministe ».

4.2 Structure de MicroC/OS-II

Le système MicroC/OS-II peut être vu comme une bibliothèque de fonctions réparties sur des couches logicielles. Cette bibliothèque est liée avec l'application à développer. Ainsi, les services de MicroC/OS-II sont appelés depuis l'application comme de simples fonctions. Et comme le montre la Figure 37, le code source de ce noyau est divisé en deux sections : la première est indépendante du processeur et la seconde en est dépendante.

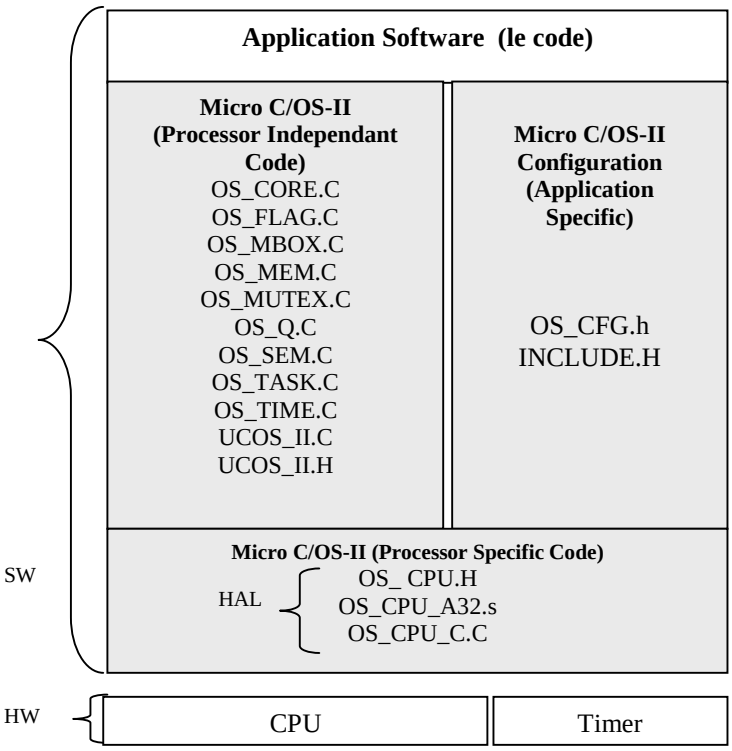

Figure 37: Structure de MicroC/OS-II

4.3 Fonctionnement de MicroC/OS-II

Lors de l'initialisation du programme MicroC/OS-II, les différents programmes de l'utilisateur sont considérés comme des tâches qui sont toutes créées pendant cette période d'initialisation. Le programmeur doit alors spécifier le point d'entrée de la tâche, l'emplacement des données pour cette tâche, l'adresse de la tête de la pile de la tâche et le degré de sa priorité. Ainsi, la tâche du plus haut degré de priorité est prête à l'exécution. Les tâches peuvent communiquer avec d'autres grâce aux sémaphores, boîtes aux lettres, files d'attentes et aux drapeaux d'événements, ou bien avec des périphériques grâce aux ISRs.

4.3.1 Création d'une tâche

Une telle tâche de l'application est constituée par une zone d'initialisation (une zone permettant d'initialiser les variables du programme utilisateur), une zone où l'utilisateur place le code de son programme et une instruction OSTimeDly(n) permettant de céder « n » coups d'horloge aux autres tâches. La création de la tâche se fait en appelant la routine suivante :

OSTaskCreate(AppTask1, (void *)0, (void *)&AppTask1Stk[255], 3);

AppTask1: point d'entrée du programme utilisateur (nom de l'étiquette).

(void *) 0 : adresse des données.

(void *)&AppTask1Stk [255] : adresse de la tête de la pile de la tâche.

3 : degré de priorité de la tâche.

4.3.2 Fonctions de base

Les principales routines de MicroC/OS-II sont [Lab02] :

- Initialisation de µCOSII : OSInit()
- Démarrage du multitâche : OSStart(),
- Gestion des tâches: OSTaskCreate, OSTaskCreateExt, OSTaskQuery, OSTaskDel, OSTaskDelReq, OSTaskChangePrio, OSTaskSuspend et OSTaskResume.
- Gestion d'interruption : OSIntEnter et OSIntExit.
- Gestion du temps: OSTimeDly, OSTimeDlyHMSM, OSTimeDlyResume, OSTimeSet, OSTimeGet et OSTimeTick.
- Gestion des sémaphores : OSSemCreate, OSSemAccept, OSSemPost, OSSemPend, OSSemDel et OSSemQuery.

- Gestion des mails box : OSMboxCreate, OSMboxAccept, OSMboxPost, OSMboxPend, OSMboxDel et OSMboxQuery.

- Gestion des files de communication : OSQCreate, OSQAccept, OSQPost, OSQPend, OSQQuery et OSQDel.

- Gestion des drapeaux d'événements : OSFlagCreate, OSFlagPost, OSFlagPend, OSFlagDel, OSFlagAccept et OSFlagQuery.

4.3.3 Communication inter tâches

Deux mécanismes élémentaires sont adoptés :

4.3.3.1 Partage de variable

Dans le cadre d'un partage de variable, le plus souvent, une tâche produit des données qui sont utilisées par une (ou plusieurs) autre(s) tâche(s). La coopération des tâches de l'application entre elles s'effectue à travers les messages, et les queues de messages. Alors que le sémaphore est employé pour gérer l'accès exclusif à la ressource partagée du système (mémoire vidéo). Le commun entre toute coopération est la présence de deux actions :

- Signalisation ; appelée aussi envoi (Posting).
- Attente ; appelée aussi réception (Pending).

Avec MicroC/OS, lors de la création d'un tel outil de communication, un ECB (Event Control Block) est créé pour maintenir l'état courant de cet outil. En fait, un ECB est une structure de données désignée pour décrire le type de l'événement en cours, ainsi que la liste des tâches en attente sur cet événement, avec d'autres informations nécessaires pour sa gestion.

Les actions de synchronisation mises au point sont les suivantes :

- Lors d'un PEND sur un sémaphore, un mutex, un message de la queue de messages ou un message d'un mail box, la fonction OS_EventTaskWait() est appelée pour retirer la tâche courante de la liste OSRdyGrp, et la mettre à l'état bloqué dans la liste OSEventGrp.
- Lors d'un POST sur l'un de ces outils, la fonction OS_EventTaskRdy() est appelée pour déterminer la prochaine tâche en attente qui aura la section critique. Et donc, celle-ci sera retirée de la liste OSEventGrp et mise à nouveau, active dans la liste OSRdyGrp.
- Lors d'un retour au PEND sur timeout, la fonction OS_EventTo() va retirer la tâche de la liste OSEventGrp, mais sans la mettre à nouveau active car elle l'est déjà. En effet, c'est

OS_TickTime() qui a la responsabilité de mettre OSTCBDly à jour et puis de rendre la tâche active lorsque ce dernier arrive à 0.

4.3.3.2 Synchronisation par événements

Dans ce cadre, les tâches sont synchronisées via les événements. En fait, si deux tâches ont besoin de se synchroniser avec l'apparition de multiples événements, typiquement, la seconde, afin de poursuivre son exécution, devra attendre que la première parvienne à un point donné. La synchronisation est maintenue à travers les drapeaux d'événements (Event Flag).

Les drapeaux d'évènements de µCOS-II sont constitués de deux éléments : une série de bits (8 ou 16 ou 32 bits) utilisés pour maintenir l'état courant des événements dans le groupe, et une liste de toutes les tâches en attente de la combinaison de ces bits (0 et 1) selon l'ordre désiré.

La gestion d'un événement se fait généralement au moyen des actions suivantes:

- Lors d'un PEND, la fonction OS_FlagBlock() est appelée pour maintenir le blocage de la tâche en attente sur l'apparition de l'événement. En fait, si les bits désirés dans le groupe de drapeaux d'événements (Event Flag Group) ne sont pas encore obtenus, cette tâche restera en attente indéfiniment jusqu'à la production de l'événement, ou bien l'expiration du timeout. Dans le cas de notre application, nous attribuons la valeur 0 au champ « timeout », étant donné que les tâches en attente sur un événement ne consomment aucune capacité de traitement, donc elles restent indéfiniment en attente jusqu'à ce que l'événement se produise.
- Lors d'un POST, la fonction OS_FlagTaskRdy() est appelée pour retirer la tâche bloquée de la liste d'attente (Waiting List of the Event Flag Group), et la remettre à nouveau à l'état prêt pour s'exécuter. Pour garantir qu'à tout moment le système puisse répondre aussi rapide que possible à un événement, cette tâche devrait commencer son exécution juste après la terminaison de la tâche produite. Pour ce faire, si la priorité associée à la tâche produite est « i », alors celle de la tâche consommatrice sera « i+1 » sachant que la valeur la plus petite correspond à la priorité la plus élevée.

5 Conclusion

La première partie de ce chapitre a été consacrée à la présentation de l'application de synthèse d'images 3D. On a présenté les différentes étapes du moteur graphique 3D qui

permet la création d'un objet 3D. On a présenté aussi les modifications qui ont été apportées à cette application pour qu'elle supporte la création de plusieurs objets. On a ajouté aussi la possibilité de changer le type d'algorithme d'ombrage et le nombre de polygones au cours du fonctionnement du système.

On a présenté dans la deuxième partie l'environnement de conception Hardware/Software adopté ainsi que le système d'exploitation temps réel MicroC_OS-II fourni avec l'environnement.

La deuxième partie de l'étape de validation consiste à mettre en place l'approche proposée à travers l'environnement et l'application choisis dans le but de mesurer ses performances et valoriser son apport.

Chapitre 5 : Expérimentation et validation

1 Introduction

Dans ce chapitre nous décrivons dans la première partie les différentes étapes qui ont conduit à la mise en œuvre concrète de cette application pour la validation de l'approche. Ces étapes regroupent la mise en place de la plateforme de prototypage et la caractérisation des configurations choisies pour le système. La deuxième partie de ce chapitre est consacrée à la présentation des résultats expérimentaux issus de l'exécution de notre approche sur un système physique. Nous clôturons ce chapitre par la présentation des résultats fournis avec les algorithmes d'optimisation implémentés.

2 Construction de la base des configurations

Nous allons suivre les étapes décrites dans le troisième chapitre pour la mise en place de la base des configurations. Nous commençons par l'étude de différents paramètres applicatifs qui influent sur la QoS du système. Ensuite nous appliquons l'approche de partitionnement hardware/software pour déterminer les parties candidates pour une implémentation matérielle pour notre application. Une fois les configurations implémentées et interfacées sur le bus nous passons à l'étape de caractérisation.

2.1 Configuration purement logicielles

Dans cette étude, nous utilisons un même attribut architectural : une implémentation purement logicielle (processeur + mémoire) sans avoir recours à des modules d'accélération hardware. Nous disposons d'un code C faisant du rendu 3D sur écran en utilisant les deux techniques d'ombrage flat et Gouraud (Dore, 2005) décrites dans le chapitre précédant. Ce code permet de charger un fichier d'extension .ASC décrivant les sommets et les polygones de l'objet 3D et de lui faire les traitements nécessaires pour le visualiser sur l'écran. Le cycle d'exécution de l'application de synthèse d'images 3d est représenté par la Figure 38.

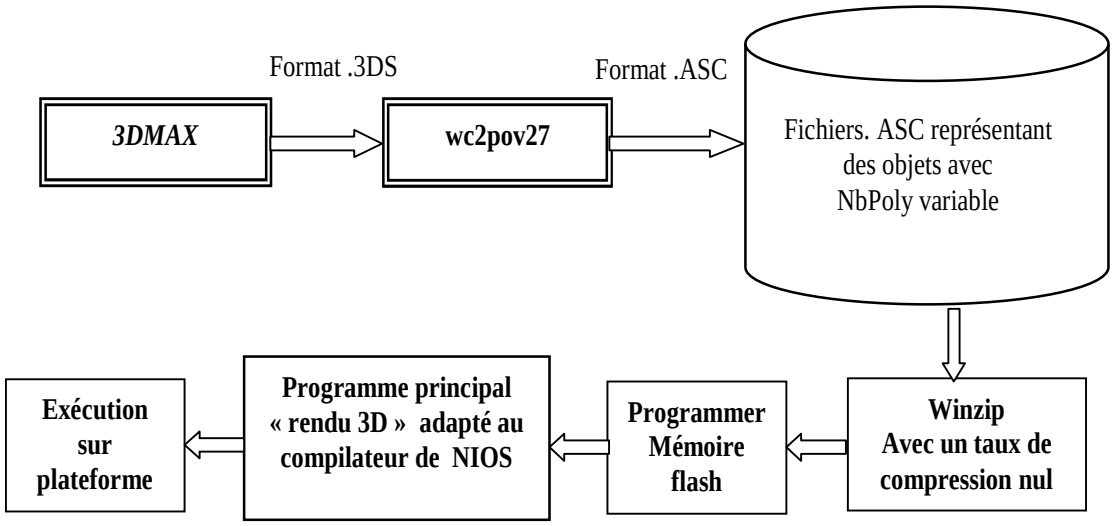

Figure 38: Procédure d'exécution de l'application synthèse d'images 3D

L'environnement de conception de la partie software le Nios-IDE, ne prend pas en considération le traitement des fichiers séquentiels. Afin de remédier à ce problème nous avons utilisé la mémoire flash pour que l'application puisse lire les coordonnées des polygones formant l'objet 3D. Cette méthode exige que les fichiers chargés soient être compressés par le logiciel Winzip avec un taux de compression nul (Figure 39).

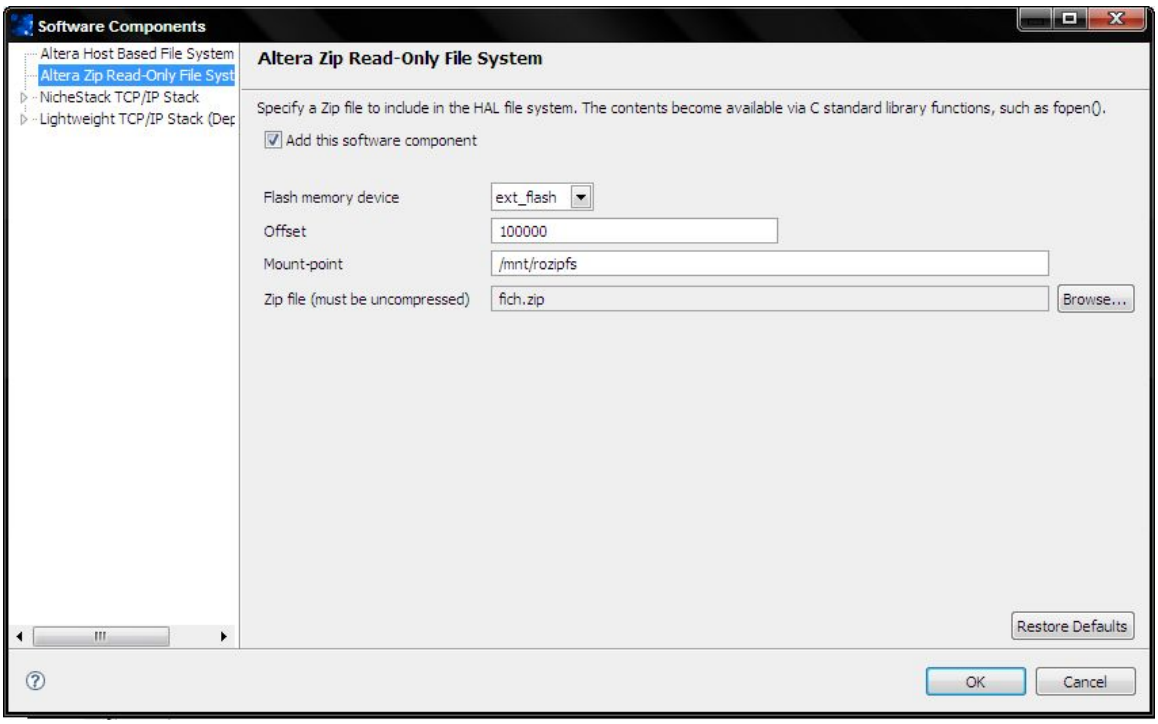

Figure 39 : Configuration des composants logiciels

2.1.1 Etude de l'effet des attributs applicatifs sur le temps d'exécution

Nous étudions dans l'impact du changement du nombre de polygones de l'objet 3D et le type d'algorithme d'ombrage sur le temps d'exécution de l'application. Cette étude est faite dans les deux cas d'ombrage : ombrage plat et ombrage de Gouraud. Pour la mesure du temps d'exécution on a utilisé le Timer. Le résultat est en nombre de tics ; pour le convertir en secondes il suffit de diviser la valeur par la fréquence de fonctionnement du système (100Mhz dans notre système).

La Figure 40 illustre le résultat de l'exécution de l'application 3D en modifiant le nombre de polygones et de l'algorithme d'ombrage.

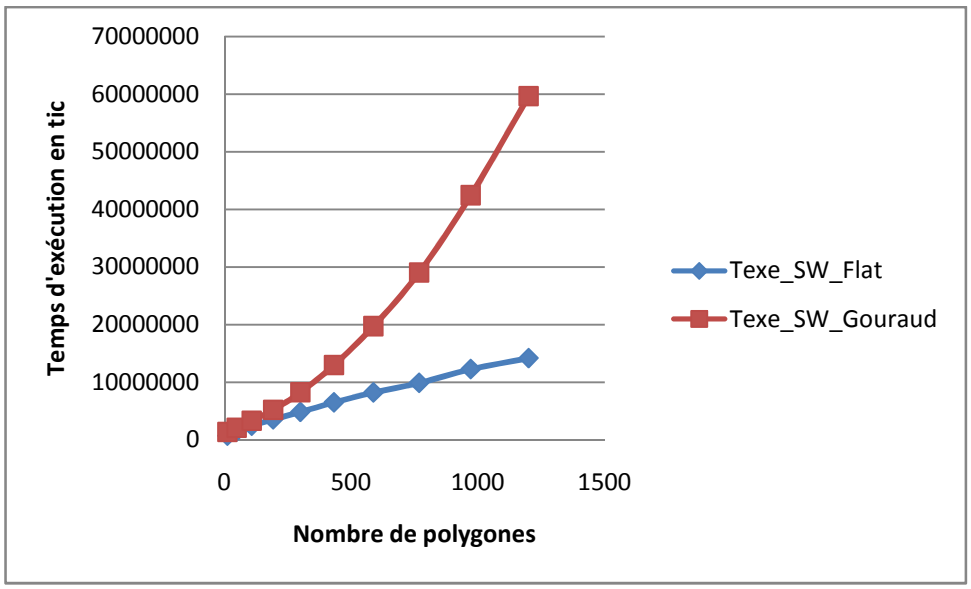

Figure 40: Impact du changement du nombre de polygones et de l'algorithme d'ombrage sur temps d'exécution de l'application

Nous constatons que le temps est d'autant plus élevé qu'on augmente le nombre de polygones. Même chose l'utilisation de l'algorithme d'ombrage de Gouraud demande un temps d'exécution plus élevé que l'algorithme de Lambert (Plat). Cette étude permet de fixer les premières caractéristiques des configurations à construire : temps d'exécution, type d'ombrage, nombre de polygones.

Nous détaillons dans la suite l'étude de l'effet des attributs architecturaux sur le temps d'exécution de l'application.

2.2 Conception des configurations mixtes HW/SW

Dans cette section, nous étendons notre étude au deuxième niveau d'adaptation qui est le niveau architectural et nous étudions l'influence des attributs architecturaux sur le temps d'exécution de l'application.

Nous détaillons dans un premier temps la mise en place des accélérateurs, ensuite nous étudions l'effet de l'utilisation de ces accélérateurs sur le temps d'exécution.

2.2.1 Conception d'accélérateurs dédiés à la synthèse 3D

Nous présentons dans cette section une implémentation matérielle de quelques modules de l'application de synthèse d'images 3D. Nous suivons la démarche développée au troisième chapitre pour la conception de ces accélérateurs afin d'obtenir des solutions architecturales en adéquation avec les caractéristiques algorithmiques de l'application.

2.2.1.1 Exploration de l'espace des solutions pour l'application synthèse 3D

Le but de cette tâche est d'identifier les fonctions candidates pour une implémentation hardware. Plus une implémentation comporte d'accélérateurs, plus elle sera performante (temps d'exécution plus faible) au dépend d'une architecture plus complexe et donc qui consomme plus d'énergie électrique.

Le nombre de solutions possibles correspondant à une application donnée est très élevé. Le choix des solutions retenues est très important. Ceci passe à travers une analyse de l'application synthèse 3D sous forme de profilage et d'analyse de complexité de chacune de ses fonctions. Ce qui permettra d'identifier les fonctions critiques qui nécessitent un traitement particulier.

Le profilage est obtenu grâce à un outil spécifique de NIOS IDE qui utilise un compteur matériel précis appelé « Performance Counter». Cet outil détermine le temps d'exécution des fonctions, le pourcentage ainsi que le nombre d'appels de chaque fonction, ce qui nous permettra d'identifier les fonctions critiques nécessitant des modules matériels pour leur exécution. Dans un deuxième temps, nous utilisons « Design Trotter » afin de mieux analyser les fonctions critiques.

2.2.1.1.1 Résultat de profilage sur Nios-II

Le Tableau 3 représente les résultats de profilage pour l'application de synthèse 3D décrite entièrement en langage C. Ce profilage représente le temps d'exécution pour chaque fonction ainsi que son nombre d'itération.

Tableau 3: Le résultat de profilage par l'outil « Performance Counter »

finish 3D synthesis code--Performance Counter Report--
Total Time: 437.069 seconds (21853450472 clock-cycles)

Section	%	Time (sec)	Time (clocks)	Occurrences
transformation	6.03	26.87581	443790261	360
calnormal	13.5	58.96694	2948347128	360
trie_face	5.92	25.87717	1293858382	360
translation	0.165	0.72616	36307753	360
load_ASC	0.013	0.05660	2829821	1
dessin_objet	77.6	340.79640	17039819772	360
Rotation	0.479	2.10350	105174764	360
dessin_poly	68.6	301.24407	15062203478	11489
normalise	2.23	9.78738	489369090	12960
vectoriel	2.22	9.71742	485871140	25920
Ghline	43.8	191.37397	9568698351	11489
Echelle	0.144	0.62963	31481286	360

D'après le Tableau 3, nous constatons que les fonctions «trie_face , dessin_objet, transformation, calnormal, Ghline et dessine poly» admettent un temps d'exécution important. Il est à noter que les valeurs présentées dans ce tableau correspondent à l'exécution de l'application dans boucle de 360 itérations. En fait, nous avons appliqué un mouvement de rotation à l'objet donc le nombre d'itérations correspond à un déplacement de 1°.

Bien entendu, la comparaison doit être faite entre les fonctions de l'application avec les mêmes paramètres applicatifs (nombre de polygones et algorithme d'ombrage).

D'après le résultat du profilage, on peut dire que ces fonctions sont candidates à une implémentation hardware. Passons maintenant aux résultats de l'outil « Design Trotter » qui nous permet de donner des estimations sur l'orientation des fonctions.

2.2.1.1.2 Les résultats de Design Trotter

Le Tableau 4 représente les résultats issus de l'utilisation Design Trotter. Des modifications ont été apportées au code de l'application pour tenir compte de la syntaxe utilisée par cet outil.

Tableau 4: Les résultats des métriques par « Design Trotter »

Fonction	Gamma	MOM	COM
transformation	3.8750	0.3939	0.000
Calnormal	3.3244	0.6527	0.0425
Normalise	3.2149	0.6522	0.0441
Vectoriel	3.2500	0.5600	0.000
trie des faces	1.8571	0.6666	0.1818
Dessin_objet	1.2429	0.8298	0.0038

D'après le Tableau 4, nous constatons que les fonctions transformation, calnormal, normalise, ont une valeur relativement élevée de gamma, donc elles ont un parallélisme moyen important. Elles sont donc candidates à une implémentation matérielle.

Combinant les résultats des deux méthodes, nous avons choisi les fonctions suivantes de la synthèse 3D :

- La fonction **ombrage** permettant de calculer la couleur en un point d'un objet 3D.
- La fonction **Normalisation** qui permet de normaliser un vecteur. Cette opération est nécessaire avant toute transformation (rotation, homothétie) sur un vecteur.
- La fonction **calcul normal** qui permet de calculer la normale à une face ou aux trois sommets d'un polygone, elle sert au calcul de l'intensité de la couleur.
- La fonction **transformation** elle permet d'appliquer des transformations sur l'objet
- Une fois les fonctions ont été choisies nous passons à leurs implémentations sous forme d'accélérateur en utilisant un langage de bas niveau.

2.2.1.2 Accélérateur dédié à l'ombrage Plat (ou de Lambert)

Le schéma bloc de l'accélérateur pour l'ombrage plat est donné sur la Figure 41. Il permet de calculer l'intensité de couleur d'un pixel en appliquant l'équation E19 d'illumination :

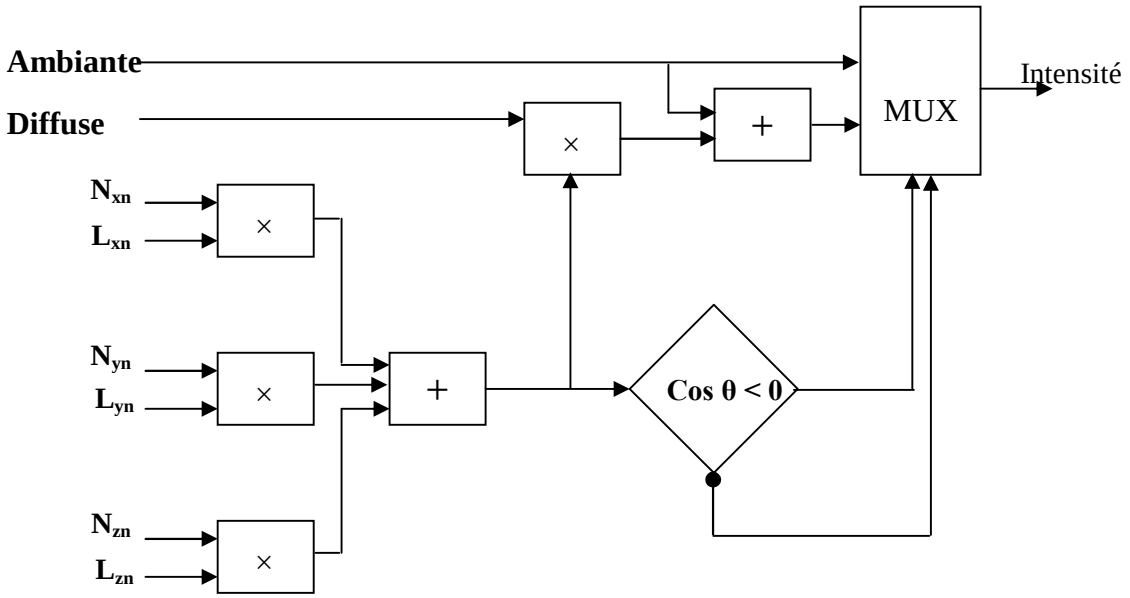

Figure 41: Module de calcul de l'équation d'illumination de Lambert

La valeur de la couleur d'un point dépend du signe de cos(θ).

- Si cos θ < 0 alors la sortie est égale à la valeur de la couleur ambiante.
- Si cos θ > 0 alors la sortie est calculée suivant l'équation d'illumination.

2.2.1.2.1 Accélérateur dédié à l'ombrage de Gouraud

Le module d'ombrage de Gouraud comme la Figure 42 est formé de :

- Un module pour le calcul des couleurs aux trois sommets du triangle élémentaire (a, b et c de la Figure 43), il est formé par trois accélérateurs d'ombrage plat.
- Deux modules interpolateurs de couleurs qui implémentent les équations E20 et E21.
- Un module pour le calcul d'incrément de couleur, il implémente l'équation E22.

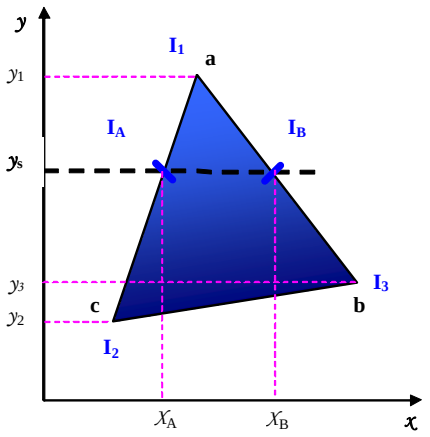

Figure 42: Ombrage Gouraud

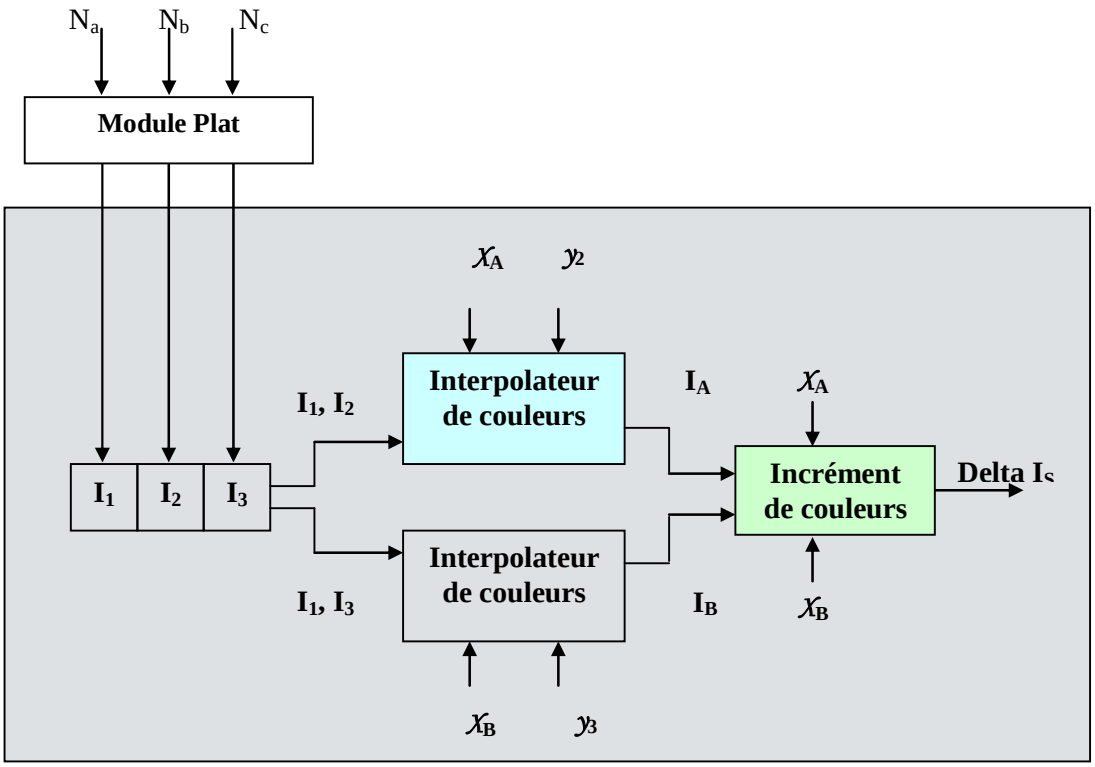

Figure 43: Le schéma bloc d'un module d'ombrage de Gouraud

Le 1er interpolateur de couleur calcule les couleurs des points de la ligne [ac], le second calcule les couleurs des points de la ligne [ab].

La Figure 44 représente le schéma bloc du module « interpolateur de couleurs » pour le calcul de la couleur au point I_A.

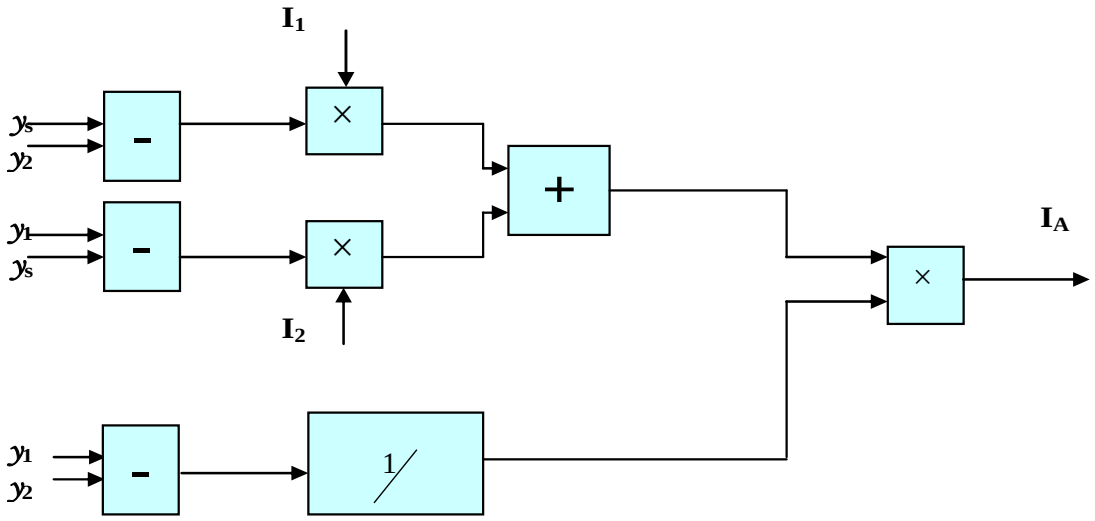

Figure 44: Interpolateur de couleurs

La Figure 45 représente le schéma bloc du circuit « incrément de couleurs ». Il s'agit de diviser la différence entre les couleurs de deux points par le nombre de pixels qui les séparent.

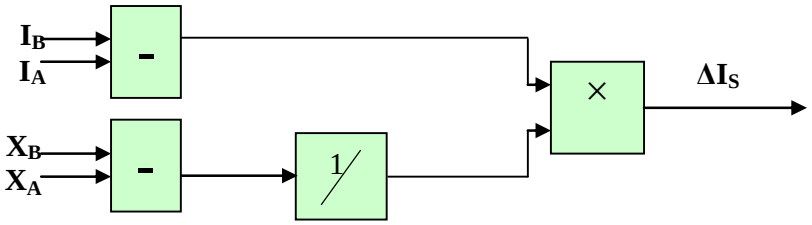

Figure 45: Incrément de couleurs

2.2.1.2.2 Accélérateur dédié au calcul de normale

En se référant au système de l'équation E11, le schéma bloc du circuit de la détermination de la normale d'une facette triangulaire est représenté par la Figure 46.

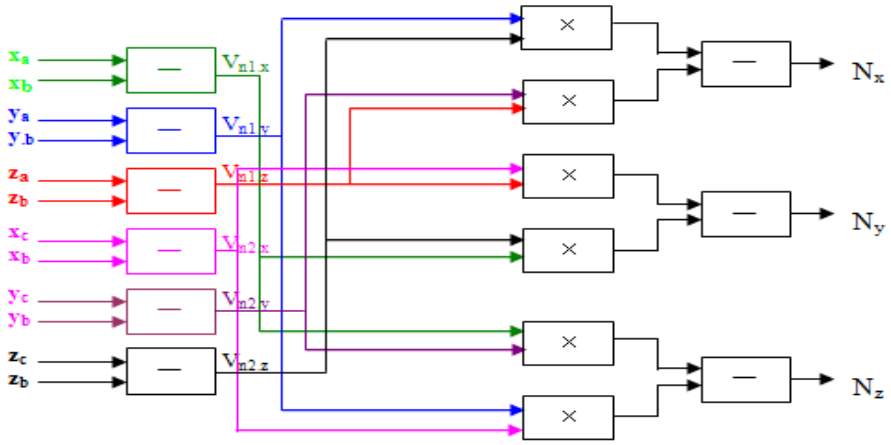

Figure 46: Schema bloc du circuit de calcul de la normale

2.2.1.2.3 Accélérateur pour la normalisation des vecteurs

Les transformations de repères et le calcul des couleurs effectuées dans la synthèse d'image 3D nécessitent l'utilisation de vecteurs unitaires (dont la norme est égale à 1).

La Figure 47 illustre le schéma bloc du module de normalisation

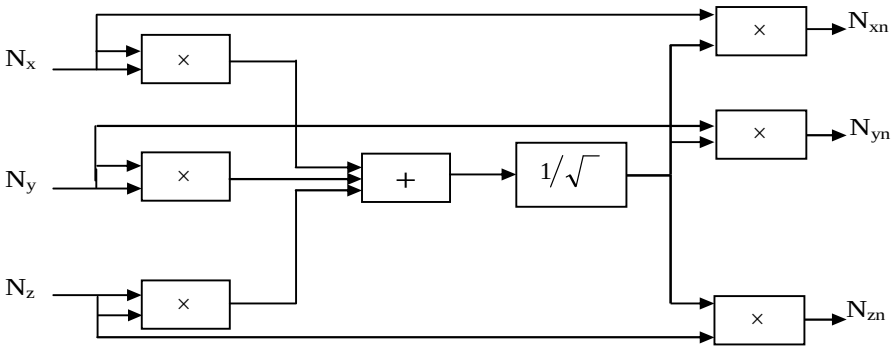

Figure 47: Schéma de bloc de la normalisation d'une normale

2.2.1.2.4 Accélérateur dédié à la transformation

La fonction « transformation » permet de passer les coordonnées locales de l'objet 3D dans le repère du monde (3D) puis dans le repère de la caméra 2D. Elle permet également d'animer l'objet 3D. Pour ce faire nous appliquons, aux objets de différentes matrices des translations, des rotations, et une normalisation des vecteurs.

Remarque

Les accélérateurs réalisés traitent des valeurs entières alors que dans notre application nous avons besoin de traiter des nombres réels. Comme solution, à chaque fois qu'on a besoin de traiter des nombres réels on multiplie la valeur par 10^n (n représente le nombre de chiffres après la virgule). Bien entendu on tient compte de ces multiplications dans le reste du traitement de notre accélérateur et au niveau des résultats fournis.

Par exemple pour le calcul de la racine carrée d'un nombre avec deux chiffres après la virgule.

Idée : $\sqrt{3}=1.73$ et $E(\sqrt{3\times 10000})=173$

Donc, dans la partie implantation en VHDL, on multiplie le nombre auquel on veut faire appliquer la racine carrée, par 100^2 pour obtenir une précision de deux chiffres après la virgule.

Or le bloc désiré est $(1/\sqrt{\ })$.

Alors, la structure finale du bloc doit être $(100^2/\sqrt{100^2\times nombre})$.

Par conséquent, les sorties de ce circuit sont des multiples de cent dont on va tenir compte dans les modules d'ombrages.

2.2.2 Implémentation des accélérateurs

Pour implémenter les accélérateurs hardwares conçus il est nécessaire de passer par un langage de programmation de bas niveau tels que VHDL, Verilog ou SystemC. L'environnement de conception d'Altera fournit un utilitaire qui permet de réaliser des blocs d'une manière graphique à travers l'assemblage d'opérateurs prédéfinis fournis avec l'environnement. Il donne aussi la possibilité d'ajouter des blocs écrits en langage de bas niveau. Après avoir réalisé le circuit adéquat schématiquement cet environnement peut générer directement le fichier correspondant en langage de bas niveau VHDL ou Verilog. La Figure 48 illustre l'accélérateur de Lambert réalisé avec cet utilitaire. Après avoir réalisé la totalité du circuit nous pouvons regrouper toutes les fonctions utilisées dans un seul circuit

sous forme de boite noire qui ne présente que les entrées/sorties du circuit et ce pour l'encapsulation du traitement. La Figure 49 présente le schéma bloc de même module.

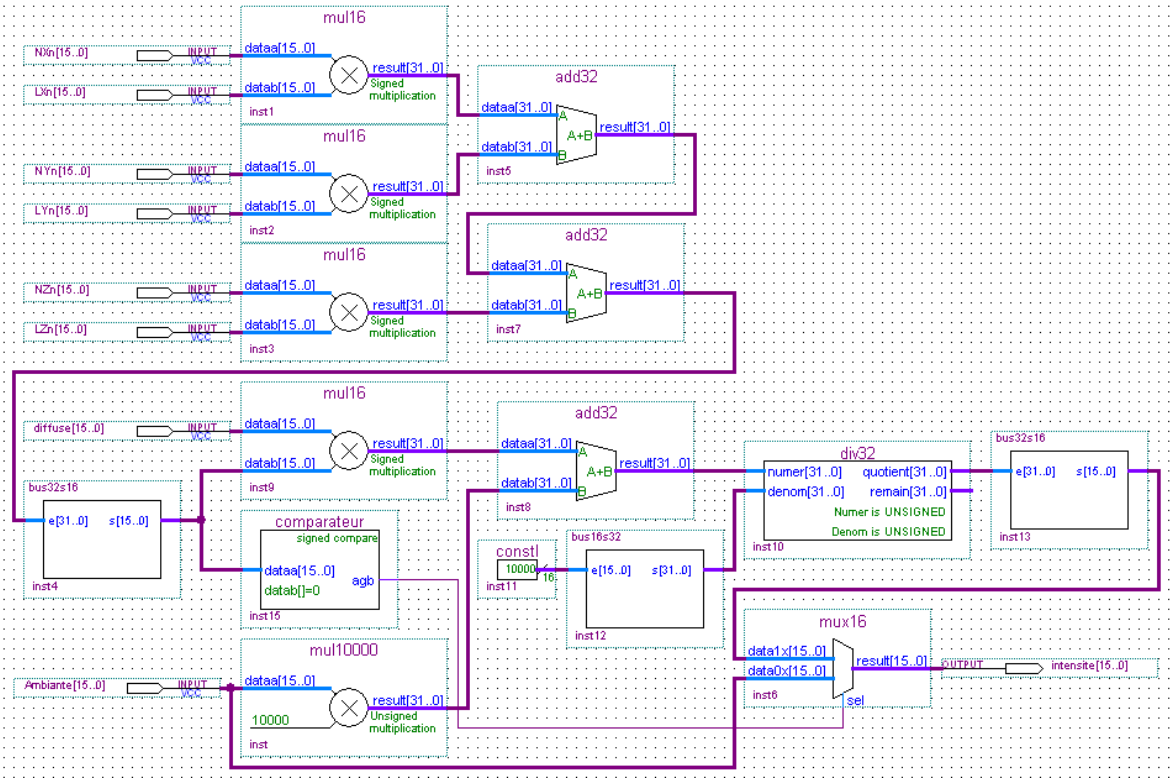

Figure 48:Circuit de Lambert

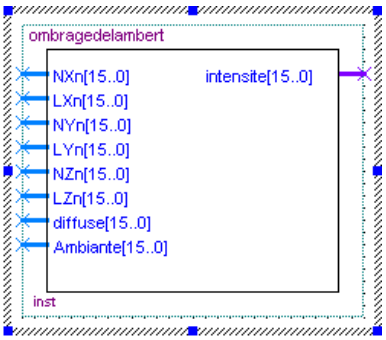

Figure 49: Schéma bloc de la fonction Lambert

2.2.2.1 Simulation des circuits implémentés

La simulation est une étape primordiale dans la conception des accélérateurs hardwares. Elle permet de vérifier le bon fonctionnement du composant réalisé. Cette étape se fait à travers des vecteurs de test « waveform vector» qui permettent d'envoyer des valeurs aux entrées de l'accélérateur et de récupérer les résultats fournis. Les figures (Figure 50 et Figure 51) présentent les étapes de validation par simulation du fonctionnement du circuit de Lambert.

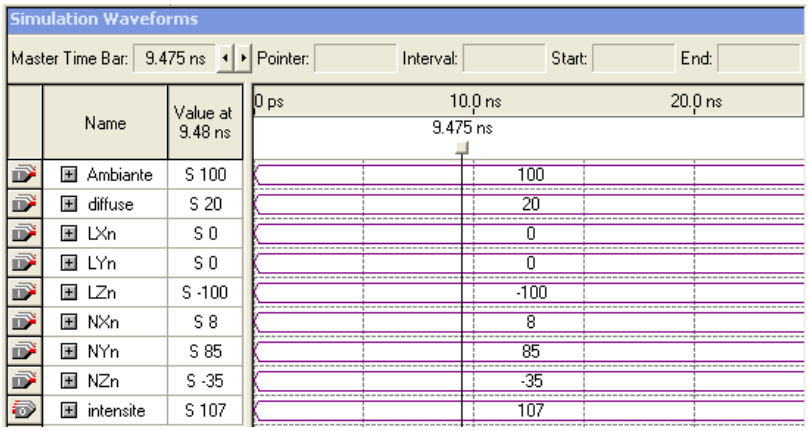

Figure 50:Résultat de simulation du circuit de Lambert avec θ négative

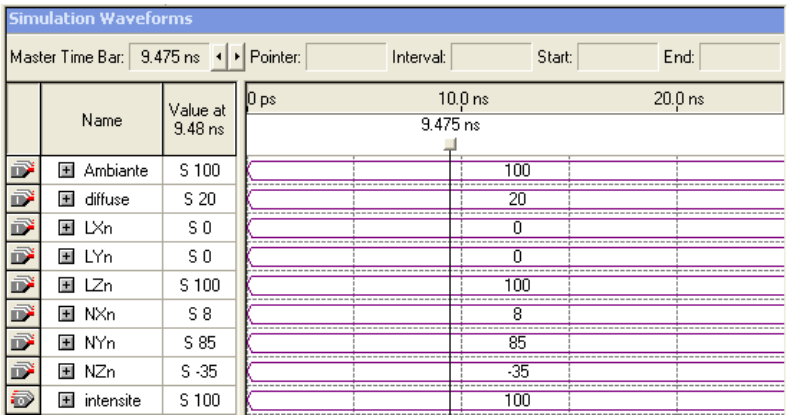

Figure 51:Résultat de simulation du circuit de Lambert avec θ positive

Les deux figures ci-dessus illustrent les deux cas que l'on peut rencontrer dans l'algorithme de Lambert.

- Le premier cas représente un cos θ >0, suite à une vérification de l'application numérique, la valeur de l'intensité trouvée est conforme.
- Le deuxième cas représente un cos θ <0, l'intensité trouvée est égale à la valeur de la lumière ambiante.

2.2.2.2 Intégration des accélérateurs au processeur NIOS

Nous détaillons dans cette section les différentes méthodes d'interfaçage des accélérateurs avec le processeur Nios-II.

2.2.2.2.1 L'interface de l'accélérateur esclave

Cette interface est constituée de trois modules. Le processeur NIOS-II avec ses mémoires, le bus Avalon, et l'accélérateur. Ce système se compose d'un processeur qui est le seul maître connecté au bus Avalon et un accélérateur matériel esclave. Les spécifications d'accélérateur

matériel sont limitées à un port de lecture et d'écriture. Ainsi, le maître « le processeur » doit alimenter le module matériel avec des données et attendre le résultat du port esclave.

2.2.2.2.2 L'interface de l'accélérateur maître

L'architecture se compose par le processeur NIOS II associé à une mémoire externe, une unité de traitement (accélérateur) et le bus avalon. Le principe de fonctionnement de l'interface maître peut être décrit ainsi : le processeur NIOS-II envoie l'adresse de base de la RAM à l'accélérateur qui lui aussi peut lire les données directement de la mémoire, effectue le calcul et réécrit le résultat dans la mémoire. Une fois achevée une IRQ est générée pour le processeur pour lui indiquer l'achèvement du traitement. Dans ce mode, nous évitons la sollicitation du processeur et les retards induits. Ainsi le processeur peut effectuer d'autre traitement en parallèle avec l'accélérateur.

2.2.2.2.2.1 Description de l'interface

Afin d'assurer la communication entre l'accélérateur maitre et les autres composants du système, les signaux de l'accélérateur doivent être conformes avec les signaux du bus qui va assurer cette communication décrite dans le chapitre 4. La Figure 52 décrit l'interface globale avec les signaux. L'entrée *« read_master »* et la sortie« *write_master* »sont sur 32 bits et synchronisées par l'horloge *du système*. Le signal *« read_master »* prend des entrées à partir d'une RAM extérieure Le signal *write_master* fournit le résultat qui sera écrit par la suite dans la RAM.

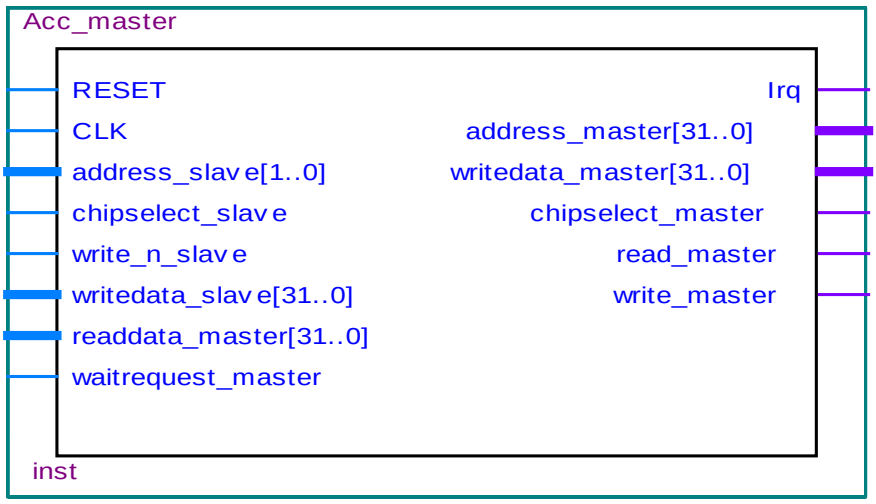

Figure 52: Structure de l'interface de l'accélérateur maître

2.2.2.2.2.2 Intégration des accélérateurs

Ayant les fichiers nécessaires à la création d'un périphérique, l'ajout d'un composant au SOPC Builder passe par une interface graphique qui gère toutes les étapes nécessaires.

La première étape, consiste à indiquer les fichiers de description matérielle du périphérique à ajouter, puis il va les analyser, voir s'ils sont synthétisables, définir l'entité de haut niveau et valider cette étape.

A la deuxième étape à partir des fichiers de description, l'assistant va déduire les différents signaux nécessaires à l'interfaçage de ce composant avec le bus Avalon (Figure 53) et donne la main à l'utilisateur pour spécifier leurs types. Cette étape nécessite la connaissance des contraintes imposées par le bus Avalon.

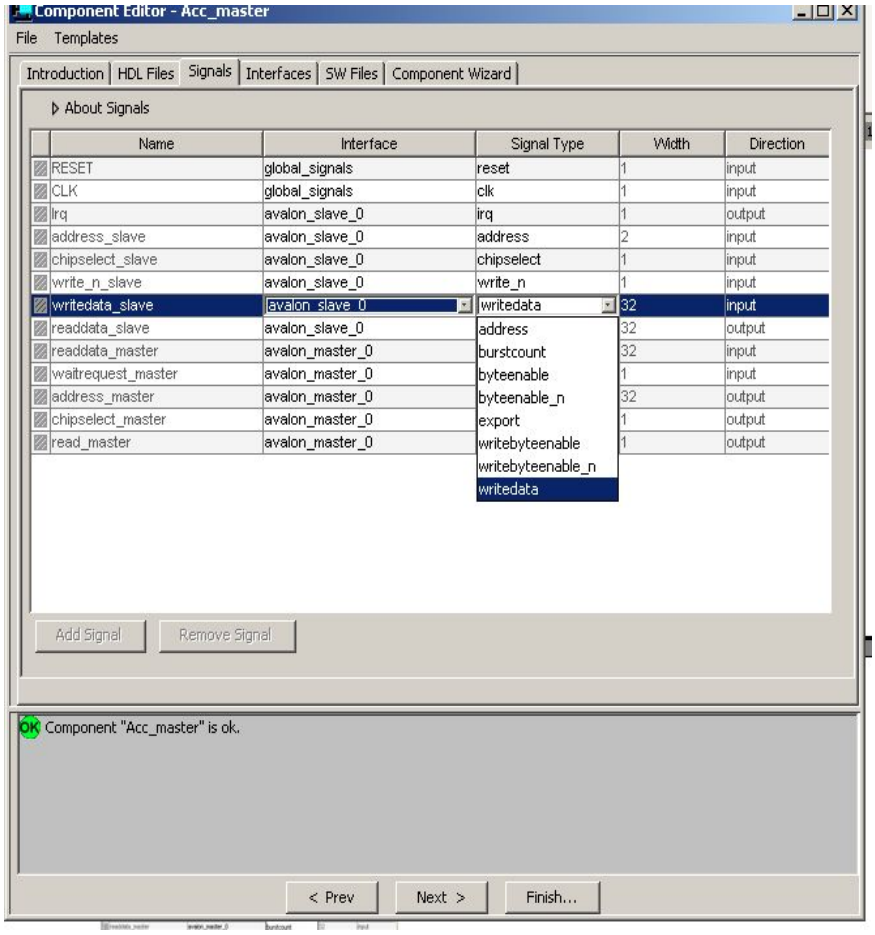

Figure 53: Mise au point des signaux Avalon

Une fois cette étape faite avec succès le composant sera ajouté à la bibliothèque des composants préconçus fournis avec l'environnement. Ainsi l'utilisateur peut ajouter autant d'instances de ce type d'accélérateur qu'il veut. Bien entendu ces accélérateurs fonctionnent d'une manière indépendante.

Deux fichiers seront créés automatiquement. Le premier fichier *class.ptf* c'est grâce à ce fichier, que le logiciel de développement pourra identifier et implanter l'interface dans le système. Il contient toutes les informations de connexion.

Le deuxième fichier *system.h* à travers lequel se fait la liaison entre les parties matérielles et logicielles. Ce fichier contient l'adresse des registres de notre bloc matériel (IP).

La dernière étape consiste à modifier le programme de l'application pour qu'il tienne en compte les accélérateurs existants dans l'architecture.

Dans cette section, nous avons détaillé les étapes suivies pour la conception des accélérateurs d'ombrage. La 1ère étape consiste à identifier les fonctions à implémenter en matériel à travers le profilage de l'application et l'outil « Design Trotter ». La deuxième étape concerne l'implémentation des modules hardwares en langage de bas niveau. La dernière étape se manifeste à l'interfaçage des modules implémentés sur le bus. Ainsi, nous disposons d'une multitude de choix architecturaux pour notre application.

2.3 Ajout des coprocesseurs hardwares

Par définition un coprocesseur est un processeur dédié à un traitement particulier. Il décharge le processeur principal des opérations qui lui sont dévouées. Le fait d'ajouter des coprocesseurs dans l'architecture c'est le fait d'ajouter des instructions spécialisées dans le jeu d'instructions du processeur. Contrairement aux accélérateurs hardwares qui peuvent avoir 0 ou n entrées/sorties un coprocesseur possède au maximum deux entrées et une sortie pour le résultat.

Dans notre travail, nous avons choisi d'ajouter les quatre opérations élémentaires de base sous forme de coprocesseur hardware avec virgule flottante. Il est à noter que le code VHDL de ces opérations avec virgule flottante est fourni avec l'environnement Figure 54.

Figure 54:Interface d'ajout d'accélérateur

Dans la section suivante, nous étudions l'effet de ces implémentations sur le temps d'exécution de l'application.

2.4 Etude de l'effet des paramètres architecturaux sur Texe

Nous détaillons dans cette section l'impact du changement de l'architecture hardware sur le temps d'exécution.

A noté que :

- Texe_SW_Flat/gouraud : représente le temps d'exécution de l'application sur le processeur.
- Texe_cop_flat/gouraud : représente le temps d'exécution de l'application en utilisant les quatre opérations de base sous forme de coprocesseurs.
- Acc_normal_flat/gouraud : représente le temps d'exécution de l'application en utilisant un accélérateur matériel qui calcule la normale à une face.
- Cop_normal_flat/gouraud : représente le temps d'exécution de l'application en utilisant les coprocesseurs et l'accélérateur de calcule de la normale

- **Cas de l'ombrage plat**

La Figure 55 montre la courbe Texe=f(NbPoly).

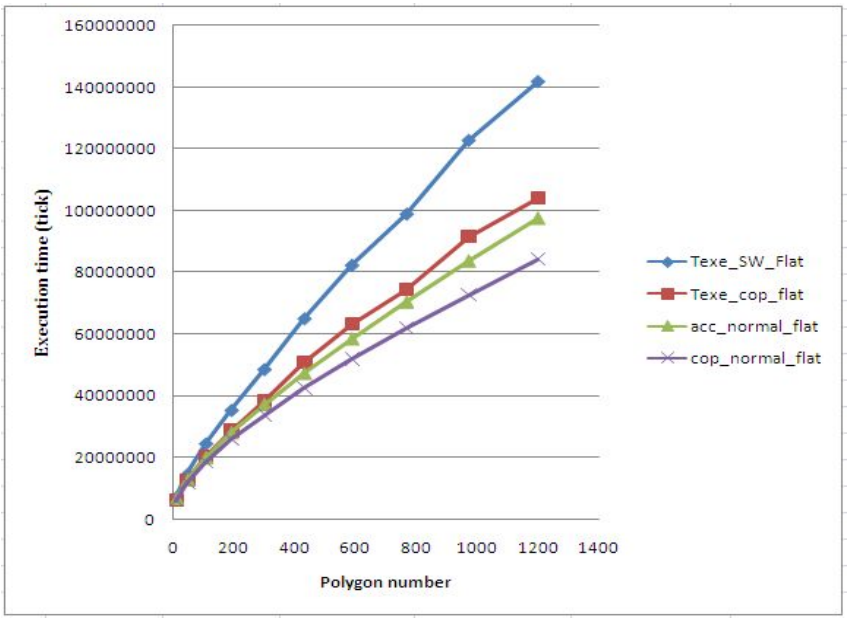

Figure 55: Impact du changement de l'architecture sur le temps d'exécution(Plat)

- **Cas de l'ombrage de Gouraud**

La Figure 56 montre la courbe Texe=f(NbPOly).

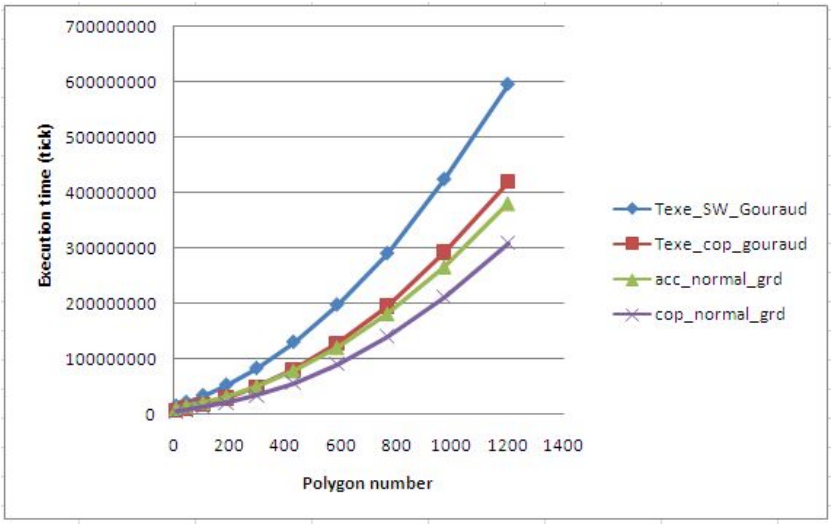

Figure 56: Impact du changement de l'architecture sur le temps d'exécution(Gouraud)

Nous remarquons d'après les figures ci-dessus que plus que nous ajoutons de composants hardwares plus qu'on exécute l'application plus rapidement mais sans doute ce gain est pénalisé par une augmentation de la consommation du système, thème du paragraphe suivant.

2.5 Mesure de la consommation

La quantification de la consommation des configurations passe par plusieurs étapes. Tout d'abord, afin de mesurer la consommation statique du système on a exécuté la tâche IDLE (tâche d'attente active) sur une architecture composée du processeur et des mémoires nécessaires sans utiliser aucun accélérateur HW (configuration SW) et on a pris la valeur maximale atteinte sur l'afficheur de la carte. Ensuite, dans le but de quantifier l'impacte de chaque accélérateur sur la consommation statique du système on ajoute l'accélérateur correspondant tout seul au processeur et on exécute la tâche IDLE. L'impact de cet accélérateur sera égal à la différence entre la valeur mesurée et la consommation statique. Enfin, l'application cible est exécutée sur chaque type d'architecture pour mesurer la consommation dite à l'exécution de l'application. Cette valeur est égale à la différence entre la valeur mesurée et la consommation de la tâche IDLE sur la même architecture.

Le Tableau 5 indique les mesures de consommation de quelques configurations.

Pour la mesure de la consommation on a utilisé la carte stratix3 qui intègre les circuits nécessaires et affiche la consommation d'énergie du noyau FPGA en mW au cours du fonctionnement du système.

Tableau 5: Caractérisation de la puissance des configurations

	Puissance de l'exécution de l'application 3D(mw)	Puissance de la tâche Idle (mw)	impact de l'accélérateur HW (mw)	Puissance dite à l'exécution de l'application (mw)
CPU + MEM	437	422	0	15
Acc_normale	440	431	9	9
Copro (+,-,*,/)	459	444	22	15
Corpo+ Acc_normale	472	454	32	18
scalaire	440	428	6	12
vectoriel	444	425	3	19
normalisation	450	432	10	18
transformation	448	437	15	11

Sachant que :

- CPU + MEM : représente l'architecture standard qui ne contient pas d'accélérateur matériel spécifique pour l'application de synthèse d'images 3D
- Acc_normale : représente l'accélérateur matériel de calcule de la normale
- Copro(+,-,*,/) : représente l'utilisation des quatre opérations de base sous forme de coprocesseurs.
- Copro + Acc_normale : représente l'utilisation des coprocesseurs et de l'accélérateur normale.
- Scalaire, vectoriel, normalisation et transformation : correspondent à l'accélérateur de calcul respectivement du produit scalaire, du produit vectoriel, de la normalisation d'un vecteur et du résultat de fonction de transformation.

Ainsi, en utilisant le tableau ci-dessus on peut calculer la consommation globale du système suivant l'architecture hardware utilisée.

$$\text{Consommation (mj)} = (\text{puissance\_idle\_task} + \sum \text{impact\_HW\_acc}) * \text{hyp} + \sum \text{impact\_app} * \text{Texe}$$

La caractérisation complète des configurations nécessite également la mise en place d'un modèle capable de caractériser la QoS d'une configuration donnée.

2.6 Mise en place du modèle de QoS

Dans cette approche il est nécessaire que chaque type d'application exécuté dans le système possède son propre modèle de QoS. Vu qu'on a travaillé avec l'application de synthèse d'images 3D on est amené à mettre en place un modèle permettant de quantifier la qualité d'un objet 3D affiché à l'utilisateur. Bien entendu la qualité d'une image 3D dépend de plusieurs paramètres tels que le nombre de polygones représentants l'objet, le type de

l'algorithme d'ombrage utilisé, la taille occupée sur l'écran, la vitesse d'animation etc. Puisque dans cette approche on est limité à la variation du nombre de polygones et le type d'algorithme d'ombrage, le modèle adopté ne tient compte que de ces deux facteurs.

Dans ce travail on a utilisé le modèle de QoS proposé dans [Pan05]. Ce modèle est représenté par l'équation E23 il dépend uniquement du nombre de polygones constituant l'objet.

$$QoS_1(NbPoly = 100 * \frac{,1834}{,1834\ \ 1{,}81\ \ *(1\ \ \ ,\ \ ^2} \qquad \text{E23}$$

Ce modèle a été amélioré dans [Ben07] pour qu'il tienne en compte du type d'algorithme d'ombrage utilisé. D'où la nouvelle formule de calcul de la valeur de QoS :

$QoS=\alpha(ombrage)*QoS_1(Nbpoly)$

La valeur de α(ombrage) est calculée par l'équation E24.

$$\alpha(ombrage = \frac{1}{}\exp\left(\frac{pp}{Npi\ \ e\_image}\right. \qquad \text{E24}$$

où :

- Npixel_image : nombre total de pixels de l'image
- NPP : nombre moyen de pixels par polygone
- β : coefficient qui caractérise l'apport de l'algorithme Gouraud par rapport à celui du plat.

Ainsi la formule de QoS de l'algorithme de Gouraud est donnée par l'équation E 25

$$QoS = 100 * \frac{1}{}\exp\ \frac{pp}{Npi\ \ e_{image}}\ \ * \frac{,1834}{,1834\ \ 1{,}81\ \ *(1\ \ \ ,\ \ ^2} \qquad \text{E25}$$

La Figure 57 permet de déterminer la valeur de QoS en connaissant le nombre de polygones et le type d'algorithme d'ombrage utilisé avec une valeur de β=500.

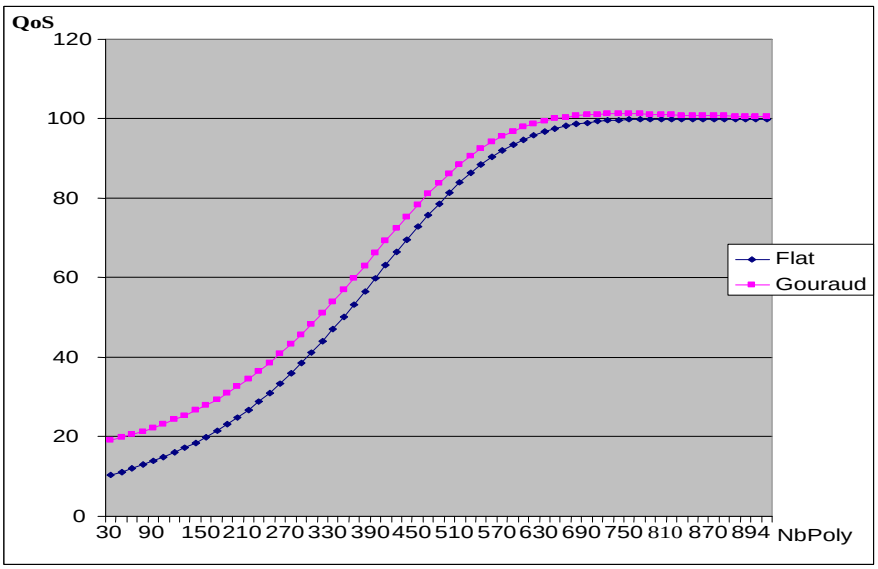

Figure 57 : Modèle de QoS

2.7 Configurations retenues

Suite aux différentes mesures déjà présentées on a pu mettre en place la base des configurations qui contient les informations suivantes (la période, l'échéance, le temps d'exécution, le niveau de qualité de service, le type d'algorithme d'ombrage utilisé, la puissance statique de l'architecture, l'impact de l'exécution de l'application et la référence de l'architecture hardware).

Le Tableau 6 représente un exemple de configuration pour deux objets, le cube et le cylindre. Pour des raisons de simplicité et afin qu'on puisse vérifier les résultats fournis par l'approche on s'est limité à dix configurations pour chaque objet.

Chaque configuration contient les informations suivantes :

- Period : période de l'application
- Deadline : échéance
- Texe : temps d'exécution au pire cas
- QoS level : niveau de la qualité de service fourni à l'utilisateur
- Shade_algo : l'algorithme d'ombrage utilisé
- Puis_conf : la puissance consommée par l'accélérateur
- Puis_app : puissance dite à l'exécution de l'application
- Ref_HW : le numéro de l'architecture (CPU+ ensemble d'accélérateurs). Dans notre cas 1 représente une configuration purement software et 2 c'est une architecture qui contient l'accélérateur de calcul de la normale, produit scalaire et vectoriel. Il est à noté qu'on n'a pas utilisé les coprocesseurs déjà implémentés à cause d'un problème lié à la plateforme qui ne permet pas d'activer et désactiver les coprocesseurs pour une tâche donnée. Donc si on active les coprocesseurs pour une application ils seront activés pour toutes les applications en cours d'exécution.

Tableau 6: Exemple de configurations retenues

ident_obj	Cube01	Cube02	Cube02	Cube02	Cube03	Cube03	Cube03	Cube04	Cube04	Cube5
period (s)	2,5	2,5	2,5	2,5	2,5	2,5	2,5	2,5	2,5	2,5
deadline (s)	2,4	2,4	2,4	2,4	2,4	2,4	2,4	2,4	2,4	2,4
Texe (s)	0.08	0.18	0.25	0.19	0.3	0.42	0.3	0.43	0.68	0.6
QoS level	3	10	22	22	14	28	28	26	36	45
shade_algo	Flat	Flat	Gouraud	Gouraud	Flat	Gouraud	Gouraud	Flat	Gouraud	Flat
Puis_conf (mw)	0	0	0	9	0	0	9	0	0	0
Puis_app (mw)	15	15	15	9	15	15	9	15	15	15
Ref_HW	1	1	1	2	1	1	2	1	1	1
ident_obj	Cyl01	Cyl05	Cyl05	Cyl05	Cyl07	Cyl09	Cyl09	Cyl11	Cyl11	Cyl11
period (s)	2,5	2,5	2,5	2,5	2,5	2,5	2,5	2,5	2,5	2,5
deadline (s)	2,4	2,4	2,4	2,4	2,4	2,4	2,4	2,4	2,4	2,4
Texe (s)	0.18	0.28	0.4	0.24	0.69	0.52	0.8	0.6	0.95	0.75
QoS level	10	15	24	24	36	30	40	36	46	46
shade_algo	Flat	Flat	Gouraud	Gouraud	Gouraud	Flat	Gouraud	Flat	Gouraud	Gouraud
Puis_conf (mw)	0	0	0	9	0	0	0	0	0	9
Puis_app (mw)	15	15	15	9	15	15	15	15	15	9
Ref_HW	1	1	1	2	1	1	1	1	1	2

3 Mise en place d'un système d'exploitation temps réel

Suite à l'étude faite dans le chapitre trois nous avons choisi de travailler avec un système d'exploitation qui utilise un ordonnanceur de type EDF. Donc le problème consiste à trouver un système d'exploitation qui répond à nos besoins en premier lieu, et qui peut être « porté » sur notre plateforme de travail.

Puisque nous travaillons avec l'environnement d'Altera qui comporte le système d'exploitation temps réel MicroC_OS-II, deux solutions ont été envisageables. La première consiste à chercher un RTOS avec un ordonnanceur EDF et ensuite à se lancer dans la complexe tâche de modification de la couche HAL (hardware abstraction Layer) pour le configurer et le porter sur notre plateforme de travail. La deuxième solution, était de modifier l'ordonnanceur du MicroC_OS-II puisque son code source est ouvert et disponible avec toute la documentation.

Nous avons choisi dans notre travail la deuxième solution. Pour ceci, nous détaillerons les étapes nécessaires qui ont conduit à la mise en place d'une part de la notion de périodicité des tâches et d'autre part de l'implémentation de l'ordonnanceur EDF.

3.1 Description de l'EDF (Earliest Deadline First)

Earliest deadline first est un algorithme d'ordonnancement préemptif utilisé dans les systèmes temps réel. Il appartient à la classe des algorithmes à priorité dynamique, où une instance de tâche change de priorité durant son exécution. Cette politique d'ordonnancement permet d'exécuter les instances dans l'ordre de leur urgence où le degré d'urgence est mesuré par la proximité de leur échéance. Cela implique qu'une instance ne peut utiliser la ressource que si toutes les instances d'échéances plus petites ont terminé leur exécution ou ne sont pas encore actives.

Dans le cadre de l'ordonnancement préemptif des tâches, EDF a le très grand avantage d'être optimal vis-à-vis de la faisabilité du système dans des contextes variés, c'est à dire que tout ensemble de tâches faisable sous une politique autre qu'EDF sera nécessairement faisable sous EDF.

3.2 Implémentation de l'EDF sous µC_OS-II

Notre politique EDF a été introduite dans les fonctions internes de µC_OS-II de manière à pouvoir commuter entre elles et la politique déjà existante (à priorité fixe).

Deux étapes étaient nécessaires pour l'implémenter : l'implémentation de la périodicité des tâches, ensuite, la gestion des échéances.

3.2.1 Gestion de la périodicité

Les tâches périodiques lancent leurs instances dans des intervalles de temps réguliers appelés périodes (se réveillent toutes les p unités de temps). Une tâche périodique est alors caractérisée par (Figure 58) :

- sa période p,
- son échéance d. L'échéance est le temps séparant l'instant de mise à l'état prêt de l'instance de tâche et celui au bout duquel cette instance doit terminer son exécution,
- son temps d'exécution c,
- son temps d'exécution au pire cas wcet.

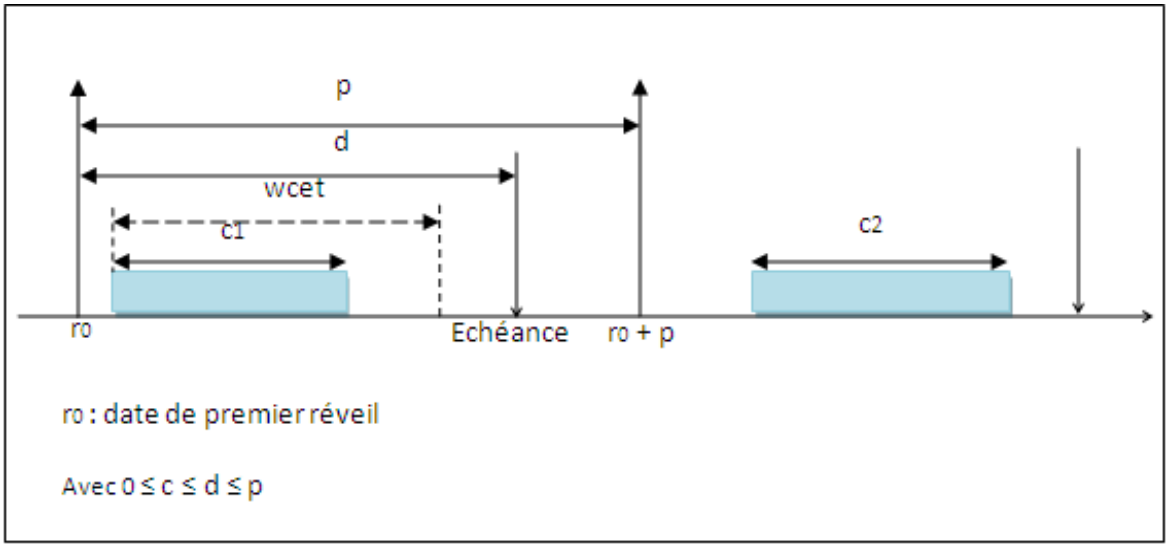

Figure 58 : Paramètres des tâches périodiques

Comme il est illustré dans la Figure 58, la tâche périodique doit s'exécuter une fois par période. Elle peut commencer son exécution à divers instants et avoir différents temps d'exécution, mais elle doit terminer avant son échéance. Les paramètres relatifs à cette tâche doivent être définis tout en respectant l'ordre suivant : $0 \leq c \leq wcet \leq d \leq p$.

Il est vrai que µC/OS-II ne gère pas la périodicité des tâches, mais il offre des services de gestion du temps fonctionnant selon l'horloge système qui déclenche une interruption périodique à une fréquence fixée au départ. En effet, lorsqu'une tâche se suspend, elle spécifie un délai en nombre de tics d'horloge pour s'endormir. A chaque tic, la routine d'interruption de l'horloge système OSTimeTick() s'exécute. Elle doit vérifier les délais de toutes les tâches

pour en détecter ceux qui sont expirés. Dans ce cas, les tâches concernées seront remises à l'état prêt. A la fin de la routine, l'ordonnanceur est appelé. Il décide qu'une commutation de contexte s'avère nécessaire s'il y a une tâche, parmi celles remises à l'état prêt, qui a une priorité plus élevée que celle de la tâche ayant été interrompue. Dans ce cas, l'ordonnanceur interrompt cette dernière et retourne à la tâche de plus haute priorité. La Figure 59 illustre l'ensemble de ces étapes.

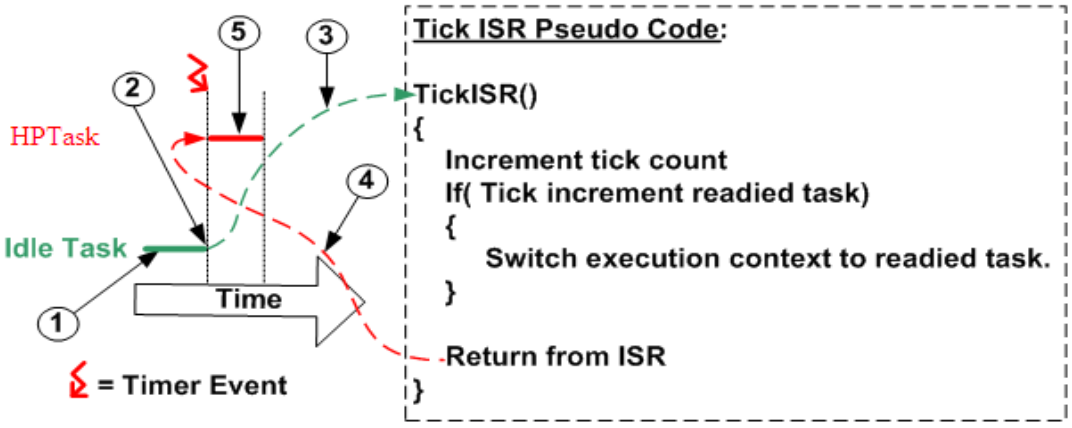

Figure 59:Gestion du temps

Le temps s'écoule de gauche à droite. En (1) la tâche idle est en exécution. L'interruption d'horloge arrive en (2) et le contrôle du CPU passe à la routine correspondante (3). Cette routine met la tâche HPTask à l'état prêt. Puisque cette dernière à une priorité plus haute que la tâche idle, l'ISR fait appel à une commutation vers le contexte de HPTask à la sortie de l'ISR (4) et la tâche commence à s'exécuter (5).

Afin d'intégrer la gestion de la périodicité dans le noyau de µC/OSII, l'idée était de profiter des services de gestion du temps qu'il offre et de la routine d'interruption d'horloge qui permettent de manipuler les tâches à des instants bien spécifiques puis obtenir des informations sur leurs contraintes temporelles.

L'idée était d'étendre la structure du contexte des tâches définies par µC/OS-II pour supporter la gestion de la périodicité. On a ajouté une nouvelle structure de données dans la zone de données utilisateur additionnelle du bloc de contrôle des tâches (TCB, Task Control Block). La Figure 60 illustre la structure étendue de ce bloc.

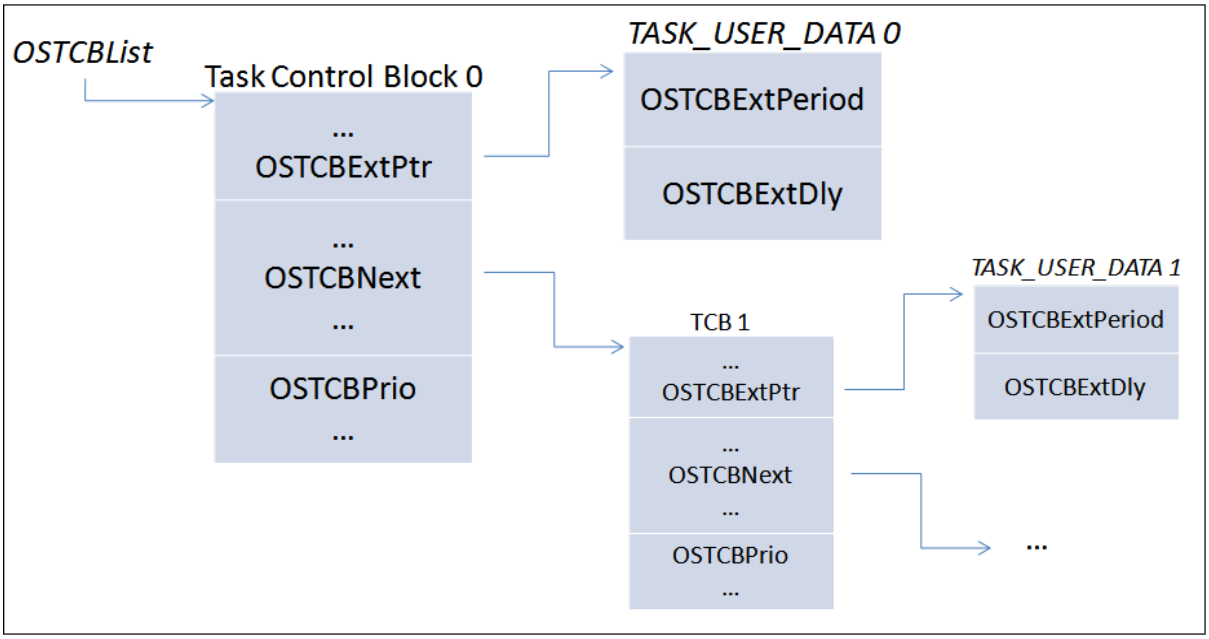

Figure 60: Structure étendue du TCB pour le support de la périodicité des tâches

La zone d'extension du TCB, *TASK_USER_DATA*, est pointée par *OSTCBExtPtr***.** L'utilisation de ce pointeur permet d'étendre le TCB tout en réduisant les modifications apportées aux fonctions internes du noyau existant.

OSTCBExtPeriod est la période de la tâche qui doit être définie en offline.

OSTCBExtDly est le compteur de délai associé à la période. Il est chargé par la valeur de période à la création de la tâche et il est décrémenté à chaque tic d'horloge. Lorsque la tâche termine son travail, elle se met en attente de sa prochaine période en utilisant la fonction OSTimeDly(). Dès que le délai de période expire, il est rechargé automatiquement et la remise à l'état prêt de la tâche est forcée par la fonction *OSTimeDlyResume()*. La gestion de la période est implémentée dans la fonction interne *OSTimeTick()* qui est déclenchée périodiquement par l'interruption de l'horloge système (System Ticker ISR).

Les TCBs sont placés dans une liste chaînée pointée par la tête de liste *OSTCBList* et triée par priorité.

La création d'une tâche avec cette structure étendue n'est possible qu'avec la fonction *OSTaskCreateExt()* et que si le flag *OS_TASK_CREATE_EXT_EN* est activé.

3.2.2 Mise en œuvre de EDF

Après avoir implémenté la périodicité des tâches, on a ajouté une autre structure de données dans la même zone d'extension permettant la gestion d'échéance. La Figure 61 montre cette structure de deadline qui est placée dans une liste chainée.

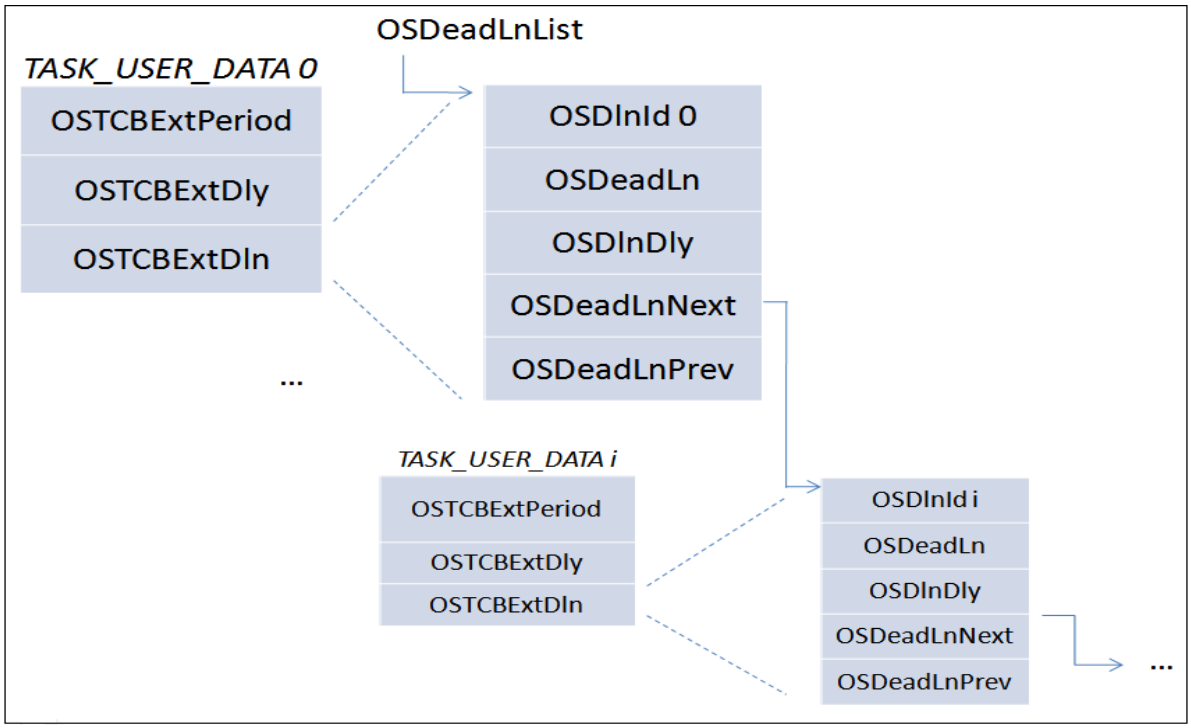

Figure 61: Structure du deadline dans la zone d'extension du TCB pour le support d'EDF

Le troisième champ de la zone d'extension, OSTCBExtDln, est de type structure deadline. Il est formé de 5 champs.

OSDeadLn stocke la valeur d'échéance de la tâche qui doit être inférieure ou égale à la période.

Tout comme pour la période, *OSDlnDly* est le compteur de délai associé à l'échéance. Il est chargé par la valeur de celle-ci à la création de la tâche et il est décrémenté à chaque tic d'horloge. La recharge d'*OSDlnDly* est faite au moment d'expiration de la période.

Comme l'algorithme EDF attribue la plus haute priorité à la tâche d'échéance la plus proche, il était nécessaire de rechercher le Min des échéances à chaque évènement de commutation. Deux solutions ont été envisagées :

- Parcourir toutes les échéances à chaque évènement de commutation pour en déterminer la plus petite.
- Arranger les structures des deadlines dans une liste chainée triée par ordre croissant d'échéance. La plus petite échéance sera en tête de liste.

On a opté pour la 2ème solution vu qu'elle est plus optimisée du point de vue temps d'exécution. En effet, la mise à jour de la liste triée des échéances ne sera réalisée que lorsqu'une nouvelle tâche prête à rouler arrive ou un délai d'une période expire. D'où le

nombre de mises à jour de cette liste sera toujours inférieur au nombre d'évènements de commutation qui arrivent à chaque tic d'horloge.

Si une nouvelle tâche prête arrive et est insérée en tête de liste, la tâche courante sera immédiatement interrompue, et la première sera élue pour exécution. Si deux tâches ont la même échéance, on en choisira une au hasard

4 Test de l'approche proposée

Pour valider le bon fonctionnement de l'approche complète, nous avons testé une série de scénarios en modifiant certaines contraintes du système tels que le niveau d'énergie disponible dans la batterie, le nombre de tâches en cours dans le système ou en changeant les préférences de l'utilisateur.

Il est à noter que:

- DlnDly représente : l'échéance de la tâche
- ExecTime représente : le temps d'exécution effectué sur le processeur
- TotExecTime représente : le temps écoulé entre le démarrage de la tâche et la fin de l'exécution
- TaskOver représente : 1si la tâche a terminé son exécution 0 si non
- En_model représente : la quantité d'énergie consommée par la configuration actuelle
- En_av représente : la quantité d'énergie disponible dans la batterie
- Life_time représente : durée de vie restante pour le système
- Task10, 11,12… représentent respectivement l'objet numéro 1,2,3…
- Task3 représente la tâche d'assemblage

Comme premier scénario nous avons choisi une scène 3D composée de deux objets un cube et un cylindre qui font une animation toutes les 2.5 s (période). Les contraintes de départ sont fixées comme suit: Lt_constraint: 240 secondes QoS_constraint 10, En_av = 105 joules (1).

La Figure 62 montre le démarrage du système. Le module d'adaptation choisi pour le cube une configuration SW avec l'algorithme d'ombrage plat et pour le cylindre une configuration SW avec un ombrage Gouraud (2). Toutes les configurations choisies répondent aux contraintes du système.

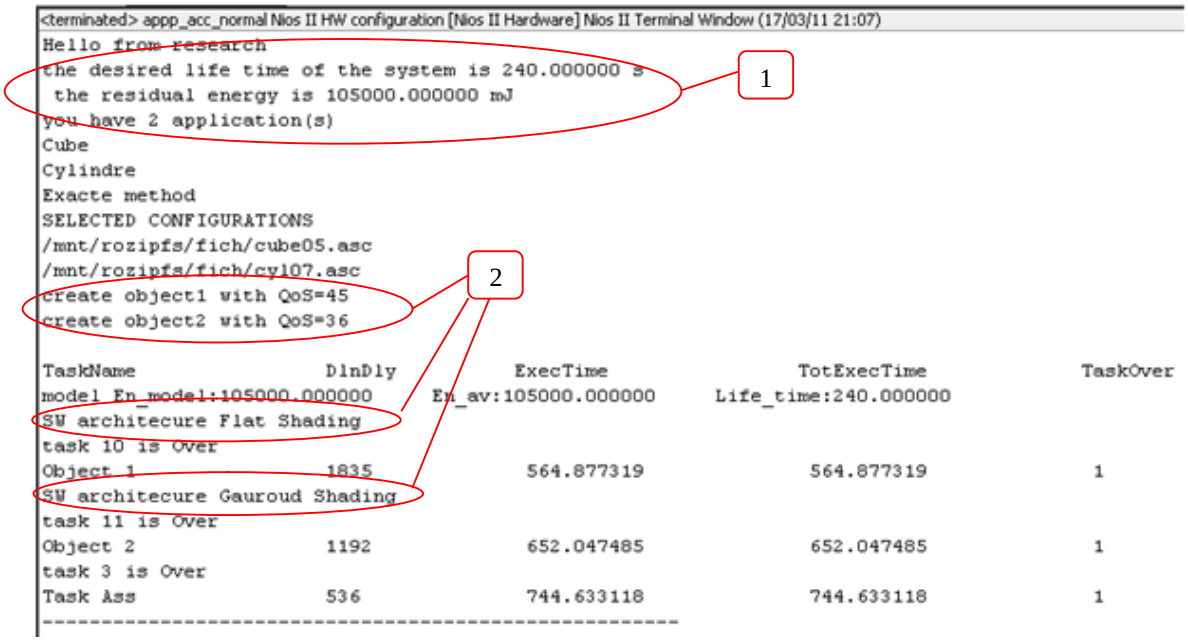

Figure 62 : Démarrage de l'approche d'adaptation

Dans la deuxième étape de ce scénario, on constate que la tâche 10 (1), dépasse son échéance, perturbe le fonctionnement du système (Figure 63) et cause un dépassement de l'hyper-période (2). Notez que le LM a activé la tâche d'adaptation locale (3), qui a consulté la base de configuration et a choisi une nouvelle configuration (4) de la tâche qui a provoqué le dépassement.

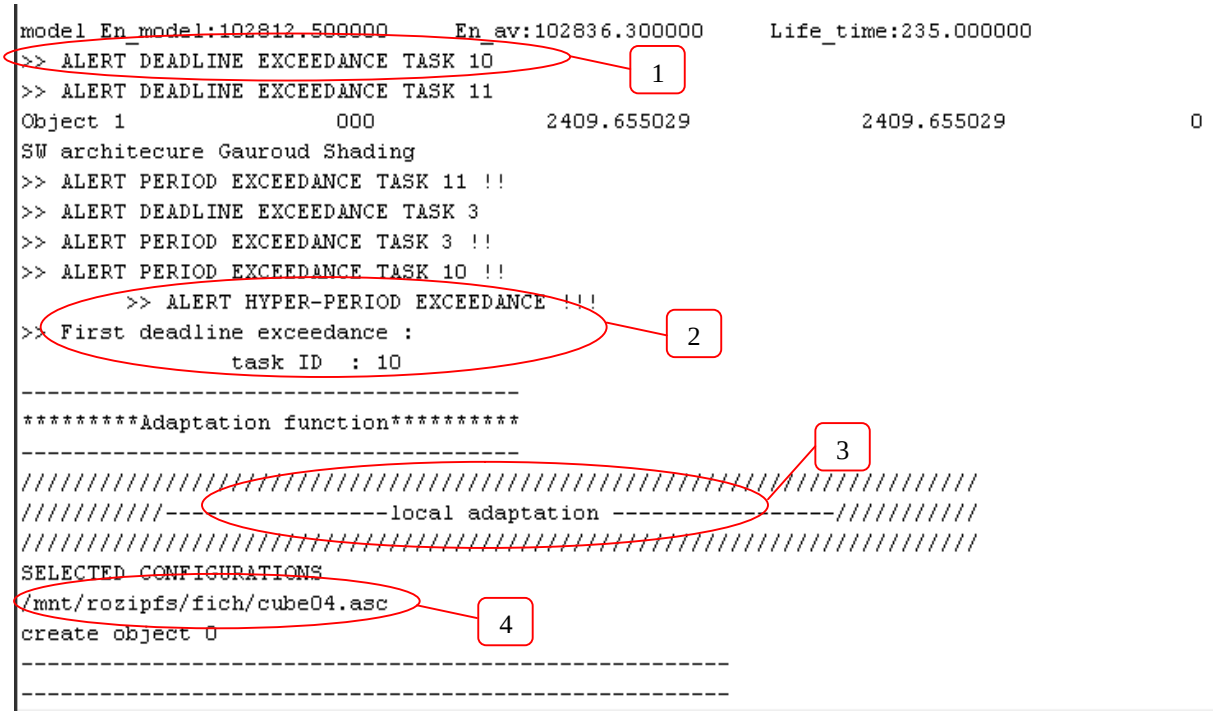

Figure 63: Appel de la fonction d'adaptation locale

Dans cette étape, un nouvel objet (deuxième cube) doit être rendu dans le système (1). La Figure 64 montre les différentes actions du module d'adaptation. Le système d'adaptation choisit de nouvelles configurations pour les tâches existantes dans le système.

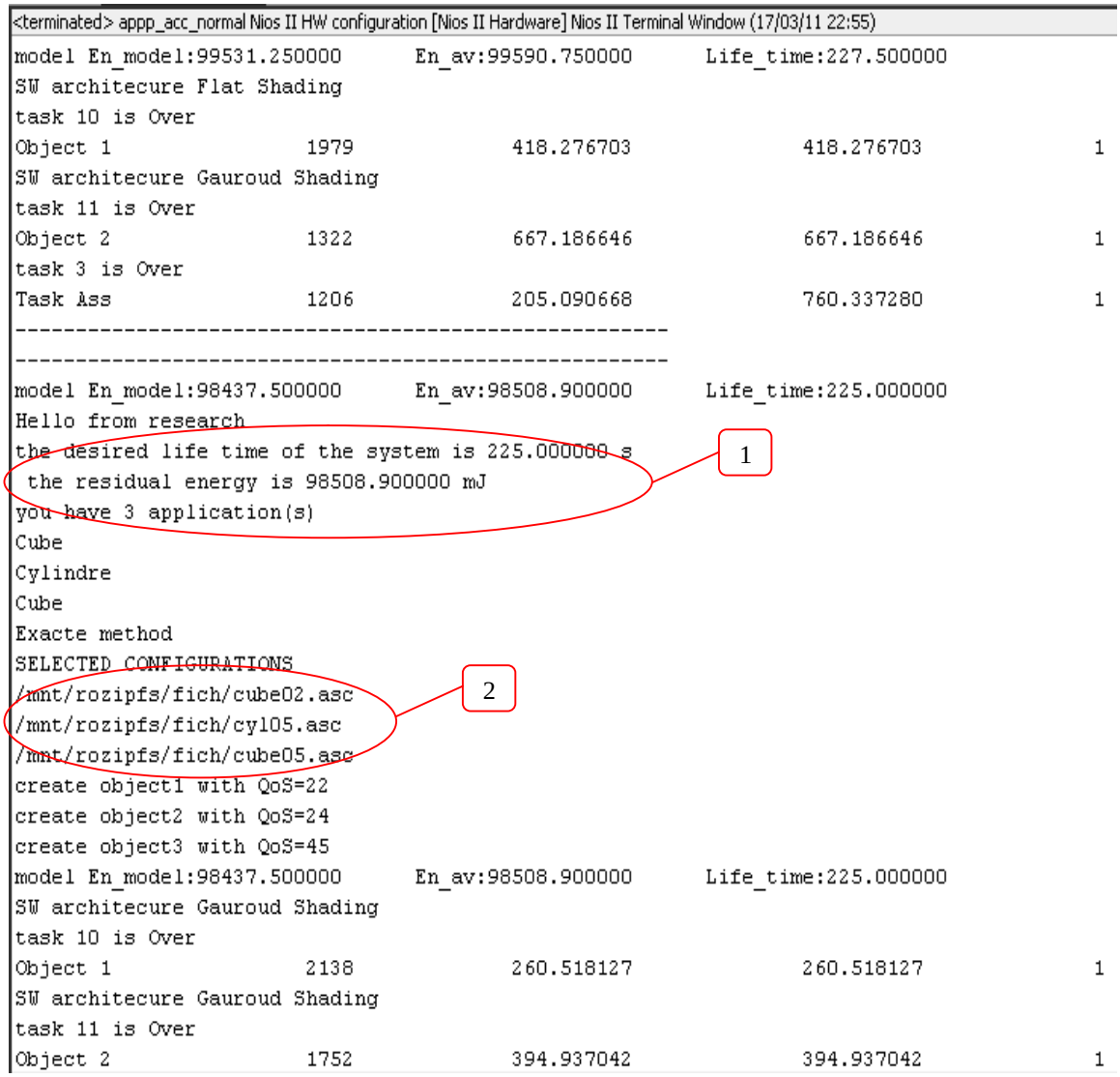

Figure 64: Modification du nombre de tâches

La dernière étape de ce scénario représente une situation où le gestionnaire local ne trouve pas de solution dans la base qui surmonte le dépassement d'échéance sans changer l'architecture du système. Ainsi, il demande au gestionnaire global de reconfigurer la totalité du système Figure 65.

En premier lieu le système détecte un dépassement de l'hyper période (1). Pour remédier à ce problème il fait appel à la fonction d'adaptation locale (2). Cette dernière n'arrive pas à

résoudre le problème localement elle active alors la fonction d'adaptation globale(3). De nouvelles configurations seront alors, choisies pour toutes les tâches (4).

Nous constatons, que la tâche 10 à été migrée vers une implémentation hardware pour accélérer le traitement et résoudre ainsi le problème de dépassement d'échéance (5).

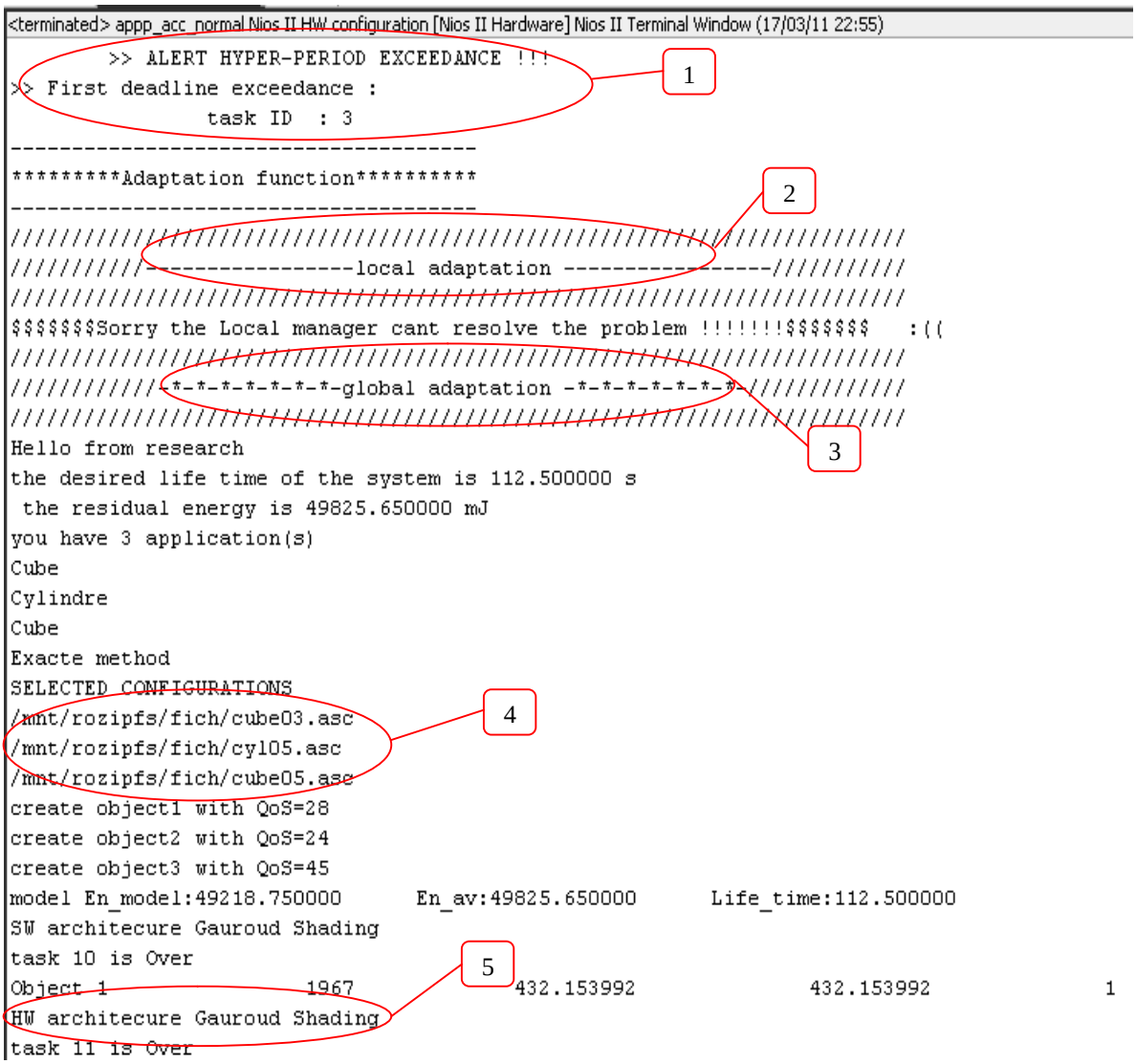

Figure 65: Activation du gestionnaire global par le gestionnaire local

5 Apport de l'approche

Le but de cette partie est de comparer le fonctionnement du système avec ou sans l'approche d'adaptation proposée. On rappelle que sans avoir utilisé l'approche de conception l'utilisateur ne peut choisir ni la durée de vie du système ni le niveau de qualité minimale pour chaque application. Toutes les applications s'exécutent sur une même architecture et avec les mêmes paramètres applicatifs fixés par le concepteur lors de la phase de conception.

Afin de valoriser l'apport de l'approche on a choisi de comparer la qualité de service du système offerte par l'approche d'adaptation et celle fournie par la version minimale qui fournit la plus basse qualité sur une architecture purement software d'une part et avec celle fournit avec la configuration de meilleure qualité sur une architecture qui contient tous les modules hardware implémentés.

1er Cas : qualité minimale

Pour ce premier cas on a choisi une quantité d'énergie disponible égale à 1800000mj et une durée de vie de 70 minutes.

La Figure 66 montre la différence entre la durée de vie du système et celle offerte par le système avec une version purement software qui fournit la plus basse qualité en modifiant le nombre d'objets présents dans le système.

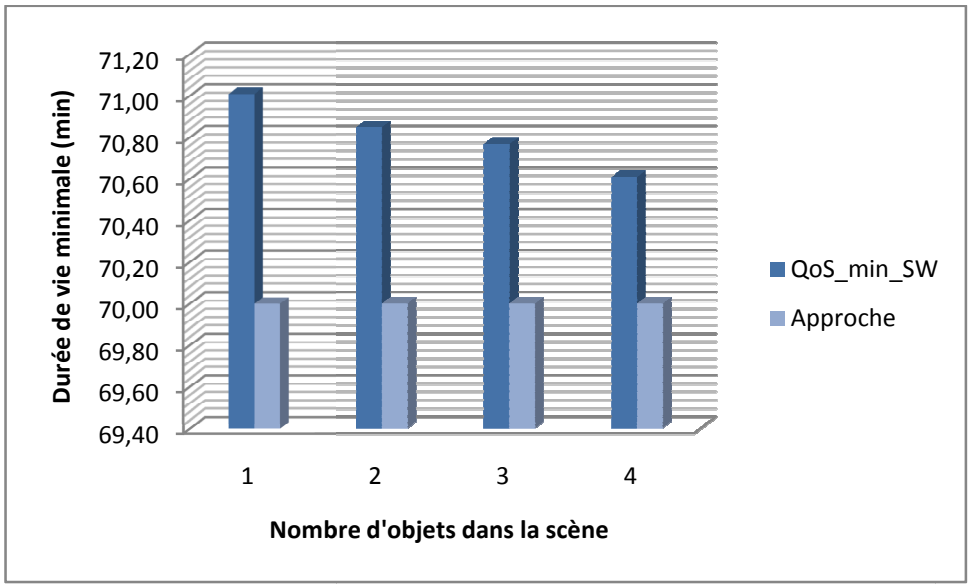

Figure 66: Durée de vie du système version minimale

La Figure 67 illustre la différence en termes de qualité de service (fonction objectif) fournit par les deux versions en variant le nombre d'objets dans le système.

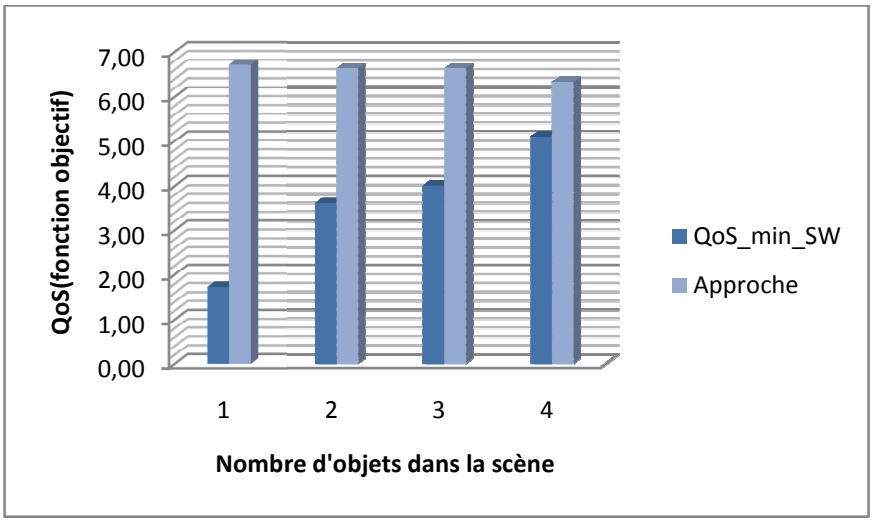

Figure 67: Variation de QoS pour une version minimale/Approche

Nous constatons d'après les deux graphes ci-dessus que la qualité de service fournie par l'approche est meilleure que la version minimale et que la différence peut atteindre 5 dans la fonction objectif alors que la différence dans la durée de vie du système ne dépasse pas 1 minute.

2ème cas Meilleure qualité

Pour le deuxième cas on a choisi une quantité d'énergie disponible égale à 1800000mj et une durée de vie égale dans les deux cas.

La Figure 68 montre la différence entre la durée de vie du système et celle offerte par le système avec une architecture qui contient tous les accélérateurs implémentés. Elle fournit la plus haute qualité de service.

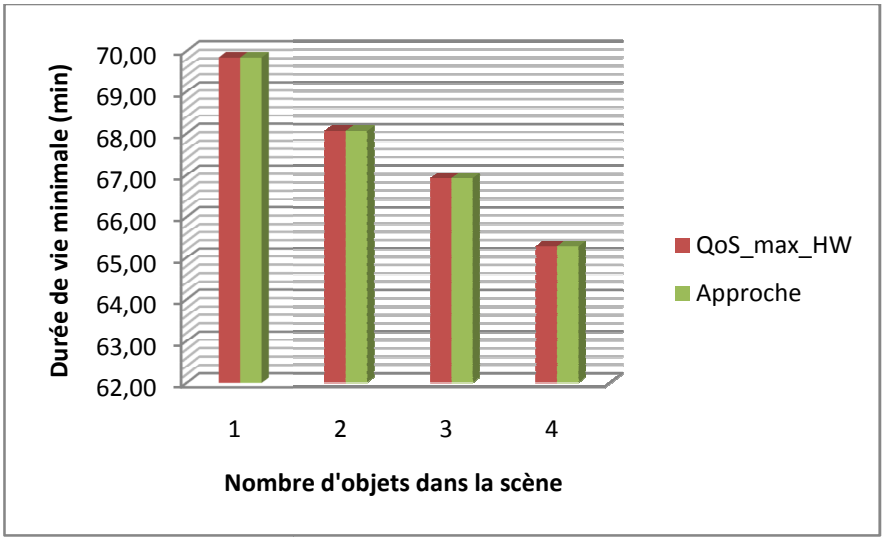

Figure 68: Durée de vie du système version maximale

La Figure 69 illustre la différence en termes de qualité de service fournie (fonction objectif) par les deux versions maximale/approche en variant le nombre d'objets dans le système.

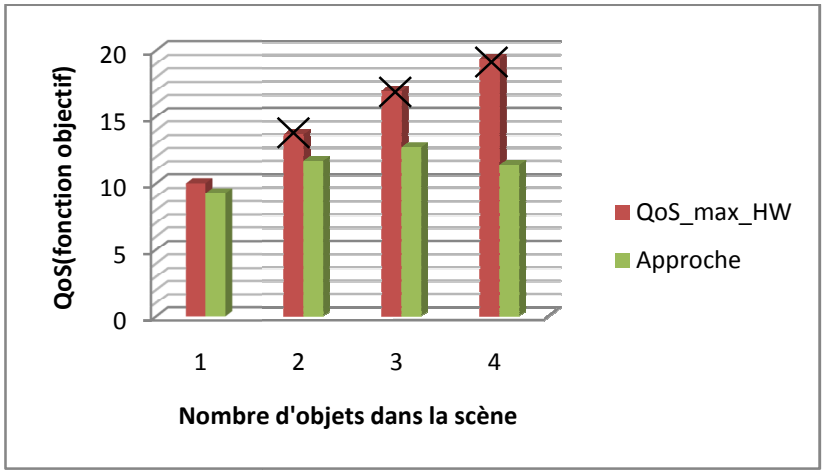

Figure 69: Variation QoS pour une version maximale/Approche

× Indique que le temps d'exécution dépasse l'hyper période du système

Nous remarquons que la version maximale fournit une qualité meilleure à celle de l'approche mais en contre partie le système ne respecte plus ses contraintes temporelles puisque le temps d'exécution de toutes les tâches dépasse l'hyper période du système.

6 Mise en œuvre des algorithmes d'optimisation

Dans cette étude nous mettons en œuvre deux algorithmes : l'algorithme génétique et le recuit simulé afin de comparer leurs performances et de choisir le plus approprié pour notre approche.

Comme nous l'avons déjà mentionné les algorithmes d'optimisation permettent de résoudre un problème et de fournir une solution au problème sans garantie d'optimalité. Toutefois, ces algorithmes possèdent des facteurs paramétrables qui influent sur la probabilité de trouver la solution optimale ou une solution très proche d'elle. Cependant, pour augmenter cette probabilité il y a toujours un prix à payer en termes de temps d'exécution. Nous sommes invités à fixer ces facteurs et à choisir l'algorithme adéquat pour notre système.

Dans notre travail, nous avons utilisé deux types d'algorithmes d'optimisation. Le premier est l'algorithme génétique dont on a choisi comme facteur le nombre d'itérations de l'algorithme. Nous adoptons une valeur fixe pour la longueur de la population pour l'algorithme génétique

(chaque nouvelle population = 20). Le deuxième est celui du recuit simulé dont on a choisi de travailler sur le facteur de dégradation de la température.

Nous rappelons que ces tests sont faits avec la même table de configurations décrite dans le tableau 6.

6.1 Premier scénario

Nous utilisons, dans ce scénario, une valeur faible pour le nombre d'itérations de l'algorithme génétique « GEN» (50 itérations) et un facteur élevé de la réduction de la température pour l'algorithme du recuit simulé « SA» (facteur = 0,8).

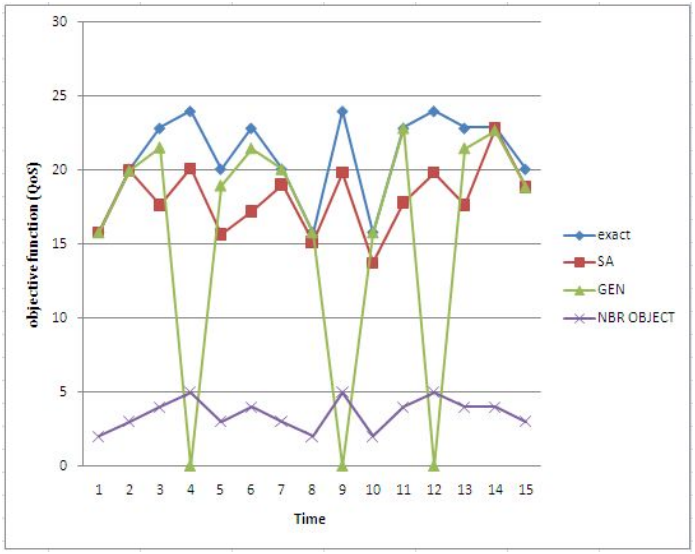

Figure 70: Variation nbr_objet/QoS pour l'algorithme génétique (nb_it=50) et le recuit simulé (fact=0.8)

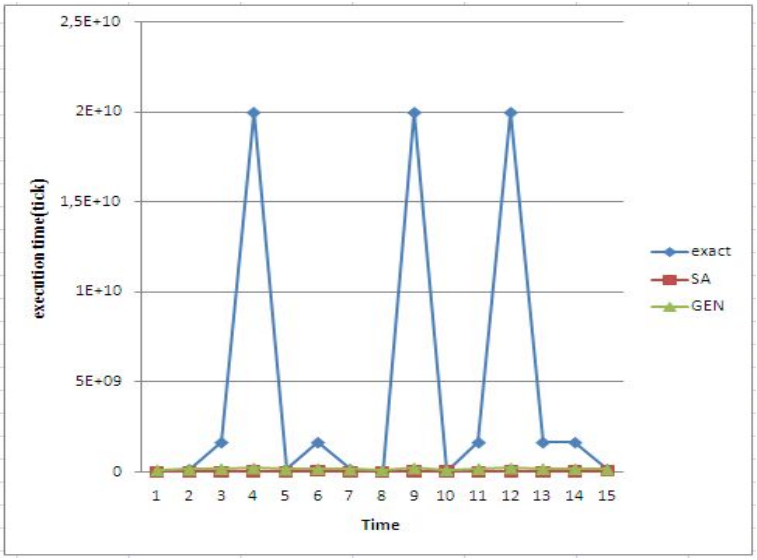

Figure 71: Variation nbr_objet/Texe pour l'algorithme génétique (nb_it=50) et le recuit simulé (fact=0.8)

Nous voyons pour ce premier essai (Figure 70 et Figure 71) que la méthode exacte pourra être utile si le nombre de tâches dans le système ne dépasse pas deux (t <0,13). Au-delà de ce nombre, le temps d'exécution de cette méthode est inacceptable (t = 1,32) pour 3 applications. Pour la méthode génétique nous notons que les valeurs de la fonction objectif sont très proches de la méthode exacte si le nombre de tâches est inférieur à quatre. Si nous dépassons cette valeur avec le même nombre d'itérations, nous ne trouverons pas les bonnes solutions. Nous notons également que le recuit simulé avec les solutions proposées est proche des résultats obtenus avec la méthode exacte et nous constatons aussi qu'il n'y a pas de solution inacceptable et que le temps est assez réduit dans tous les cas testés.

6.2 Deuxième scénario

Nous utilisons dans ce scénario une valeur élevée pour le nombre d'itérations de l'algorithme génétique (200 itérations) et un faible facteur de diminution de la température pour l'algorithme du recuit simulé (facteur = 0,95).

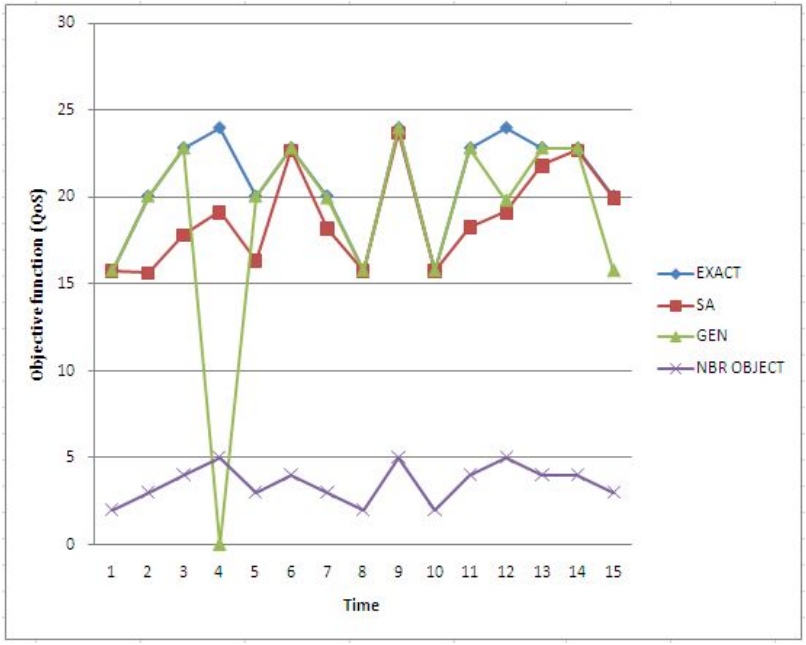

Figure 72: Variation nbr_objet/QoS pour l'algorithme génétique (nb_it=200) et le recuit simulé (fact=0.95)

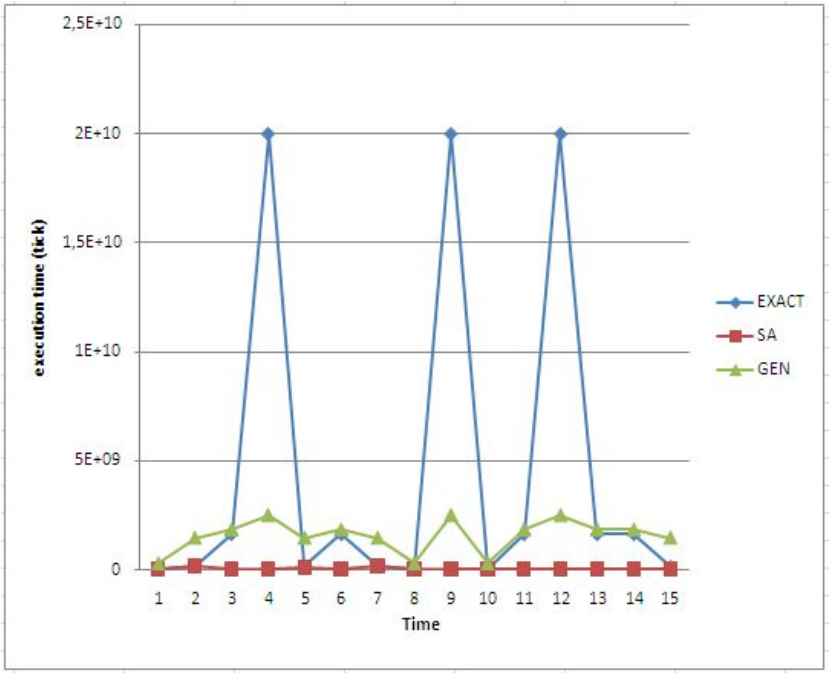

Figure 73: Variation nbr_objet/Texe pour l'algorithme génétique (nb_it=200) et le recuit simulé (fact=0.95)

Nous constatons pour le deuxième scénario (Figure 72 et Figure 73) que les valeurs fournies par l'algorithme génétique sont plus proches à la méthode exacte que le recuit simulé lorsque le nombre de tâches est inférieur à quatre. Nous notons une augmentation du temps d'exécution plus importante pour la méthode génétique.

6.3 Méthode mixte

Suite à cette expérience nous avons mené une étude visant à choisir la méthode à utiliser dans notre approche. Nous avons constaté que chacune peut être utile dans certains cas. Ainsi, nous proposons la méthode Mixte qui sélectionne l'une des trois méthodes en fonction des contraintes du système. Elle conduit à l'algorithme suivant:

- La méthode exacte si le système effectue au moins trois tâches.
- La méthode génétique avec un nombre d'itérations égal à 20 si le nombre de tâches est égal à 3
- La méthode génétique avec un nombre d'itérations égal à 200 si le nombre de tâches est égal à 4
- La méthode du recuit simulé si l'on a plus de quatre applications avec un facteur qui commence avec la valeur 0,8 et qui augmente suivant le nombre de tâches.

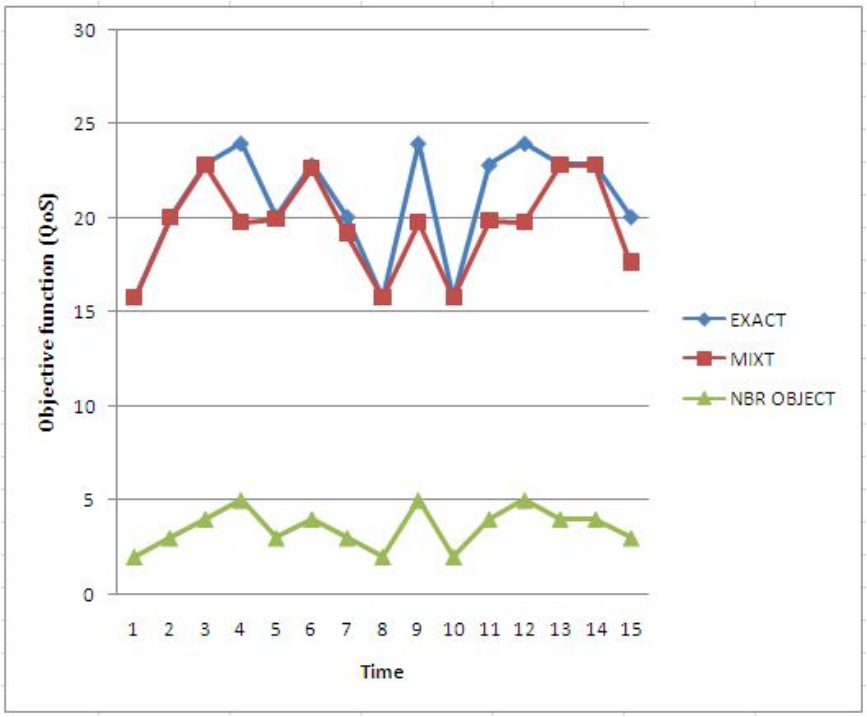

Figure 74: Variation nbr_objet/QoS pour la méthode mixte

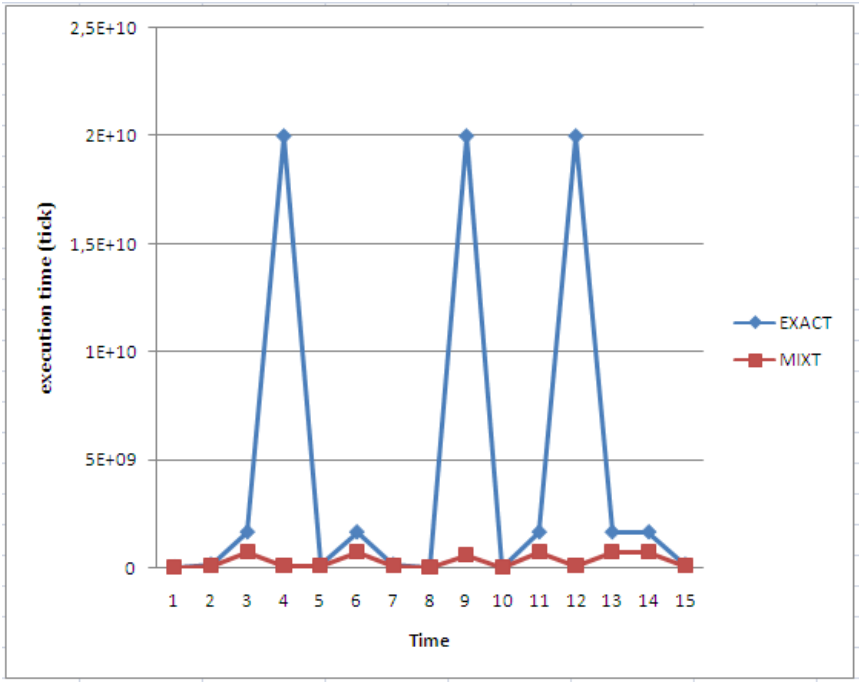

Figure 75 : Variation nbr_objet/QoS pour la méthode mixte

Nous notons que la qualité du service offert par cette méthode est très proche de la méthode exacte (Figure 74) avec un temps de réponse acceptable (Figure 75) et un taux d'erreur très faible.

7 Conclusion

Dans ce chapitre, nous avons détaillé les différentes étapes de la mise en œuvre du démonstrateur de l'approche d'adaptation à contraintes multiples. Cette validation a été faite à travers l'application de synthèse d'images 3D et l'environnement de conception d'altera. La première étape a consisté à la mise en place de la base des configurations HW/SW pour notre application. Cette partie englobe les étapes de partitionnement hardware/software de mise en place des architectures matérielles nécessaires ainsi que la caractérisation de chaque configuration (temps d'exécution au pire cas, énergie consommée pour une période, niveau de qualité fourni). Nous avons détaillé par la suite la mise en place du système d'exploitation temps réel avec l'ordonnanceur EDF et l'implémentation du gestionnaire global et local. Finalement nous avons présenté quelques scénarios pour la validation du fonctionnement de l'approche proposée. Ce chapitre est clôturé par une comparaison entre les différentes méthodes d'optimisation implémentées.

Conclusions générales

1 Conclusion

Actuellement on assiste à une émergence de systèmes multimédias électroniques embarqués destinés à un large public d'utilisateurs. Leurs fonctionnalités sont de plus en plus complexes et diversifiés. Les concepteurs de ces systèmes sont soumis envers une contrainte fastidieuse qui est le temps de mise sous le marché « Time To Market ». Pour gagner ce défi des méthodes de conception logicielle matérielle ont été développées. Elles tiennent en compte uniquement des caractéristiques des applications pour définir une architecture en adéquation.
Cependant, les systèmes sur puce doivent fonctionner dans des conditions souvent difficiles fluctuation des conditions de transmission en réseau, ressources d'énergie limitée, contraintes imposées par l'utilisateur etc. Tous ces paramètres « dynamiques » ne sont pas tenus en compte dans les méthodes classiques de codesign existantes.
De ce fait, les méthodes de conception classiques ne permettent plus de répondre aux exigences des systèmes actuels et doivent être améliorées par d'autres techniques afin de surmonter ces problèmes. En effet, elles doivent permettre la conception de systèmes performants pour pouvoir traiter les applications multimédia complexes et d'autre part flexibles pour s'adapter à l'environnement externe variable pour respecter les contraintes imposées par les ressources du système, l'environnement externe et l'utilisateur.

2 Réponse à la problématique et travail réalisé

Une bonne conception d'un système multimédia embarqué doit tenir compte non seulement de la consommation du système mais aussi du comportement du système envers le changement dynamiquement de ses contraintes.

Ces contraintes sont d'autant plus difficiles à réaliser qu'elles sont sous l'influence de paramètres externes souvent aléatoires et imprévisibles (influence de la variation des données sur la consommation, influence des choix de l'utilisateur…). Nous avons proposé donc une approche d'adaptation pour répondre aux contraintes durée de vie / Temps réel / Qualité de service. Cette méthode suppose d'une part l'existence de divers modes de fonctionnements du système et d'autre part que celui-ci est capable de passer d'un mode à un autre suivant l'évolution des paramètres durée de vie, temps d'exécution et QoS.

Cette méthode se compose de trois activités principales qu'on a classées en deux étapes : l'une se fait hors ligne lors de la conception du système et l'autre en ligne, elle intervient au cours du fonctionnement du système:

- Etape de caractérisation hors ligne : elle permet de déterminer les différents modes de fonctionnement du système (configurations). Cette étape nécessite :
  - Le choix d'un modèle d'architecture et une cible technologique.
  - L'étude de l'application afin de déterminer les attributs qui peuvent influencer le compromis Ddv/Texe/QoS. Ces attributs sont de deux natures : applicatifs et architecturaux. Les attributs applicatifs regroupent les valeurs des différents algorithmes (ou leurs paramètres associés) et dont le changement a un impact direct sur le compromis Ddv/Texe/QoS. Les attributs architecturaux concernent le type d'implémentation des configurations. Dans cette étape, nous utilisons une approche de partitionnement qui se base sur l'utilisation de l'outil Design Trotter et le résultat de profilage de l'application sur l'architecture cible afin d'obtenir des implémentations en adéquation avec l'application cible. À chaque couple d'attributs applicatif/architectural correspond une configuration, elle est caractérisée par un triplet {Energie_consommée, Texe, QoS}. Pour ceci il a fallu mettre en place des méthodes de quantification de la consommation d'une configuration en termes d'énergie et de temps d'exécution et de la qualité de service fourni à l'utilisateur.
- Etape « en ligne » qui consiste à mettre en place un modèle d'adaptation qui permet de suivre en ligne l'évolution des contraintes Ddv, Texe et QoS et de reconfigurer le système selon les consignes imposées par l'utilisateur. Cette étape est formée par deux activités : une activité d'observation et une activité d'adaptation
  - L'activité d'observation permet le suivi des trois paramètres Ddv, Texe et QoS.
  - Une activité d'adaptation qui permet de choisir les configurations adéquates pour toutes les applications présentes sur le système afin de satisfaire les consignes de l'utilisateur. Vu que cette tâche ne doit pas dégrader les performances du système on a eu recourt à trois méthodes d'optimisation pour le choix d'une combinaison à partir de la base des configurations déterminées lors de caractérisation hors ligne.

Cette méthode a été validée à travers un environnement de prototypage des systèmes sur puce reconfigurable d'Altera. Nous avons retenu la fonction de rendu d'image 3D comme application cible.

3 Perspectives

Le travail effectué dans cette thèse peut être étendu et amélioré dans plusieurs axes. Nous présentons quelques uns dans la suite.

Le premier thème que nous proposons, est la validation de l'approche sur une architecture reconfigurable dynamiquement (tel que les FPGA de type Xilinx). Bien entendu de nouveaux facteurs doivent être pris en compte tel que le coût de la reconfiguration du système en termes de temps d'exécution et de consommation. Nous proposons aussi d'ajouter d'autres applications multimédia telles que l'application H264. Il est à noter que ces travaux ont déjà commencé au sein de notre équipe dans le cadre du projet CMCU « CESAME ».

Le deuxième thème concerne l'extension de cette approche pour quelle supporte les architectures multiprocesseurs. Beaucoup de problèmes sont à surmonter pour que cette approche puisse supporter les architectures multiprocesseurs telles que l'affectation de la charge de travail à chaque application, l'affectation des tâches à un processeur, la reconfiguration de l'architecture du système et bien évidemment le problème d'ordonnancement puisque le EDF n'est plus optimal pour une architecture multiprocesseurs.

Le troisième axe touche l'utilisation des modules d'accélérations hardware existants dans l'architecture par plusieurs applications en même temps. Dans notre travail nous avons supposé que chaque accélérateur ne peut être utilisé que par une seule application mais rien n'empêche que ce dernier soit utilisé par d'autres applications puisqu'il est présent sur le système. Bien entendu, ce partage nécessite l'utilisation des mécanismes nécessaires pour l'ordonnancement des tâches sur l'accélérateur.

En dernier lieu nous proposons de faire une étude exacte pour choisir la méthode d'optimisation adéquate à utiliser dans l'approche d'adaptation.

Références

[Ana05] Ana Belén Abril Garcia *estimation et optimisation de la consommation dans les descriptions architecturales des systèmes intégrés complexes* thèse juin 2005, université de paris 6.

[Aou06a] Y. Aoudni, Kais Loukil , Guy Gogniat, Jean Luc Philippe and Mohamed Abid, *Mapping SoC architecture Solutions for an Application based on PACM Model,* International Symposium on Industrial Electronic (IEEE ISIE'06) Montreal Canada 2006.

[Aou06b] Y. Aoudni, G. Gogniat, K. Loukil, J. L. Philippe, M. Abid *Method for Embedded Application Prototyping based on SoC Platform and Architecture Model* (IEEE DTIS 2006), Tunis, Tunisia 2006. Pages 73-83

[Aud 04] Audrey MARCHAND et Maryline SILLY-CHETTO *Simulation et évaluation d'algorithmes d'ordonnancement temps-réel sous des contraintes de QoS,* Septembre 2004, rapport de recherche N° 0405

[Ara05] Arato. P, Mann. A. Orban. A, *Algorithmic Aspects of Hardware-Software partitioning,* ACM Transactions on Design Automation of Electronic System, Vol. 10, No. 1, 2005. Pages 532 - 544

[Arn09] Arnaldo Azevedo, *Parallel H.264 Decoding on an Embedded Multicore Processor*, Lecture Notes in Computer Science, 2009, Volume 5409, High Performance Embedded Architectures and Compilers, Pages 404-418

[Azz04] Azzedine Abedennour, *outil d'analyse et de partitionnement pour les systèmes temps réel embarqués,* livre publié en 2004

[Bam01] N. Bambha, S. Bhattacharyya, J. Teich, E. Zitzler, *Hybrid Global/Local Search Strategies for Dynamic Voltage Scaling in Embedded Multiprocessors,* In Proc. Hardware/Software Co-Design (CODES'01), pages. 243-248, 2001.

[Ban02] S. Banachowski and S. Brandt, *The BEST scheduler for integrated processing of best-effort and soft real-time processes* in Proc. of SPIE Multimedia Computing and Networking Conference, San Jose, CA, Jan. 2002.

[Bap06] Baptiste A. *Les métaheuristiques en optimisation combinatoire* rapport de recherche, 9 mai 2006.

[Bat98] IJ Bates *Scheduling and Timing Analysis for Safety Critical Real-Time Systems,* PhD thesis, University of York, Nov 1998.

[Bav00] A. Bavier and L. Peterson, *The power of virtual time for multimedia scheduling,* Proc. of 10th International Workshop for Network and Operating System Support for Digital Audio and Video (NOSSDAV), June 2000.

[Ben05] Ben Amor N. , Y.Le Moullec, J-Ph.Diguet, J-L.Philippe, M.Abid, *Design of a multimedia processor based on metrics computation* Special Issue for "Advances in Engineering Software", volume 36 (2005) pages 448–458.

[Ben07] N. Ben Amor *Approche de conception de processeur de vision embarqué* thèse 2007, université de Sfax

[Bra02] S. Brandt and G. J. Nutt, *Flexible soft real-time processing in middleware* Real-Time Systems Volume 22, Numbers 1-2, pages 77-118, 2002.

[Bru06] Bruno Gaujal and Nicolas Navet. *Ordonnancement temps réel et minimisation de la consommation d'énergie,* In Systèmes temps réel - volume 2, pages 109-133, 2006

[Cat01] Catha. K, Vemuri. R, Magellan, *multiway hardware-software partitioning and scheduling for latency minimization of hierarchical control-dataflow task graphs,* Proceedings of ACM international Conference on Hardware/Software Codesign and System Synthesis, page 364, 2001.

[Chr11] Christian Bachmann, *An Automated Framework for Power-Critical Code Region Detection and Power Peak Optimization of Embedded Software*, Integrated Circuit and System Design.

Power and Timing Modeling, Optimization, and Simulation Lecture Notes in Computer Science, 2011, Volume 6448/2011, pages 11-20.

[Cor01] M. Corner, B. Noble, and K. Wasserman, *Fugue: time scales of adaptation in mobile video,* in Proc. of SPIE Multimedia Computing and Networking Conference, San Jose, CA, Jan. 2001.

[Dan04] Daniel Mossé Hakan Aydin, Rami Melhem. *Periodic reward-based sheduling and its application to power-aware real-time systems* to appear in Handbook of Scheduling : Algorithms, Models, and Performance Analysis, 2004.

[Dan09] Daniel Menard, et al *Reconfigurable Operator Based Multimedia Embedded Processor,* Reconfigurable Computing: Architectures, Tools and Applications Lecture Notes in Computer Science, 2009, Volume 5453/2009, pages 39-49

[Dav97] B. Dave, G. Lakshminarayana, N. Jha, *COSYN : Hardware-Software CoSynthesis of Embedded Systems,* in Proc. ACM/IEEE Design Automation Conference, pages 703-708, Anaheim, CA, June 1997.

[Fli01] J. Flinn, E. de Lara, M. Satyanarayanan, D. Wallach, and W. Zwaenepoel, *Reducing the energy usage of office applications,* in Proc. of Middleware 2001, Heidelberg, Germany, Nov. 2001.

[Fli99] J. Flinn and M. Satyanarayanan, *PowerScope: A tool for profiling the energy usage of mobile applications,* in Proc. of 2nd IEEE Workshop on Mobile Computing Systems and Applications, Feb. pages 2-10, New Orleans, LA , USA 1999.

[For98] W. Fornaciari, P. Gubian, D. Sciuto, C. Silvano, *Power Estimation of Embedded Systeme: A Hardware/Software Codesign Approach* IEEE Trans. VLSI Systems, June 1998, pages. 266-275.

[Fra10] Francky Catthoor *Global State-of-the-Art Overview,* book chapter Ultra-Low Energy Domain-Specific Instruction-Set Processors 2010, pages 17-32

[Fré00] Frédéric Parain et al. *Techniques de réduction de la consommation dans les systèmes embarqués temps-réel,* Rapport de recherche Mai 2000

[Fre02] Frédéric Le Mouël, *AeDEn: An adaptive framework for dynamic distribution over mobile environments,* Annals of Telecommunications Volume 57, 2002

[Gan10] Ganghee Lee, et al, *Communication architecture design for reconfigurable multimedia SoC platform,* Design Automation for Embedded Systems, 2010, Volume 14, Number 1, Pages 1-20

[Gru01] F. Gruian, K. Kuchcinski, *LEneS : Task Scheduling for Low-Energy Systems Using Variable Supply Voltage Processor*, In Proc. Asia South Pacific - Design Automation Conf. (ASP-DAC'01), pages. 449-455, 2001.

[Hen08a] M. Hentati, N. Ben Amor, K. Loukil, M.abid, *HW/SW Interface Impact on an Adaptive Multimedia System Performance: Case study*,First IEEE International Workshops on Image Processing theory, Tools& applications November 24-26, 2008-Tunisia.

[Hen08b] Hentati Manel, *Techniques d'interfaçage dans les SoC : étude de cas*, juin 2008, école nationale des ingénieurs de sfax, tunisie

[Hen99] Henry Chang, Larry Cooke, Merrill Hunt, Grant Marti n, Andrew McNelly, Lee Todd, *Surviving the SOC Revolution : A Guide to Platform-Based Design,* Kluwer Academic Publishers, ISBN 0-79238679-5, 1999

[Hif07] Hifi M. , Michrafy M. Reduction strategies and exact algorithms for the disjunctively constrained knapsack problem, Computers and Operations Research, 2007.

[Hua06] Huaxiang Lu, *SOC Dynamic Power Management Using Artificial Neural Network,* Advances in Natural Computation Lecture Notes in Computer Science, 2006, Volume 4221/2006, pages 555-564

[Iva05] Ivano Barbieri et al. *A Simulation and Exploration Technology for Multimedia-Application-Driven Architectures,* The Journal of VLSI Signal Processing, 2005, Volume 41, Number 2, Pages 153-168

[Jal09] Jalel Ktari, Mohamed Abid, *A Low Power Design Space Exploration Methodology Based on High Level Models and Confidence Intervals, A*SP Journal of Low Power Electronics March, 2009 Volume : 5, Pages : 1-14

[Jph11] cle: J.Ph Diguet, Y. Eustache, G. Gogniat, *Closed-loop based self-adaptive HW/SW embedded systems: design methodology and smart cam case study*, in ACM Transactions on Embedded Computing Systems, vol.10, issue.3, April 2011.

[Jul04] Júlio C. B. Mattos, *Design Space Exploration with Automatic Selection of SW and HW for Embedded Applications,* Computer Systems: Architectures, Modeling, and Simulation Lecture Notes in Computer Science, 2004, Volume 3133/2004, pages 103-118

[Kan02] Kanishka Lahiri et al. *Battery Driven System Design: A Nex Frantier in Low Power Design*

Design Automation Conference, 2002. Proceedings of ASP-DAC 2002. 7th Asia and South Pacific and the 15th International Conference on VLSI Design. Proceedings. 2002 , Pages 261 - 267

[Kan11] Kan Fang, *A New Shop Scheduling Approach in Support of Sustainable Manufacturing,* Glocalized Solutions for Sustainability in Manufacturing 2011, pages 305-310

[Kim03] Kim N., Austin T., Blaauw D., Mudge T., *Leakage Current: Moore's Law Meets Static Power*, IEEE Computer Society revue Computer, Pages 68-75, 2003.

[Koi06] Koichiro ISHIBASHI et al. *Low-Voltage and Low-Power Logic, Memory, and Analog Circuit Techniques for SoCs Using 90 nm Technology and Beyond,* IEICE TRANS. ELECTRON., VOL.E89–C, NO.3 MARCH pages 250-262, 2006

[Kon05] Konstantinos Masselos and Nikolaos Voros, *Introduction to Reconfigurable Hardware*

2005, System Level Design of Reconfigurable Systems-on-Chip, Part A, Pages 15-26

[Kri08] Kristopher Welsh and Pete Sawyer, *When to Adapt? Identification of Problem Domains for Adaptive Systems,* Lecture Notes in Computer Science, 2008, Volume 5025, Requirements Engineering: Foundation for Software Quality, Pages 198-203

[Kri09] Kris Gaj and Pawel Chodowiec**,** *FPGA and ASIC Implementations of AES* Cryptographic Engineering , pages 235-294, 2009

[Kuh98] Kuhn P.M., Stechele W. *Complexity Analysis of the Emerging MPEG-4 Standard as a Basis for VLSI Implementation* Visual Communications and Image Processing SPIE 3309, San Jose, california, 1998.

[Lab02] Labrosse J.J., *MicroC/OS-II, The real time kernel,* R&d books editions, 2002, ISBN 1-57820-103-9

[Lac04] Lacroix J., Terrade S. *Algorithmes Génétiques,* livre17 novembre 2004.

[Lad01] Laddaga, R. *Active software*. Lecture Notes in Computer Science, 2001

[Lay09] LAYEB.A, *introduction aux métaheuristiques,* cours recherche opérationnelle, 2009

[Lee00] S. Lee, T. Sakurai, *Run-time Voltage Hopping for Low-power Real-time Systems*, In Proc IEEE 37th Design Automation Conf. (DAC00), pages 806-809. 2000.

[Lic06] Licandro F., et al. *A Multimedia Adaptive-Quality Platform For Real-Time E-Learning Over IP***,** Signals and Communication Technology, Distributed Cooperative Laboratories: Networking, Instrumentation, and Measurements, Chapter I, Pages 29-44, 2006.

[Lin10a] Lina Jarboui, *Étude et implémentation d'une technique d'adaptation multi contraintes pour les systèmes multimédia embarqués,* Rapport projet fin d'étude Enis-2010

[Lin10b] Linfeng Ye, Jean-Philippe Diguet, Guy Gogniat, *Rapid Application Development on Multi-processor Reconfigurable Systems,* FPL'2010. Pages 285-290

[LL73] C. L. Liu and James W. Layland, *Scheduling algorithms for multiprogramming in a hard-real-time environment,* Journal of the ACM, 1973.

[Lor98] Lorch Jacobs R. And Alan Jay Smith, *Software strategies portable computer energy management*, IEEE Personal communications Magazine. June 1998

[Lou04] Loukil K., Ben Cheikha H., *Conception d'accélérateurs pour la synthèse d'image 3d*, juin 2004, université du sud, Faculté des Sciences de Sfax, tunisie.

[Lou09a] K. Loukil, N. Ben Amor, Mouna Ben said, Mohamed Abid, *OS service update for an online adaptative embedded multimedia system,* the fourteenth IEEE Symposium on Computers and Communications (ISCC'09) July 5-8, 2009 Sousse, Tunisia

[Lou09b] K. Loukil, N. Ben Amor, M. Hentati, M. Abid, *HW/SW partitioning approach on reconfigurable multimedia system on chip,* The 7th ACS/IEEE International Conference on Computer Systems and Applications, Rabat, Morocco. May 10-13, 2009

[Lou09c] K. Loukil, N. Ben Amor, M. Abid, *Self adaptive reconfigurable system based on middleware cross layer adaptation model,* IEEE SSD 2009 Djerba, Tunisia 2009.

[Luo02] J. Luo, N. K. Jha, *Static and dynamic variable voltage scheduling algorithms for real-time heterogeneous distributed embedded systems,* Int. Conf. on VLSI Design, Jan. 2002.

[Man03] A. Manzak, C. Chakrabarti, *Variable Voltage Task Scheduling Algorithms for Minimizing Energy/Power,* in IEEE Trans on VLSI Systems, April 2003.

[Mar05] Marcus T. Schmitz, Bashir M. Al-Hashimi and Petru Eles, *Power Variation-Driven Dynamic Voltage Scaling,* System-Level Design Techniques for Energy-Efficient Embedded Systems 2005, pages 35-59

[Mat07] Matthias Grumer, *Software Power Peak Reduction on Smart Card Systems Based on Iterative Compiling,* Emerging Directions in Embedded and Ubiquitous Computing Lecture Notes in Computer Science, 2007, Volume 4809/2007, pages 627-637

[Mat09] Matt Klein, *Power Consumption at 40 and 45 nm,* White Paper: Spartan-6 and Virtex-6 Devices, WP298 (v1.0) April 13, 2009

[Mes02] M. Mesarina and Y. Turner, *Reduced energy decoding of MPEG streams,* in Proc. of SPIE Multimedia Computing and Networking Conference, San Jose, CA, Jan. 2002.

[Moo65] G. E. Moore. *Cramming more components onto integrated circuits*, Electronics, Volume 38, Number 8, April 19, 1965.

[Mou03a] Y. Le Moullec, D. Heller, J.-P. Diguet, J.-L. Philippe, *Estimation du parallélisme au niveau système pour l'exploration de l'espace de conception de systèmes enfouis,* Technique et Science Informatiques (RSTI-TSI), Vol. 22, nº3/2003, pages315-349, Lavoisier Hermes-Science publications.

[Mou03b] Y. Moullec, *Aide à la conception de systèmes sur puce hétérogènes par l'exploration paramétrable des solutions au niveau système,* Thèse, l'Université De Bretagne Sud, 2003.

[Muh10] Muhammad Khurram Bhatti, *Assertive Dynamic Power Management (AsDPM) Strategy for Globally Scheduled RT Multiprocessor Systems,* Integrated Circuit and System Design. Power and Timing Modeling, Optimization and Simulation Lecture Notes in Computer Science, 2010, Volume 5953/2010, p116-126

[Muh11] Muhammad Khurram Bhatti, *Hybrid power management in real time embedded systems: an interplay of DVFS and DPM techniques,* Real-Time Systems Volume 47, Number 2, pages 143-162, junary 2011

[Nab10] Nabila Moubdi et al. *Product On-Chip Process Compensation for Low Power and Yield Enhancement,* Lecture Notes in Computer Science, Volume 5953/2010, pages 247-255, 2010

[Oku01] T. Okuma, T. Ishihara, H. Yasuura, *Software Energy reduction Techniques for Variable Voltage Processors*, IEEE Design & Test of Computers, March-April, 2001.

[Pan01] D. Panigrahi , C. Chiasserini, S. Dey, R. Rao , A. Raghunathan and K. Lahiri, *Battery Life Estimation of Mobile Embedded Systems,* 14ème Intl. Conf. on VLSI Design, Bangalore, Inde, janvier 2001.

[Pan05] Pan Y., Cheng I., Basu A. *Quality Metric for Approximating Subjective Evaluation of 3-D Objects,* IEEE transactions on multimedia, Vol. 7, N°. 2, pages269, avril 2005.

[Pha03] N. Pham Ngoc, G. Lafrui, G. Deconinck, and R. Lauwereins *Terminal QOS management on run-time reconfigurable platforms*, Third PA3CT-symposium pages 22-23 September 2003

[Pha04] N. Pham Ngoc, W. van Raemdonck, G. Lafruit, G. Deconinck, and R. Lauwereins, *A QoS framework for interactive 3D applications*, Proceedings of the ninth international conference on 3D Web technology 2004

[Qua01] G. Quan, X. Hu, *Energy efficient fixed-priority scheduling for real-time systems on variable voltage processors,* Design Automation Conference, Proceedings, 2001, pages 828 -833.

[Ram05] Raman. B, Chakraborty. S, Ooi. W, *Meeting CPU constraints by delaying playout of multimedia tasks*, Proceedings ACM International Workshop on Network and Operating System Support for Digital Audio and Video, 2005, pages 165-170.

[Rut02] Rutten. M et al. *A Heterogeneous Multiprocessor Architecture for Flexible Media processing*

IEEE Transactions on Design and Test of Computers, Vol. 19, No. 4, 2002, pages 39-50.

[Sia03] Siarry P., Dréo J., Pétrowski A., Taillard É. Métaheuristiques pour l'optimisation difficile

Livre Septembre 2003.

[Seo10] SEONGSOO LEE, *Modeling and Description of Semiconductor IP interfaces for System-Level SoC Design,* CSECS '10 Vouliagmeni, thens, Greece December 29-31, 2010

[Shi01] D. Shin, J. Kim and S. Lee, *Intra-Task Voltage Scheduling for Low-Energy Hard Real-Time Applications*, IEEE Design and Test of Computers, Vol.18 No.2, pages 20-30, March 2001.

[Shi04] D. Shin and J. Kim, *Dynamic Voltage Scaling of Periodic and Aperiodic Tasks in Priority-Driven Systems,* in Proc. ASPDAC'03, pages 653-658, Jan. 2004.

[Sof07] Sofien Chtourou, *Méthodologies de synthèse de réseaux de neurones pour applications de traitement de signal adaptatif et implémentation sur circuits reconfigurables dynamiquement ,*

thèse 2007

[Swa01] V. Swaminathan, K. Chakrabarty, S. S. Iyengar, *Dynamic I/O power management in hard real-time systems,* Proceedings of the International Symposium on Hardware/ Software Codesign, pages 237-243, 2001.

[Tou00] Touradj E., Horne C., *MPEG-4 natural video coding - An overview,* journal of Signal Processing: Image Communication, vol. 15, 2000 , pages 365-385.

[Tur03] Turley J., *Embedded Processors of Tomorrow*, Embedded Systems Programming, novembre 2003, pages 36-38.

[Van02] W. Van Raemdonck, G. Lafruit, E.F.M. Steffens, C.M. Otero Pérez, R.J. Bril, *Scalable graphics processing in consumer terminals*, Multimedia and Expo, 2002. ICME '02. Proceedings. 2002 IEEE International Conference

[Var04] Varatkar .G.V and Marculescu. R, *On-chip traffic modelling and synthesis for MPEG2 video applications*, IEEE Transactions on Very Large Scale Integration (VLSI) systems, 2004, pages 108-119.

[Wan03] Wanghong Yuana, Klara Nahrstedta, Sarita V. Advea, Douglas L. Jonesb, Robin H. Kravets, *Design and Evaluation of a Cross-Layer Adaptation Framework for Mobile Multimedia Systems*, Appears in SPIE/ACM Multimedia Computing and Networking Conference (MMCN), 2003

[Xin07] Xiang LingXiang, *The Design and Implementation of the DVS Based Dynamic Compiler for Power Reduction,* Advanced Parallel Processing Technologies Lecture Notes in Computer Science, 2007, Volume 4847/2007, pages 233-240

[Yua06] Yuan W., and K. Nahrstedt, *Energy-Efficient CPU Scheduling for Multimedia Applications,* ACM Transactions on Computer Systems, Vol. 24, No. 3, Pages 292–331, Aug. 2006

[Yut11] Yuto Hayamizu, *Application-Aware Power Saving for Online Transaction Processing Using Dynamic Voltage and Frequency Scaling in a Multicore Environment,* Architecture of Computing Systems - ARCS 2011 Lecture Notes in Computer Science, 2011, Volume 6566/2011, pages 50-61

[Zhi08] Zhiyi Yu, et al. *Architecture and Evaluation of an Asynchronous Array of Simple Processors,*

Journal of Signal Processing Systems Volume 53, Number 3, pages 243-259, 2008

Webographie

[Alt11] Altera : www.altera.com

[Alt02] Altera Documentation *Avalon Bus Specification – Reference Manual* http://www.altera.com/literature/manual/mnl_avalon_bus.pdf, July 2002.

[Cad] Cadence: HYPERLINK http://www.cadence.com

[Cel] Celoxica : HYPERLINK http://www.celoxica.com

[Kar03] Karnofsky K. *Signal-processing system design tackles tough wireless apps* http://www.commsdesign.com/, août 2003.

[Ouh03] P. G.-Ouhamou, C. Belleudy, M. Auguin, *Dynamic Voltage Scaling: implementations during the scheduling step of a codesign tool* http ://www.same-conference.org/technical.htm, SAME 2003.

[Ste10] Steuard J. An Introduction to Lagrange Multipliers http://www.slimy.com/~steuard/teaching/tutorials/Lagrange.html, 2010.

[Sys] SystemC : www.systemc.org

[Syn] Synopsys: www.synopsys.com

Résumé
Actuellement on assiste à une émergence de systèmes multimédias électroniques embarqués destinés à un large public d'utilisateurs. Leurs fonctionnalités sont de plus en plus complexes et diversifiés. Cependant, les systèmes sur puce doivent fonctionner dans des conditions souvent difficiles fluctuation des conditions de transmission en réseau, ressources d'énergie limitée, contraintes imposées par l'utilisateur etc. Tous ces paramètres « dynamiques » ne sont pas tenus en compte dans les méthodes classiques de co-design existantes.
Dans cette thèse une nouvelle approche multi-niveaux combinant l'adaptation au niveau système d'exploitation, applicatif et architectural a été proposée. En fait, c'est une approche originale et générique d'adaptation qui comporte essentiellement deux gestionnaires (global manager and local manager). Le gestionnaire global peut intervenir dans les trois couches afin de répondre aux grandes variations des contraintes du système (QoS et énergie). Le gestionnaire local intervient seulement dans les couches application et système d'exploitation. Il est mis en place pour contrôler le respect de la contrainte temps réel du système. Une étude de cas à été présentée sur un système réel en utilisant l'environnement de conception d'Altera et l'application de synthèse d'images 3D.

Mots clefs : système embarqué, qualité de service, auto-adaptation, énergie, temps réel

Abstract

The emergence of mobile and battery operated multimedia systems as well as the diversity of supported applications put new challenges in term of design efficiency. These systems must provide a maximum application quality of service (QoS) in the presence of a dynamically varying environment such as video streaming and multimedia conferencing and multiple resources constraints (e.g. battery level). These problems cannot be solved at design time and some efficiency gains can be obtained at run-time using an adaptive architecture.

In this thesis we propose a new cross layer adaptation solution for embedded mobile systems. It supports application QoS under real time and lifetime constraints via coordinated adaptation in the hardware, OS, and application layers. Our method relies on an original middleware solution implemented on a global and a local manager. The global manager (GM) handles large and long-term variations whereas the local manager (LM) is used to guarantee real time constraints. The GM acts in three layers, whereas the LM manages application and OS layers only. The main role of GM is to select the best configuration for each application to meet system constraints and respect user preferences. The approach has been applied to a 3D graphics application and successfully implemented on an Altera FPGA.

Key words: embedded system, quality of service, auto-adaptation, power, real time

الخلاصة
حاليا أصبح استعمال الأنظمة الإلكترونية المدمجة ومتعددة الوسائط جزءا لا يتجزأ من الحياة اليومية لجمهور واسع من المستخدمين . وباتت وظائفها أكثر تعقيدا وتنوعا .ومع ذلك، يجب عليها العمل في ظروف غالبا ما تكون صعبة كشبكة الاتصالات المتقلبة، وموارد الطاقة المحدودة، والقيود المفروضة من قبل المستخدم. كل هذه المعايير "الديناميكية" ليست مدروسة في أساليب التخطيط التقليدية لهذه الأنظمة.
في هذه الأطروحة قدمنا نهجا جديدا يجمع التكيف الذاتي على عدة مستويات: نظام التشغيل، البرنامج وهندسة النظام. هذا المنهج الجديد يعتمد على استعمال متصرفين (متصرف عام و متصرف محلي). يمكن للمتصرف العام التدخل في جل الطبقات لتلبية التباين الواسع من قيود النظام كجودة الخدمة والطاقة. يمكن للمتصرف المحلي التغيير فقط في طبقة البرامج ونظام التشغيل ولقد تم إعداده لرصد الامتثال لقيود الوقت الآني للنظام .وقد تمت دراسة حالة عن نظام حقيقي باستخدام بيئة تصميم ألتيرا وبرنامج ثلاثي الأبعاد.

الكلمات المفاتيح: الأنظمة المدمجة, جودة الخدمة, التكيف الذاتي, الطاقة, الوقت الآني