République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université Mouloud MAMMERI, Tizi-Ouzou

Faculté de Génie Electrique et d'Informatique
Département d'Automatique

Mémoire de Fin d'Etudes

En vue de l'obtention du diplôme

Master académique en automatique
Option commande des systèmes

Thème

Ordonnancement en temps réel de tâches optimisant la consommation énergétique d'un processeur

Dirigé par :
- Mr KARA Redouane

Proposé et présenté par :
- Mr BOUAYAD Nacer
- Mr RAMDANI Boualem

Soutenu le : 11/09/2012

Promotion 2012

Ce travail a été préparé au laboratoire de conception et conduite des systèmes de production de l'UMMTO

Remerciements

Avant tout, nous remercions Dieu de nous avoir donné le courage et la force nécessaires pour réaliser ce travail.

Nous tenons à remercier vivement notre promoteur Mr R.KARA de nous avoir fait confiance et de nous avoir encadré dans ce thème, sa générosité scientifique et sa disponibilité pour la recherche nous ont permis d'avancer dans notre travail et de surmonter avec lui toutes les difficultés.

Nos remerciements vont aussi à Mr A.MAIDI pour ses conseils judicieux et son aide dans ce travail.

Nous remercions tous les professeurs qui nous ont épaulés pendant ces années.

Nous tenons à remercier le président et les membres de Jury pour l'intérêt qu'ils ont porté à notre travail.

Nous ne saurions terminer sans remercier nos amis qui ont contribué de près et de loin afin d'accomplir ce travail.

Dédicaces

Ce travail, et bien au-delà de mon espérance, je le dois à mes très chers parents qui m'ont fourni au quotidien un soutien et une confiance sans faille et de ce fait, je ne saurais exprimer ma gratitude seulement par des mots.

Que dieu vous protège et vous garde pour nous.

- A mes chers frères Mohand et Nacer

- A mes chères sœurs Fazia et Fettouma, ainsi que leurs maris

- A mes neveux et nièces adorés

- A ma chère Nadjet

- A mes amis et tous les gens qui m'aiment

Je dédie ce travail.

BOUALEM

Je dédie ce travail à :

Mes très chers parents, à vous que je ne peux pourrait assez exprimer mon éternel amour, respect et gratitude. Pour votre amour, vos sacrifices, patience et tendresse.

Que dieu vous protège et vous garde pour nous.

- A mes chers frères et leurs femmes

- A mes chères sœurs et leurs maris

- A mes neveux et nièces adorés

- A ma précieuse Hanane, les mots ne peuvent résumer ma reconnaissance et mon amour à ton égard.

- A tous mes amis avec lesquels j'ai partagé mes moments de joie et de bonheur

NACER

Que toute personne ayant aidée de près ou de loin, trouve ici l'expression de nos reconnaissances.

Table des matières

Chapitre II : La consommation d'énergie dans un processeur et technique de réduction

Introduction

Chapitre III : Application à la minimisation d'énergie d'un système à faible puissance

Introduction générale

Introduction générale

Aujourd'hui, les systèmes informatiques temps réel embarqués sont présents dans de nombreux secteurs d'activités : transport terrestre (automobile, ferroviaire, etc.) et aérien, spatial, militaire, robotique, télécommunication, contrôle des systèmes automatisés de production, etc.

On qualifie de temps réel tout système informatique dont le fonctionnement est conditionné par des contraintes temporelles. La maîtrise des aspects temporels dans un système temps réel est un problème non encore complètement maitrisé et qui constitue un enjeu important en termes de recherche. Pour un système temps réel un résultat juste, mais ne respectant pas ses contraintes temporelles, est un résultat inutilisable qui correspond à une faute temporelle. La présence de contraintes temporelles dans ces systèmes est l'aspect fondamental qui les distingue des systèmes classiques. De plus pour certains de ces systèmes temps réel le non respect des contraintes peut avoir des conséquences catastrophiques en termes de pertes humaines, d'écologie, etc.

Les systèmes temps réel embarqués ont comme particularité d'être intégrés dans un équipement généralement mobile, comme une automobile, un avion, un téléphone, etc., ce qui les distingue des systèmes informatiques classiques. De tels systèmes ont en général des ressources limitées qu'il convient d'exploiter au mieux. De manière générale les systèmes temps réel embarqués conduisent à minimiser des ressources : puissance de calcul, mémoire, consommation électrique, etc. Ces systèmes sont souvent réalisés avec des architectures comportant plusieurs processeurs distribuées (multiprocesseur). Ceci est dû d'une part, au besoin de puissance de calcul qu'un seul processeur (monoprocesseur) ne peut fournir et d'autre part, à un besoin de modularité et de flexibilité, à la fois lors de la conception système pour réutiliser et faire évoluer le nombre de processeurs utilisés et à la fois pour rapprocher les capteurs et les actionneurs des processeurs qui les traitent. Cela afin de minimiser le câblage qui est un élément très coûteux dans une architecture multiprocesseur, et aussi de minimiser les perturbations électromagnétiques.

Au cours des trente dernières années, l'informatique temps réel s'est progressivement établie comme une discipline à part entière qui rassemble une forte communauté issue à la fois du monde académique et de l'industrie. Elle conduit principalement à traiter des problèmes d'ordonnancement temps réel. En effet chacun des processeurs de l'architecture étant séquentiel il s'agit de déterminer dans quel ordre et à quels instants on va devoir exécuter les fonctionnalités du système, concrétisées par des programmes séquentiels que l'on appelle des tâches. Cette dénomination est utilisée dès que ces programmes sont caractérisés temporellement (période, échéance, etc.). Le choix de cet ordre peut se faire avant l'exécution du système (hors ligne) ou bien pendant l'exécution du système (en ligne).

Afin de répondre aux problèmes d’optimisation il existe des algorithmes qui consistent à identifier le processeur qui conduira à la meilleure performance possible de système selon la nature des tâches à exécuter.

L'algorithme d'ordonnancement adopte des politiques d'optimisation qui visent à rentabiliser les temps de calcul et de rapidité d'exécution. Indirectement, le rendement énergétique du calculateur (nombre de tâches exécuté par rapport à l'énergie globale consommée) s'en voit valorisé, mais ce n'est pas pour autant que la consommation d'énergie est diminuée. En effet la consommation électrique d'un composant se caractérise par le produit de sa consommation énergétique instantanée et d'un temps. Actuellement, les fonctionnalités implémentées par les algorithmes d'ordonnancement visent à diminuer la consommation énergétique globale.

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

Il existe naturellement des techniques dites hybrides basées sur une collaboration entre composants matériels et logiciels. Par exemple des stratégies de mise en veille plus ou moins profondes des composants, ou d'adaptation du voltage du processeur, et donc de la fréquence, au besoin courant de l'application en termes de performances. Ces techniques permettent des réductions de consommation importantes car, sans considérer la puissance statique, l'énergie consommée varie au minimum en le carré du voltage dans les technologies CMOS actuelles.

Notre objectif est d'appliquer la technique d'adaptation dynamique de la tension et de la fréquence appelée DVS pour minimiser la consommation énergétique d'un processeur (circuit CMOS), tout en respectant les contraintes temporelles (date d'activation, temps de traitement, échéance…) des tâches à exécuter. Ce problème est modélisé comme un problème d'optimisation avec contraintes. Pour résoudre ce problème on a proposé un programme sous Matlab qui permet l'évaluation de la consommation d'énergie en fonction du temps de traitement des tâches.

Structure du mémoire

Le chapitre 1 présente les notions de systèmes temps réel et leurs classes. Ainsi que la théorie d'ordonnancement temps réel et ces différentes classes. Comme nous étudions quelques algorithmes d'ordonnancement et leurs conditions d'ordonnançabilité et faisabilité.

A la fin on a présenté un logiciel de simulation d'ordonnancement appelé *Cheddar* qui permet aussi de tester le respect des contraintes temporelles de jeu de tâche.

Le chapitre 2 présente l'origine de la consommation énergétique dans les processeurs (les circuits CMOS) et les techniques de réduction de cette consommation, en se basant sur la technique d'adaptation dynamique de la tension et de la fréquence (DVS).

Le chapitre 3 présente une technique d'adaptation dynamique de la tension et de la fréquence appelée DVS pour minimiser la consommation énergétique d'un processeur (circuit CMOS), tout en respectant les contraintes temporelles (date d'activation, temps de traitement, échéance…) des tâches à exécuter. Ce problème est modélisé comme un problème d'optimisation avec contraintes. Pour résoudre ce problème on a proposé un programme Matlab qui permet de calculer les temps de traitement et l'évolution de la consommation d'énergie en fonction de ce temps.

Chapitre I

Introduction

En industrie, on parle d'un système temps réel lorsque ce système contrôle (ou pilote) un procédé physique à une vitesse adapté à l'évolution du procédé contrôlé. Ces systèmes se différencient des autres systèmes par la prise en compte temporelles, dont le respect est aussi important que l'exactitude du résultat. Autrement dit, le système ne doit pas simplement délivrer des résultats exacts, il doit les délivrer dans un délai imposé.

Nous nous concentrons sur une partie fondamentale d'un système temps-réel : l'ordonnanceur. Pour chaque processeur l'ordonnanceur est en charge de définir l'ordonnancement, qui est une séquence infinie d'éléments de type : (date, contrainte temporelle et ressource). Chaque élément correspond à l'élection de la tâche à exécuter, puis à un changement de contexte, qui fait suite à une décision d'ordonnancement par l'ordonnanceur.

après avoir présenté les classes et types de ces problèmes d'ordonnancement, on a discuté quelques algorithmes qui conduisent à la meilleure performance possible du système selon le type de tâches.

Pour calculer les différents critères de performances (contraintes temporelles, dimensionnement de ressources), on a présenté un logiciel de simulation d'ordonnancement appelé CHEDDAR, qui permet aussi de tester le respect des contraintes temporelles de jeu de tâche.

I.1 Qu'est ce qu'un système temps réel ?

Les systèmes temps réels sont de plus en plus présents dans le quotidien, on les trouve dans l'aéronautique, transport ferroviaire, l'automobile, l'électroménager ou le multimédia.

On désigne de temps réel, toute application mettant en œuvre un système informatique dont le comportement (fonctionnement) est conditionné par l'évolution dynamique de l'état d'un environnement (appelé procédé) qui lui y'est connecté et dont il doit contrôler le comportement [01]. Le rôle du système informatique est alors de suivre ou de piloter ce procédé en respectant des contraintes temporelles définies dans le cahier des charges de l'application.

Cette définition implique, que la seule rapidité moyenne d'exécution du logiciel ne conditionne pas la validité du système, mais des contraintes temporelles doivent être respectées (par exemples les échéances des traitements) sont relatives à un temps physique mesurable, et font partie de la spécification du système à implanter.

Dans les systèmes temps réel, le système informatique doit réagir en permanence aux variations du procédé et agir en conséquence sur celui-ci afin d'obtenir le comportement ou l'état souhaité. Cette caractéristique définit la notion de réactivité du système informatique

vis-à-vis du procédé (environnement auquel il est connecté). Une définition des systèmes réactifs, décrivant le fonctionnement d'un système temps réel (l'interaction entre le système informatique et l'environnement), est donnée dans [02] : Un système réactif est un système qui réagit continument avec son environnement à un rythme imposé par cet environnement. Il reçoit, par l'intermédiaire de capteurs, des entrées provenant de l'environnement, appelées stimuli, réagit à tous ces stimuli en effectuant un certain nombre d'opérations et produit, grâce à des actionneurs, des sorties utilisables par l'environnement, appelées réactions ou commandes.

De cette définition, un système temps réel est alors composé principalement de deux éléments distincts : une ou plusieurs entités physiques constituent le procédé (le système contrôlé) et un système de contrôle (contrôleur) qui est chargé de surveiller de manière régulière, périodique, l'état du procédé en récupérant les valeurs en provenance des capteurs. En fonction de l'algorithme de contrôle et les valeurs décrivant l'état actuel du procédé, le contrôleur commande les changements d'état, via des actionneurs. Notons qu'un système temps réel est dit *embarqué* lorsqu'il est situé à l'intérieur de l'environnement qu'il doit contrôler, comme un calculateur dans une voiture ou un avion.

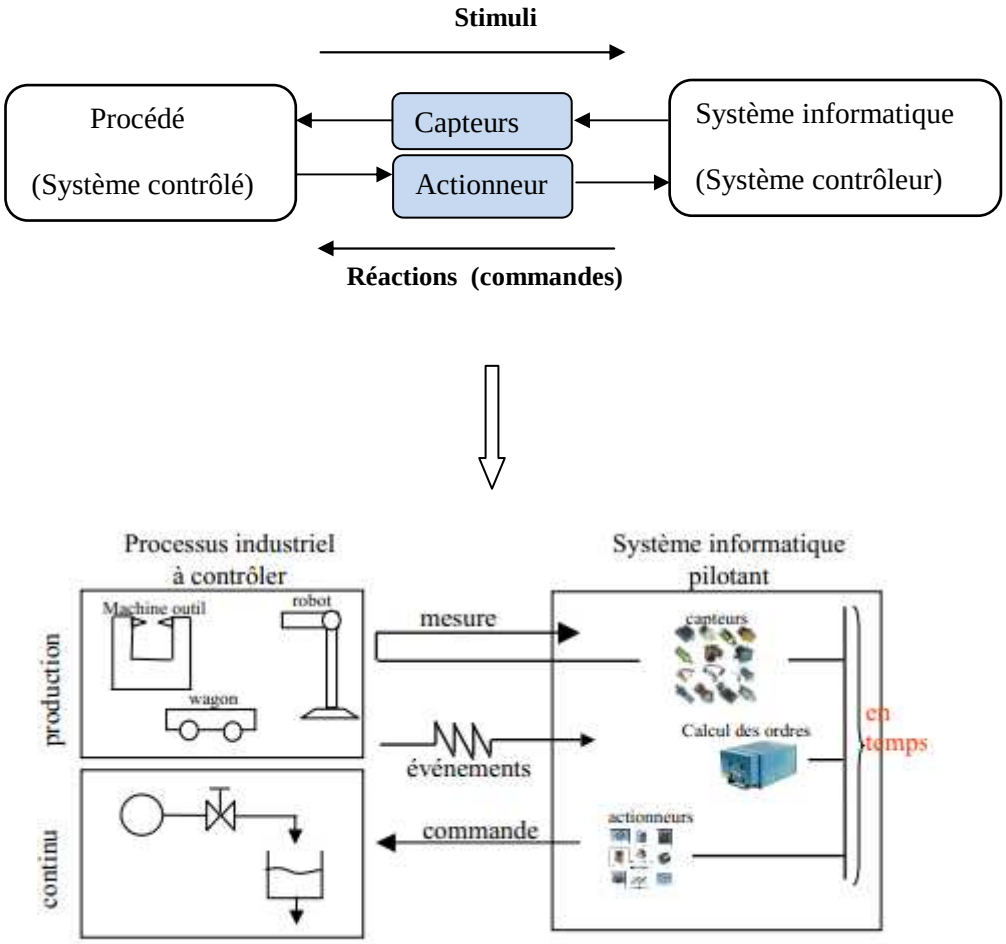


Figure I.1 : Système temps réel

I.2 Quelques exemples

–Commande et contrôle d'une ligne de production

–Guidage d'un système mobile (robotique…)

–Systèmes embarqués (ferroviaire, aéronautique, automobile…)

–Surveillance des réactions ou phénomènes physiques (nucléaire, chimie,…)

–Instrumentations et opérations assistées par ordinateur (médical…)

–Systèmes de communication (multimédia…)

–Systèmes dédiés (conduite d'expérience scientifique, traitement de signal…)

I.3 Classe des systèmes temps réels

En fonction de la criticité des contraintes temporelles, on distingue essentiellement deux types de systèmes temps réel :

1.3.1 Temps réel strict ou dur

La majorité des systèmes temps réel critiques est exclusivement constituée de traitements qui ont des contraintes temporelles strictes. C'est-à-dire que la condition indispensable de fonctionnement du système est que tous les traitements du système doivent impérativement respecter toutes les contraintes temporelles. On parle alors de traitements temps réel strict ou dur (hard en anglais). Ceci suppose deux choses :

- qu'on soit capable de définir les conditions de fonctionnement nominales en termes d'hypothèses sur l'environnement avec lequel le système interagit;

- qu'on soit capable de garantir la fiabilité du système avant son exécution, c'est à dire que tous les scénarios d'exécution possibles dans ces conditions, respecteront leurs contraintes temporelles. Ceci suppose, qu'on peut extraire ou disposer de suffisamment d'informations sur le système pour déterminer tous les scénarios possibles.

I.3.2 Temps réel souple

Une autre classe de systèmes est moins exigeante quant au respect des contraintes temporelles. Les systèmes de cette classe, dits temps réel souple (soft en anglais), peuvent souffrir d'un taux acceptable de fautes temporelles de la part d'une partie des traitements (eux-mêmes dits temps réel souple), et sans que cela ait des conséquences catastrophiques.

Cette classe comprend, entre autres, les systèmes où la qualité est appréciée par les sens de l'être humain sous la forme d'un service: c'est le cas de systèmes et d'applications multimédia (téléphonie, vidéo, rendu visuel interactif par exemple). La mesure du respect des contraintes temporelles prend la forme d'une donnée probabiliste : la qualité de service relative à un service particulier (nombre d'images ou nombre d'échantillons sonores rendus par secondes, par exemple), ou relative au comportement du système dans son ensemble (nombre de traitements qui ont pu être rendus dans les temps, tous services confondus, par exemple), ou les deux combinés.

I.4 Les tâches en temps réel

I.4.1 Définition d'une tâche

Une tâche correspond une à entité exécutable placée sous le contrôle du système d'exploitation Temps Réel. Le système d'exploitation va lancer la tâche, la suspendre pour la mettre en attente d'une ressource et finalement l'arrêter.

I.4.2 Caractéristiques d'une tâche

Afin de répondre aux exigences temporelles du système temps réel, une tâche composant l'application de contrôle peut être exécutée une multitude fois durant la vie du système, comme par exemple une tâche qui doit lire régulièrement l'état des capteurs.

Nous appelons alors une instance d'une tâche une exécution ou occurrence de celle-ci. Parmi les paramètres temporels usuels définissant une tâche, nous citons :

- *Date d'activation* a_i : La date à laquelle la tâche τ_i commence son exécution. i.e. la première instance de τ_i est réveillée à la date a_i . Dans le cas où toutes les tâches d'un système sont réveillées au même instant ($a_i = a_2 = ... = a_n$), on dit que les tâches sont simultanées (ou elles sont synchrones).

- *Temps d'exécution* T_i (charge processeur) : Le temps processeur requis par chaque instance de τ_i. Généralement, ce paramètre est le pire (borne supérieure) temps d'exécution de cette tâche (Worst-Case Execution Time WCET) sur le processeur

auquel elle est affectée. Notons que certains systèmes temps réel souple considèrent un temps d'exécution moyen, ou minimal.

- *Période d'activation* P_i : La durée de temps fixe ou minimale entre deux activations de deux instances successives de τ_i .Suivant la nature de la tâche (périodique par exemple) on peut calculer facilement les dates d'activation de τ_i à travers le temps.

- *Délai critique* D_i **(**échéance**)** : Le temps alloué à la tâche pour terminer complètement son exécution. Chaque instance de τ_i doit terminer son exécution avant D_i unités de temps après la date de son activation. Le dépassement de cette date limite d'exécution produit une faute temporelle.

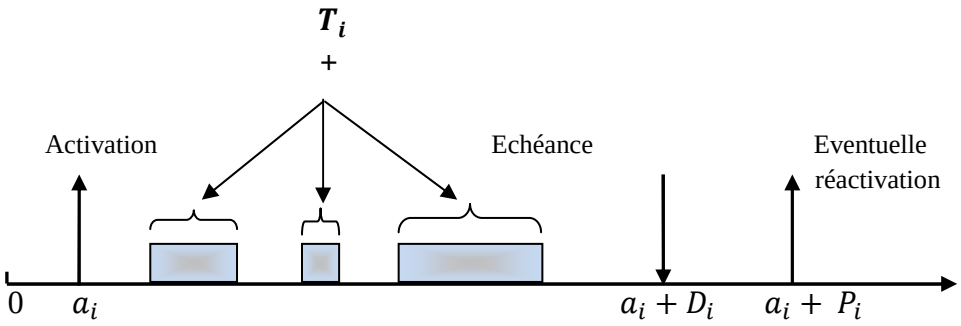


Figure I.2 : Modèle usuelle d'une tâche temps réel.

I.4.3 Nature des tâches

Une tâche peut s'exécuter à des intervalles réguliers (tâche périodique) ou de manière aléatoire (tâche non périodique).

I.4.3.1 Tâches périodiques

On dit que la tâche τ_i est périodique de période $P(\tau_i)$ si l'événement qui conditionne l'activation de cette tâche se produit à des intervalles de temps réguliers $P(\tau_i)$. Donc, le service fourni par cette tâche périodique est rendu indéfiniment. Pour cette raison, chacune de ses exécutions est appelée instance.

On distingue deux types de tâche périodique :

- *Périodique classique* : Ce type de période est la plus utilisée dans les applications temps réel. Chaque instance de la tâche τ_i doit s'exécuter entièrement à l'intérieur de l'intervalle de longueur $P(\tau_i)$ qui démarre à la date d'activation de cette instance (Figure I.3).

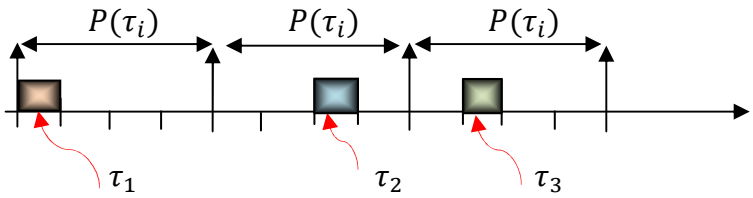


Figure I.3 : périodicité classique

- *Période stricte* : les instances d'une tâche périodique stricte τ_i doivent démarrer exactement au début de chaque intervalle de longueur(τ_i) .

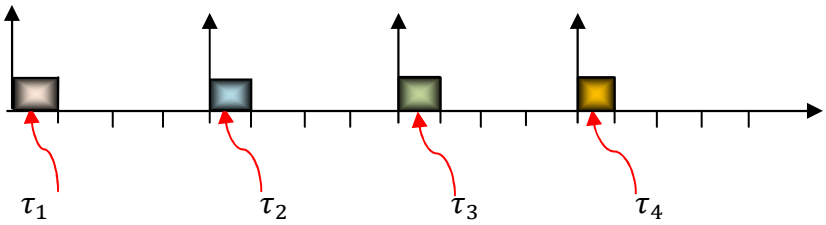


Figure I.4 : périodicité stricte

I.4.3.2 Tâche non périodique

Il existe deux types de tâches non périodiques :

- *Apériodique* : les dates d'activations sont aléatoires et ne peuvent être anticipées. Son exécution est le produit d'événements internes ou externes qui peuvent se déclencher à tout instant.

- *Sporadique* : c'est un cas particulier des tâches apériodiques où une durée de temps minimale sépare deux activations successives. Pour les prendre en compte, ces tâches sont souvent considérées comme tâches périodiques.

I.4.4 Etats d'une tâche

Au cours de la vie de l'application temps réel, plusieurs instances d'une tâche peuvent êtres exécutées. A un instant donné, une instance peut être dans l'un des états d'exécutions présentées dans la (figure I.5) qui montre aussi les passages possibles d'un état à un autre.

Une tâche (supposée déjà créée) est initialement dans l'état endormi (au repos). Quand elle se réveille, elle passe en état prêt. Dans cet état, l'instance de la tâche est activée mais en attente d'être sélectionnée par l'ordonnanceur pour l'exécution.

Lorsque l'ordonnanceur le décide, suivant la politique d'ordonnancement choisie, la tâche se verra allouer le processeur afin d'être exécutée (état Exécution). Une tâche en cours d'exécution peut être:

- Interrompue par une autre tâche plus prioritaire et passe en état prêt ;
- Se mettre en attente d'un message, d'une date, d'un évènement, ou bien de l'accès à une ressource ;
- Etre suspendue et passer en état endormi.

Une tâche en état d'attente d'une condition (arrivée d'un message, d'une date, d'un évènement ou libération d'une ressource) devient prête à exécuter une fois la condition est vérifiée.

La figure suivante nous montre les différents états d'une tâche temps réel.

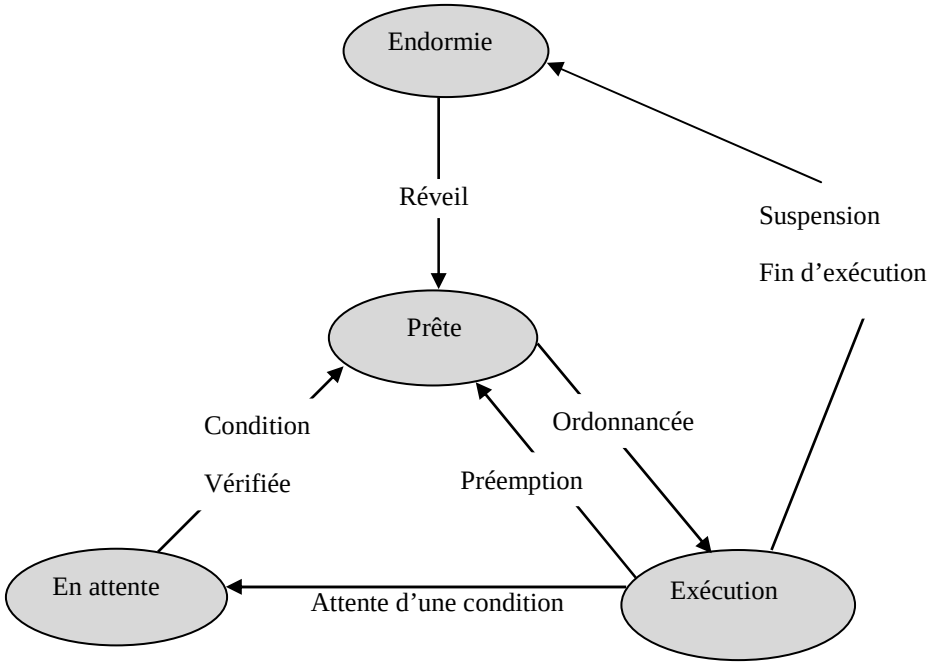


Figure I.5: Différents états possibles d'une tâche temps réel.

I.5 Problème d'ordonnancement des tâches temps réel

Un problème d'ordonnancement consiste à organiser dans le temps la réalisation de tâches, compte tenu des contraintes temporelles (délais, contraintes d'enchaînement) et de contraintes portant sur la disponibilité des ressources requises.

En production (manufacturière, de biens, de service), on peut le présenter comme un problème où il faut réaliser le déclenchement et le contrôle de l'avancement d'un ensemble de commandes à travers les différents centres composant le système.

Un ordonnancement constitue une solution au problème d'ordonnancement. Il est défini par le planning d'exécution des tâches (« ordre » et « calendrier ») et d'allocation des ressources et vise à satisfaire un ou plusieurs objectifs. Un ordonnancement est très souvent représenté par un diagramme de Gantt.

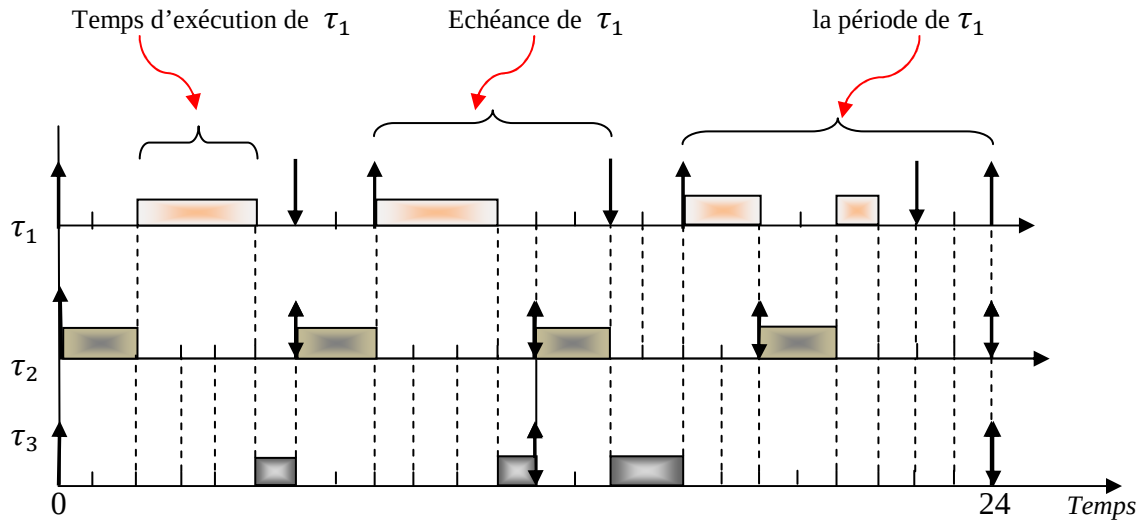


Figure I.6 : Diagramme de Gantt représentant une séquence d'ordonnancement d'une configuration de 3 tâches τ_1 (3, 6, 8) , τ_2(2, 6, 6) , τ_3(2, 12, 12) .

I.5.1 Les objectifs d'ordonnancement

Dans la résolution d'un problème d'ordonnancement, on peut choisir entre deux grands types de stratégies, visant respectivement à l'optimalité des solutions, ou plus simplement à leur admissibilité.

L'approche par optimisation suppose que les solutions candidates à un problème puissent être ordonnées de manière rationnelle selon un ou plusieurs critères d'évaluation numériques, construits sur la base d'indicateurs de performances. On cherchera donc à minimiser ou maximiser de tels critères. On note par exemple ceux : liés à une énergie ou un débit ; liés aux coûts de lancement, de production, de transport, etc., on distingue aussi ceux :

- Liés au temps : le temps total d'exécution ou le temps moyen d'achèvement d'un ensemble de tâches le stock d'en-cours de traitement différents retards (maximum, moyen, somme, nombre, etc.) ou avances par rapport aux dates limites fixées (deadline);

- Liés aux ressources : la quantité totale ou pondérée de ressources nécessaires pour réaliser un ensemble de tâches avec la charge de chaque ressource ;

Dans le chapitre trois, on présente un problème d'optimisation, qui minimise la consommation d'énergie sous contraintes temporelles (temps de traitement).

I.5.2 Les contraintes

Les contraintes expriment des restrictions sur les valeurs que peuvent prendre simultanément les variables de décision. On distingue deux types de contraintes : les contraintes temporelles et les contraintes de ressource.

I.5.2.1 Contraintes temporelles

Les contraintes temporelles que le système temps réel doit supporter sont de deux natures :

- *strictes* : les échéances ne doivent sous aucun prétexte êtres dépassées sous peine de conséquences graves pour l'être humain ou le système lui-même.
- *relatives* : les conséquences d'un dépassement d'échéance sont quantifiables et supportables pour le système dans une certaine mesure.

Le non respect de ces contraintes est appelé une faute temporelle.

I.5.2.2 Contraintes de ressources

La ressource est un moyen technique ou humain destiné à être utilisé pour la réalisation d'une tâche et disponible en quantité limitée, sa capacité.

Plusieurs types de ressources sont à distinguer. Une ressource est renouvelable si après avoir été allouée à une ou plusieurs tâches, elle est à nouveau disponible en même quantité (les hommes, les machines, l'équipement en général); la quantité de ressource utilisable à chaque instant est limitée. Dans le cas contraire, elle est consommable (matières premières, budget) ; la consommation globale (ou cumul) au cours du temps est limitée.

Qu'elle soit renouvelable ou consommable, la disponibilité d'une ressource peut varier au cours du temps. Sa courbe de disponibilité est en général connue a priori, sauf dans les cas où elle dépend du placement de certaines tâches.

On distingue par ailleurs principalement dans le cas de ressources renouvelables les ressources disjonctives qui ne peuvent exécuter qu'une tâche à la fois (machine-outil, robot manipulateur) et les ressources cumulatives qui peuvent être utilisées par plusieurs tâches simultanément mais en nombre limité (équipe d'ouvriers, poste de travail).

I.5.3 Classe des problèmes d'ordonnancement temps réel

I.5.3.1 Hors ligne (off line)

Un ordonnancement hors-ligne est effectué avant le lancement du système. Ceci implique que tous les paramètres des tâches soient connus a priori, et notamment les dates d'activation. L'implémentation du système est alors très simple : il suffit de stocker l'identifiant de la tâche à effectuer à chaque moment d'ordonnancement dans une table, l'ordonnanceur exploitant ensuite cette information pour sélectionner la tâche à exécuter.

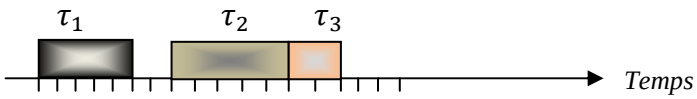


Figure I.7 : Exemple d'ordonnancement hors ligne

I.5.3.2 En ligne (on line)

Un ordonnancement en ligne est effectué durant le fonctionnement du système. L'ordonnanceur recalcule un nouvel ordonnancement à chaque fois qu'une nouvelle tâche est activée. L'avantage de cette approche est la flexibilité, et l'adaptabilité à l'environnement que procure le calcul en ligne.

I.5.3.3 Préemptif (interruptible)

L'exécution d'une tâche peut être interrompue par une autre tâche plus prioritaire .Son exécution est reprise ultérieurement. Notons qu'une tâche demandant une ressource critique, occupée par une tâche moins prioritaire, doit attendre que cette dernière la libère (terminer l'utilisation de la ressource) afin de continuer son exécution.

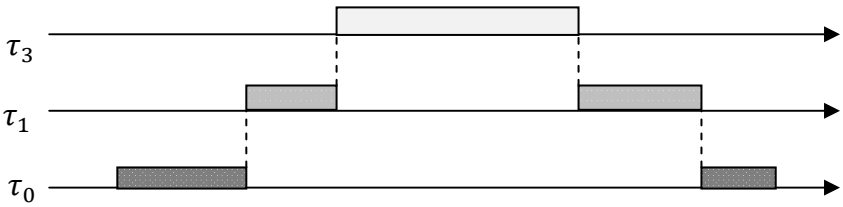


Figure I.8 : Exemple d'un ordonnancement préemptif

I.5.3.4 Non préemptif

L'ordonnanceur ne peut pas interrompre l'exécution d'une tâche (possédant le processeur) en faveur d'une autre tâche. Il doit attendre jusqu'à la terminaison de l'exécution de la tâche en cours, avant de débuter l'exécution de toute autre tâche. Si une tâche non-préemptive est interrompue, son exécution doit être repris de nouveau depuis le début.

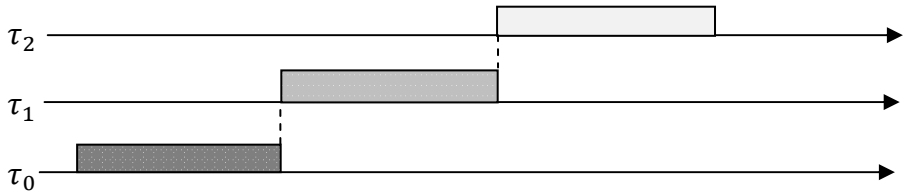


Figure I.9 : Exemple d'un d'ordonnancement non préemptif

I.5.3.5 Statique/dynamique

Un ordonnancement statique (prédictif) est basé uniquement sur les paramètres des tâches au départ (avant l'activation). À l'inverse, un ordonnancement dynamique (réactif) est basé sur les paramètres des tâches qui varient au cours de l'exécution.

I.5.3.6 Optimal / non optimal

Un algorithme d'ordonnancement est dit optimal pour un problème donné s'il permet de trouver un ordonnancement qui respecte toutes les contraintes lorsqu'un tel ordonnancement existe. Si l'algorithme optimal ne trouve pas de solution alors aucun autre algorithme ne pourra le faire. Par ailleurs, un algorithme d'ordonnancement non-optimal vise à trouver des solutions approchées (en ignorant une partie des contraintes par exemple**)**.

I.5.4 Analyse d'ordonnançabilité

Pour avoir une certaine prédictibilité et pouvoir s'assurer du respect des contraintes temporelles, en particulier quand il s'agit d'un système temps réel strict, une analyse de l'ordonnancement doit être effectuée.

Ainsi l'étude de l'ordonnancement des systèmes temps réel se focalise principalement sur deux problèmes :

- *La faisabilité* : En considérant les tâches, leurs contraintes ainsi que les processeurs dont dispose l'architecture. Le but est de dire, s'il existe un ordonnancement qui satisfait toutes les échéances.

- *l'ordonnançabilité* : Le but est de déterminer s'il existe un ordonnancement, qu'on obtient avec un algorithme qui permet le respect de toutes les contraintes, en connaissant les natures des tâches et leurs contraintes (temporelles et ressources).

I.5.5 Algorithmes d'ordonnancement et condition d'ordonnançabilité

L'objectif d'un algorithme d'ordonnancement consiste à identifier le processeur qui conduira à la meilleure performance possible de système. Les algorithmes d'ordonnancement se divisent selon les critères en plusieurs types.

I.5.5.1 Algorithme RM (Rate Monotonic)

Cet algorithme a été introduit par Liu et Layland en 1973 [03]. Cet algorithme est utilisé uniquement pour les tâches périodiques. La tâche qui a la priorité la plus forte, est élue pour être exécutée. La priorité est l'inverse de la périodicité c.-à-d. la tâche ayant la période la plus petite, dans un jeu de tâches, est la plus prioritaire. C'est un algorithme optimal dans la classe des algorithmes, facile à implanter dans un système d'exploitation. Les tâches sont ordonnançables en mode préemptif ou non préemptif.

La condition suffisante d'ordonnançabilité avec l'algorithme RM pour un système de n tâches est la suivante :

$$\sum_{i=1}^{i=n} \frac{T(\tau_i)}{P(\tau_i)} \leq n\left(2^{\frac{1}{n}} - 1\right) \qquad \text{(I. 1)}$$

Tel que :

n : nombre de tâche

τ_i : une tâche donnée pour $i = 1\ldots\ldots n$

$T(\tau_i)$: temps de traitement de la tâche τ_i

$P(\tau_i)$: période d'exécution

Exemple : On considère trois tâches, pour chaque tâche on connait son temps de traitement et sa période. Elles sont données dans le tableau suivant.

Tableau I.1: Exemple Ordonnancement de trois tâches avec l'algorithme RM.

Tâche (τ_i)	$T(\tau_i)$	$P(\tau_i)$
τ_2	4	24
τ_1	3	8
τ_0	2	6

La figure suivante nous montre l'ordonnancement de trois tâches selon l'algorithme Rate Monotonic.

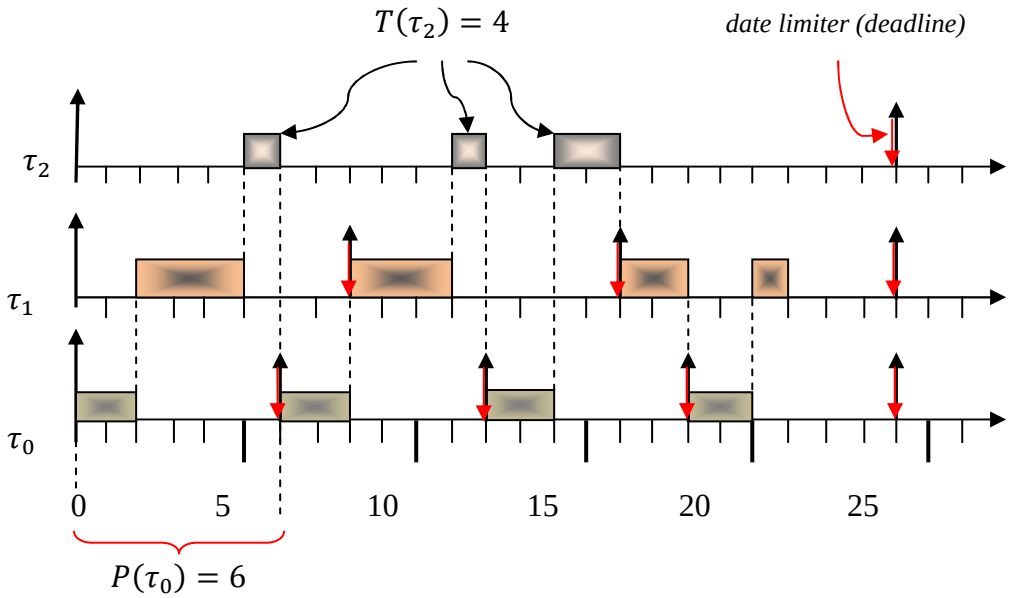


Figure I.10 : Exemple d'ordonnancement selon Rate Monotonic

1.5.5.2 Algorithme DM (Deadline Monotonic)

Cet algorithme a été introduit par Leung et Whiteheaden en 1982 [04]. C'est un algorithme à priorité statique ; la priorité d'une tâche est inversement proportionnelle à son échéance. Cet algorithme équivaut à RM quand l'échéance est égale à la période (échéance sur requête).

La condition suffisante d'ordonnançabilité de *n* tâches périodiques (triées par priorités décroissantes) $\{\tau_1, \dots \tau_i, \dots, \tau_n\}$ avec DM est :

$$\forall i: 1 \leq i \leq n : T(\tau_i) + \sum_{j=1}^{i-1} \left\lceil \frac{D(\tau_i)}{P(\tau_i)} \right\rceil . T(\tau_i) \leq D(\tau_i) \qquad (1.2)$$

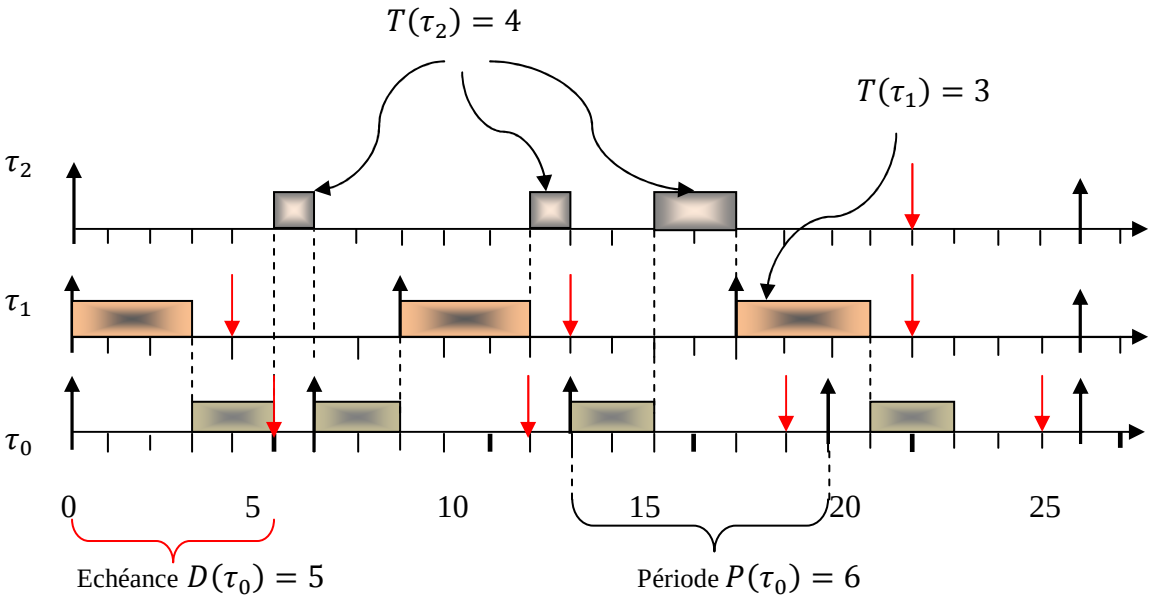


Figure I.11 : Exemple d'ordonnancement selon Deadline Monotonic

I.5.5.3 Algorithme EDF (Earliest Deadline First)

Cet algorithme a été introduit tout comme RM par Liu et Layland en 1973 [03]. C'est un algorithme à priorité dynamique, pour les tâches périodiques et apériodiques. Comme RM, les tâches sont ordonnançeables en mode préemptif et non préemptif. C'est également un algorithme optimal, difficile à implanter dans un système d'exploitation. Il est instable en surcharge et est moins déterministe que RM. Pour un jeu de tâches donné, on calcule l'échéance pour chacune d'elle et celle ayant la plus courte échéance sera élue.

La condition nécessaire et suffisante d'ordonnançabilité dans le cas préemptif si $\forall i: D(\tau_i) = P(\tau_i)$ (échéance sur requête) est :

$$\sum_{i=1}^{i=n} \frac{T(\tau_i)}{P(\tau_i)} \leq 1 \qquad \text{(I. 3)}$$

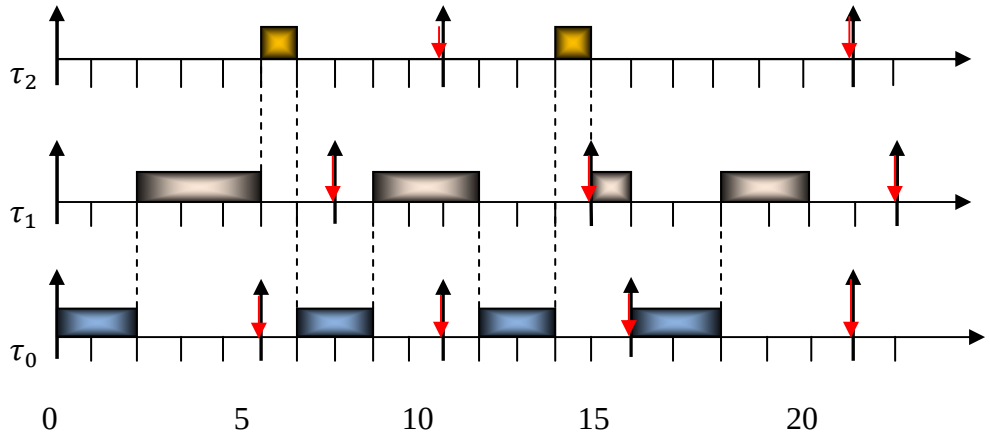


Figure I.12 : Exemple d'ordonnancement EDF

I.5.5.4 Algorithme LLF (Least-Laxity First)

Cet algorithme se base sur la laxité, il a été introduit par Mok et Dertouzos [05, 06]. C'est un algorithme à priorité dynamique qui traite des tâches périodiques pour lesquelles la préemption est autorisée. L'ouvrage [07] montre que les conditions d'ordonnançabilité pour l'algorithme LLF sont les mêmes que pour EDF. C'est-à-dire que la condition nécessaire et suffisante d'ordonnançabilité dans le cas préemptif si $\forall i: D(\tau_i) = P(\tau_i)$ (échéance sur requête) est :

$$\sum_{i=1}^{i=n} \frac{T(\tau_i)}{P(\tau_i)} \leq 1$$

Exemple : Les définitions suivantes permettent d'aborder la notion de laxité :

- $L = D - T$: sa laxité nominale. Indique le retard maximum que peut prendre la tâche sans dépasser son échéance.
- $D(t) = d - t$: son délai critique résiduel au temps *t*
- $T(t)$: sa durée d'exécution résiduelle au temps *t*
- $L(t) = D(t) - T(t)$: sa laxité résiduelle au temps *t*

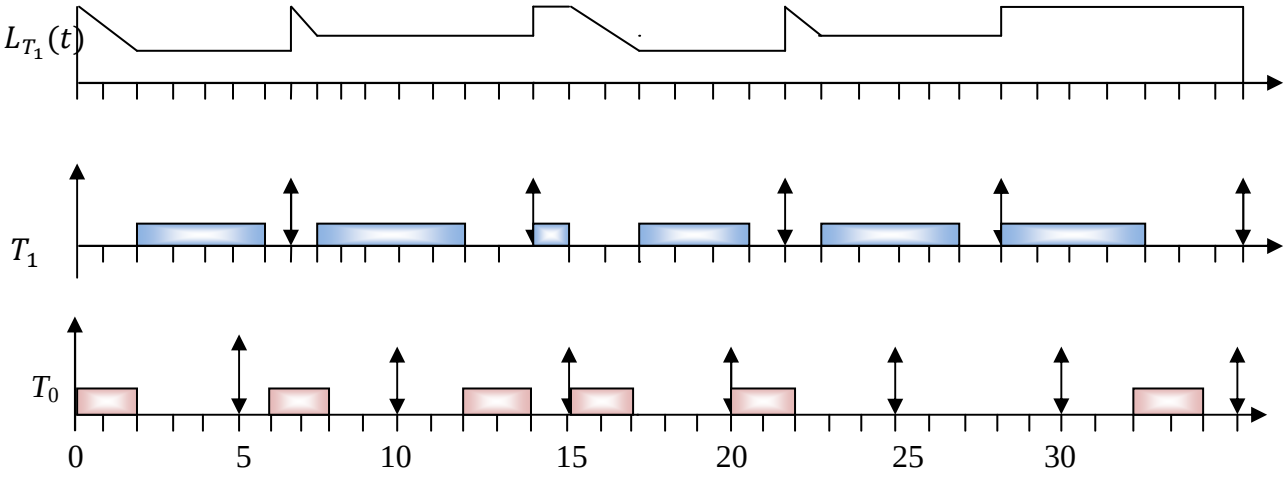


Figure I.13 : Exemple d'évolution de la laxité d'une tâche, pour un ordonnancement EDF

I.6 La simulation d'ordonnancement avec le logiciel Cheddar

I.6.1 Cheddar, c'est quoi ?

Cheddar est un outil de simulation d'ordonnancement permettant de calculer les différents critères de performance (contraintes temporelles, dimensionnement de ressources). L'outil permet, entre autre, de tester le respect des contraintes temporelles d'un jeu de tâches modélisant une application/un système temps réel.

Cheddar est principalement constitué de deux composants logiciels :

- Un éditeur qui permet à l'utilisateur de décrire l'application à analyser et sur lequel, les résultats de simulation seront présentés.
- Une bibliothèque comportant les principaux résultats de la théorie de 'ordonnancement temps réel.

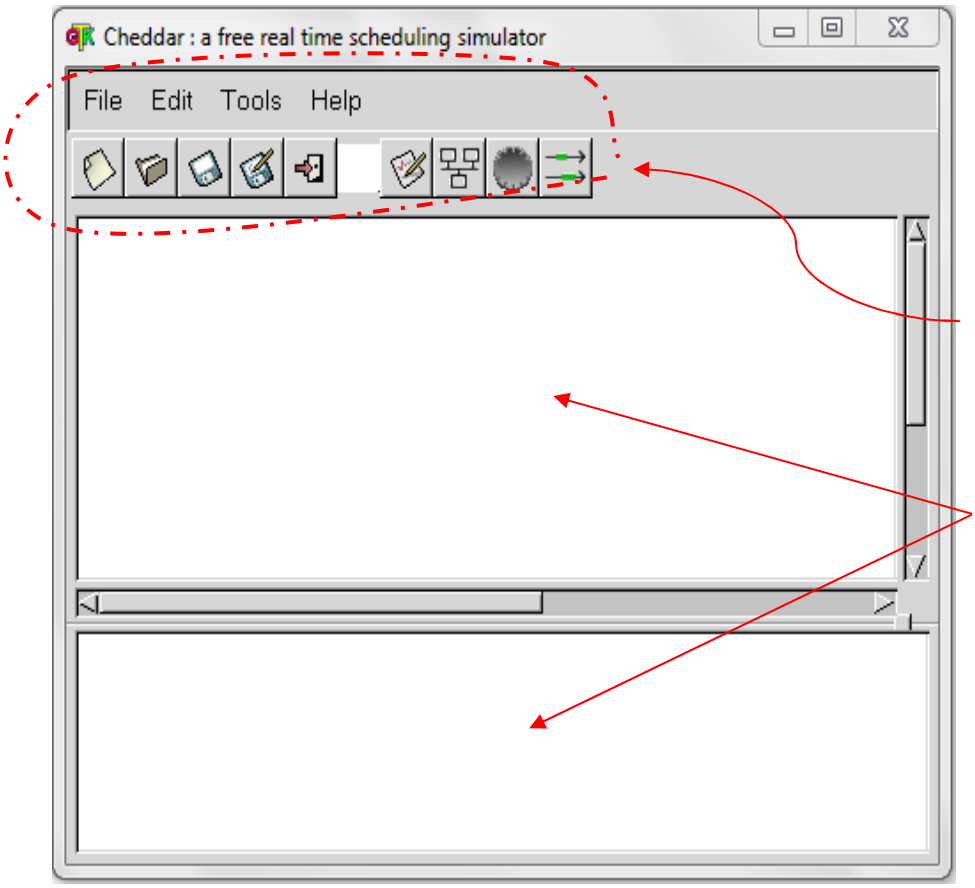


Figure I.14 : Capture de logiciel Cheddar

I.6.2 Comment ça marche ?

Cheddar fournit des outils pour vérifier des contraintes temporelles de tâches du temps. Ces outils sont basés sur les résultats classiques de théorie de la planification du temps. Avant d'appeler de tels outils, nous devons définir un système qui est composé principalement de plusieurs processeurs et tâches.

I.6.3 Exemple de simulation

Dans cette partie on va présenter un exemple d'ordonnancement de trois tâches quelconques selon l'algorithme EDF préemptif; leurs caractéristiques sont présentées dans le (tableau I.2). Avec Cheddar on peut tester aussi la faisabilité de l'algorithme.

Tableau I.2 : Exemple d'ordonnancement de trois tâches

	Capacité (T_i)	Deadline (D_i)	Période (P_i)
τ_1	3	7	20
τ_2	2	4	5
τ_3	1	8	10

Pour une meilleure simulation on doit procéder de la manière suivante :

Pour définir un processeur, nous choisissons le sous-menu "*Édit/Update processors*". La fenêtre suivante s'affiche:

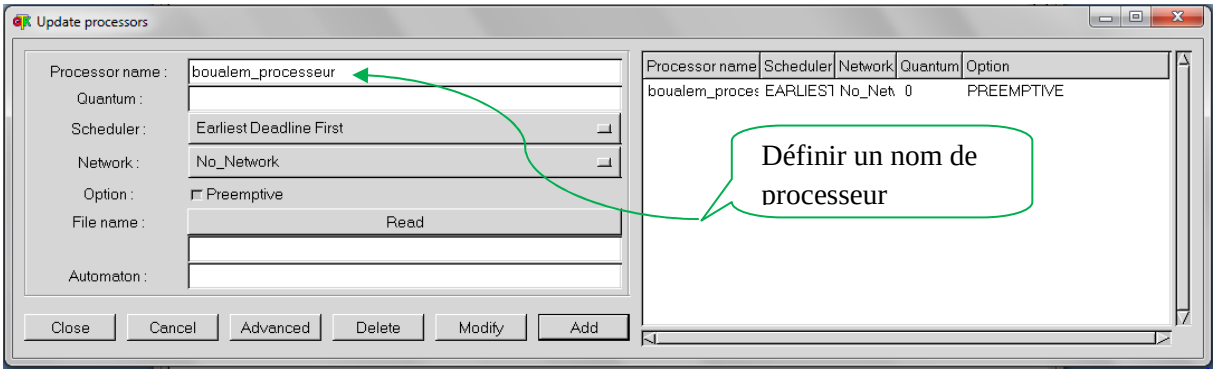


Figure I.15 : Appellation du processeur dans Cheddar

La prochaine étape pour élaborer une simulation, est de définir un espace d'adresse. Nous choisissons sous le menu "*Edit / Update adresse spaces*", un espace d'adresse qui contient des tâches ou des ressources, la fenêtre suivante s'affiche :

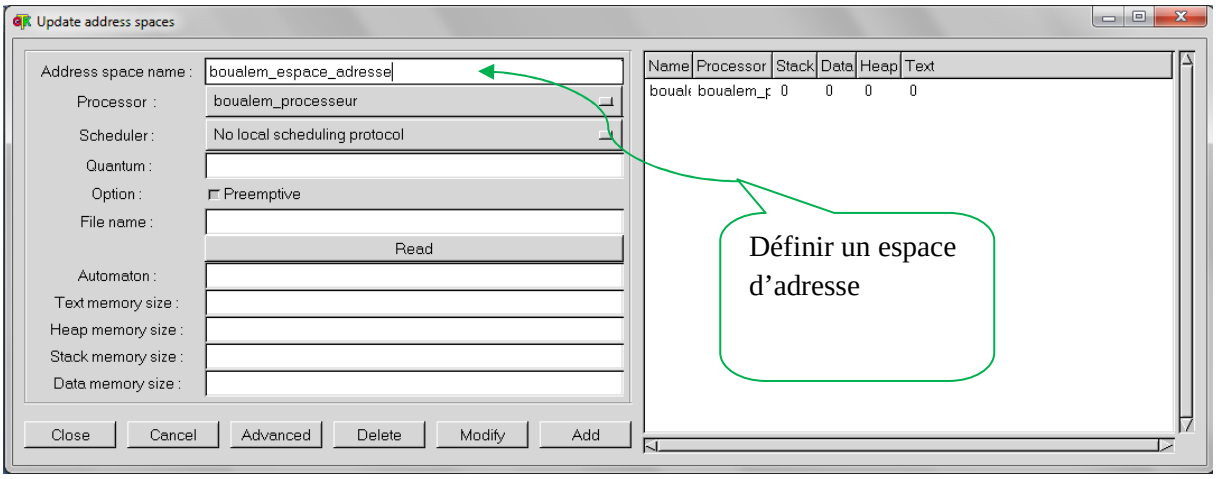
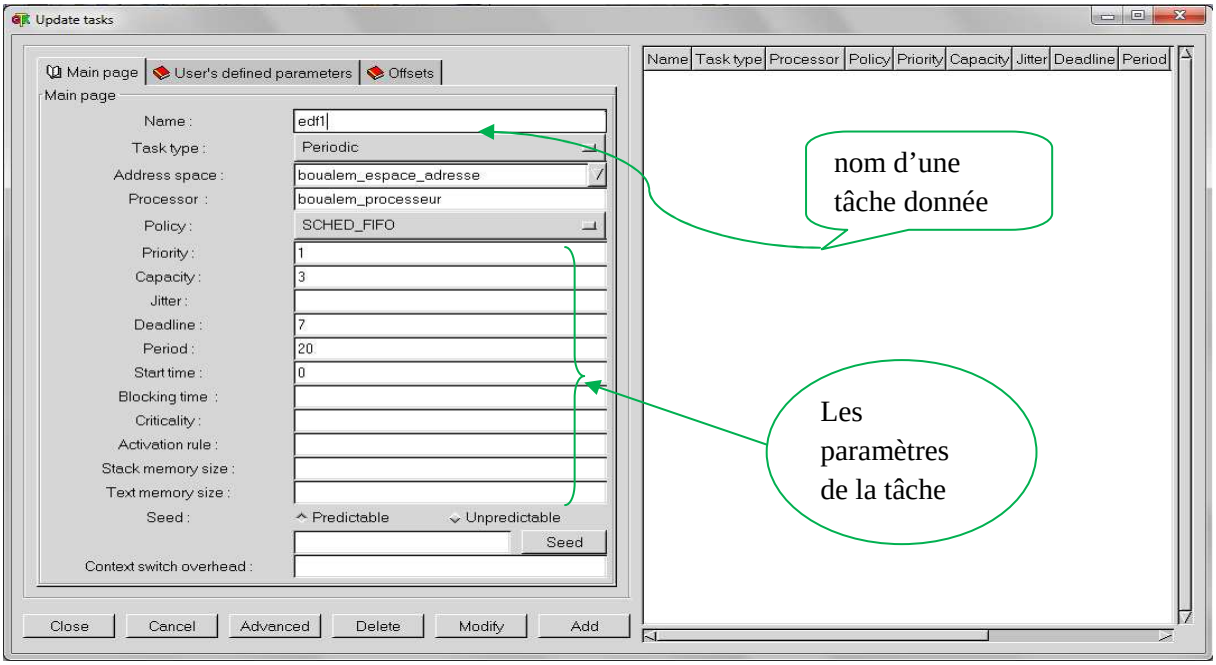


Figure I.16 : Création d'un espace d'adresse dans Cheddar

L'étape suivante nous permet de définir les caractéristiques des tâches à exécuter et le type de la politique d'ordonnancement à utiliser.

Pour faire on choisit le sous-menu "*Edit/Update tasks*", la fenêtre suivante s'affiche.



Figue I.17 : création d'une tâche donnée dans Cheddar

I.6.3.1 Résultats de simulation de l'exemple

Après avoir créé les trois tâches, on lance le test d'ordonnançabilité à travers le sous-menu "*tool/Scheduling/Customized schuduling simulation*" le simulateur affiche les résultats de simulation comme la montre la figure suivante :

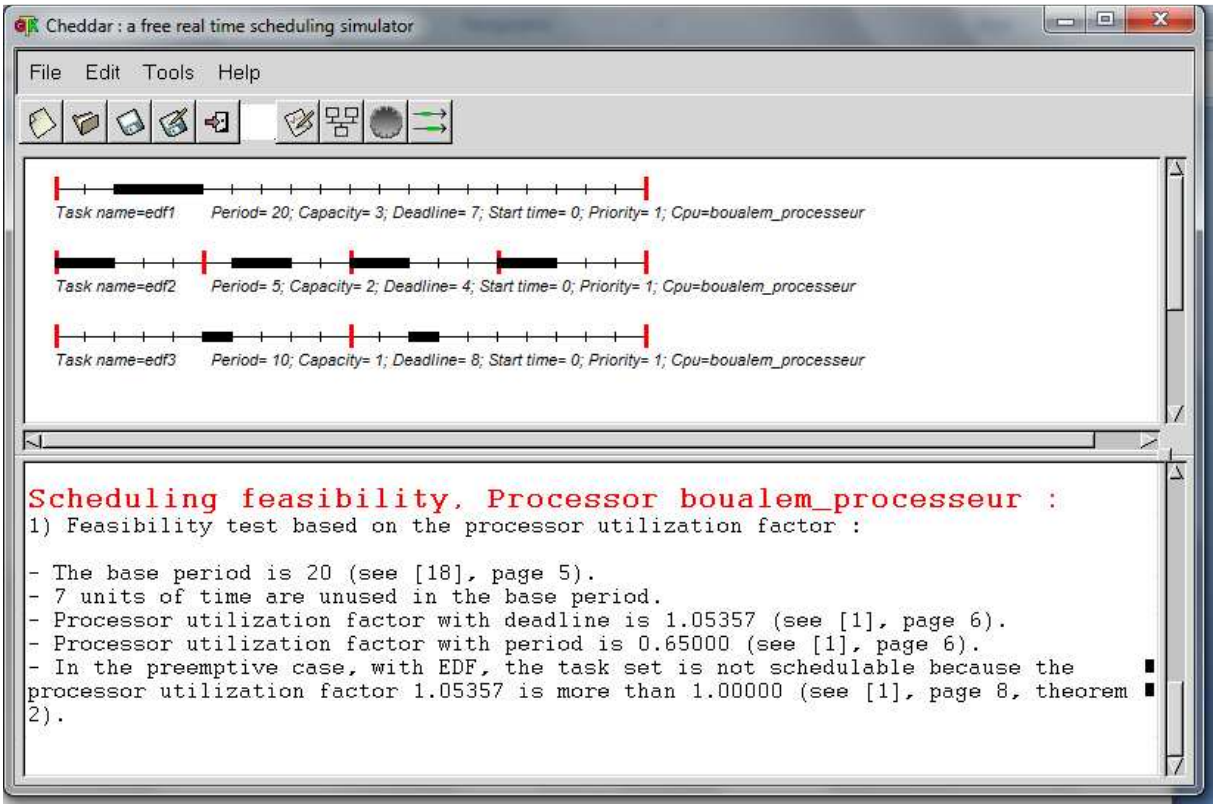


Figure I.18 : Résultats d'ordonnancement des trois tâches

Pour cet exemple d'EDF, la simulation montre un ordonnancement sans qu'aucune tâche ne dépasse son échéance. Le test de faisabilité satisfait la condition nécessaire et suffisante d'ordonnançabilité suivante :

$$\sum_{i=1}^{i=n} \frac{T(\tau_i)}{P(\tau_i)} \leq 1 \quad \rightarrow \quad \frac{3}{20} + \frac{2}{4} + \frac{1}{10} = 0.75 < 1$$

Conclusion

Dans ce chapitre, nous avons présenté les notions de tâches, leurs natures et caractéristiques, et les problèmes d'ordonnancement en temps réel, après une brève présentation des systèmes temps réel.

Ensuite on a présenté les déférentes classes et quelques algorithmes d'ordonnancement et leurs conditions d'ordonnançabilité et faisabilité.

En fin de ce chapitre, nous avons présenté un logiciel, appelé *Cheddar* qui permet de calculer et de tester les déférents critères d'ordonnancement, et on termine par un exemple de simulation avec ce logiciel.

Chapitre II

Introduction

La consommation en énergie est devenue un problème crucial dans la conception des équipements électroniques dont l'alimentation est assurée par des batteries.

Parmi tous les composants électroniques, le processeur est particulièrement utilisateurs d'énergie puisque des études ([08, 09] cités dans [10]) ont montré qu'il pouvait à lui seul utiliser plus de 50 % de l'énergie lorsqu'il est sollicité intensivement.

Dans ce chapitre nous allons présenter un aperçu sur la consommation d'énergie dans les processeurs (circuit CMOS) et ses sources de consommation. En suite, on va présenter quelques techniques de réduction de cette dernière en se basant sur une technique d'adaptation dynamique de la tension appelée DVS (Dynamique Voltage Scaling) qu'on utilisera dans la partie qui suit.

II.1 Définition d'un processeur

Le processeur, ou CPU (Central Processing Unit, « Unité centrale de traitement »), est le composant qui exécute les programmes informatiques. Avec la mémoire notamment, c'est l'un des composants qui existent depuis les premiers ordinateurs et qui sont présents dans tous les ordinateurs. Un processeur construit en un seul circuit intégré est un microprocesseur. L'invention du transistor en 1948 a ouvert la voie à la miniaturisation des composants électroniques.

Les processeurs des débuts étaient conçus spécifiquement pour un ordinateur d'un type donné. Cette méthode coûteuse de conception des processeurs pour une application spécifique a conduit au développement de la production de masse de processeurs qui conviennent pour un ou plusieurs usages. Cette tendance à la standardisation qui débuta dans le domaine des ordinateurs centraux (mainframes à transistors discrets et mini-ordinateurs) a connu une accélération rapide avec l'avènement des circuits intégrés. Les circuits intégrés ont permis la miniaturisation des processeurs. La miniaturisation et la standardisation des processeurs ont conduit à leur diffusion dans la vie moderne bien au-delà des usages des machines programmables dédiées.

II.2 Définition de la puissance instantanée

Quantité d'énergie électrique consommée par unité de temps, exprimée en Watts (W). Ce qui correspond à un Joule fourni par Seconde. Dans le cas où la tension U et l'intensité I sont continues, la puissance instantanée P_i s'écrit :

$$P_i = U \times I \qquad \text{(II.1)}$$

II.3 Définition de l'énergie consommée

Quantité d'énergie électrique consommée pendant une période de temps donnée, exprimée en Joules (J). Dans le cas où la puissance instantanée et le temps de fonctionnement s'écrivent respectivement P_i et t alors l'énergie consommée E_c s'écrit

$$E_c = P_i \times t \quad \text{(II.2)}$$

II.4 Consommation énergétique d'un processeur

Dans cette section, nous allons présenter quelques points-clés portant sur la consommation en énergie des processeurs. Nous examinons ensuite les différentes techniques permettant la réduction de la consommation énergétique.

II.4.1 Source de consommation énergétique d'un processeur

Les concepts expliqués dans cette section sont valables pour tous circuits CMOS (processeur) qui est la technologie dominante dans la fabrication des circuits électroniques. La puissance électrique consommée dans ces circuits est souvent divisée en deux composantes :

- une composante de consommation dynamique due à l'activité des transistors ;
- une composante de consommation statique due aux courants de fuites.

Ces deux composantes doivent être prises en compte lors de l'optimisation de la consommation et requièrent toutes les deux des méthodes de gestion différentes.

II.4.1.1 Consommation dynamique

La consommation dynamique, liée à l'activité du circuit, résulte de deux facteurs technologiques distincts : le courant de court-circuit I_{cc} et le courant de commutation. Ces deux courants apparaissent lors de la commutation des transistors.

$$P_{dynamique} = P_{court-circuit} + P_{charge} \quad \text{(II. 3)}$$

II.4.1.1.1 Courant de court circuit

Le courant de court-circuit apparaît lors de la commutation d'une porte logique comme le montre l'exemple de cet inverseur MOS (figure II.1).

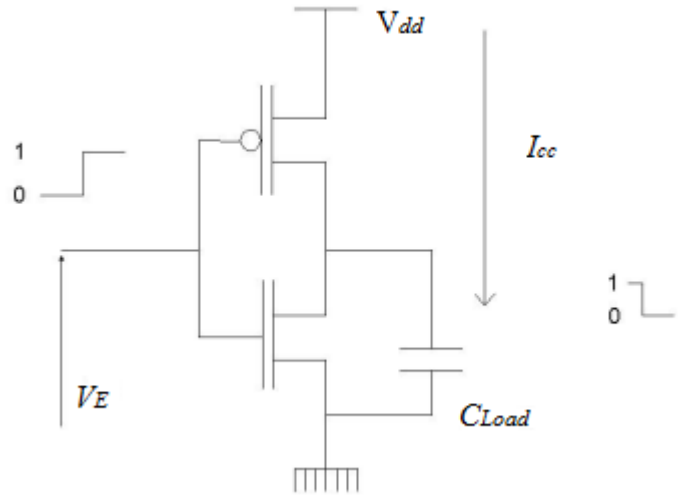


Figure II.1 : courant Court-circuit dans un inverseur CMOS

Lors d'une transition montante de V_E , les deux transistors vont commuter en même temps et créer pendant un court laps de temps un chemin direct entre l'alimentation et la masse, créant ainsi un courant de court-circuit.

La puissance dissipée aura donc pour expression (avec I_{cc} le courant de court-circuit proportionnel au temps de montée de l'ordre du µA) :

$$P_{cc} = V_{dd} \times I_{cc} \qquad \text{(II.4)}$$

II.4.1.1.2 Courant de commutation

Le courant de commutation est dû à la charge/décharge de la capacité équivalente présente en sortie des portes logiques. Lors des transitions, les transistors sont alternativement passants ou bloqués et peuvent donc être modélisés par des interrupteurs comme le montrent les figures suivantes (figure II.2).

Lors de la transition descendante (figure II.2.B), le transistor NMOS est bloqué et le transistor PMOS passant, la capacité en sortie de l'inverseur va donc se charger. La puissance fournie par l'alimentation va donc être égale à :

$$P(t) = V_{dd} \times C_l \times \frac{d\,V_{C_l}}{dt} \qquad \text{(II.5)}$$

Notation :

PMOS : un transistor à canal p

NMOS : un transistor à canal n .Pour plus de détails, le lecture peut se documenter dans [23]

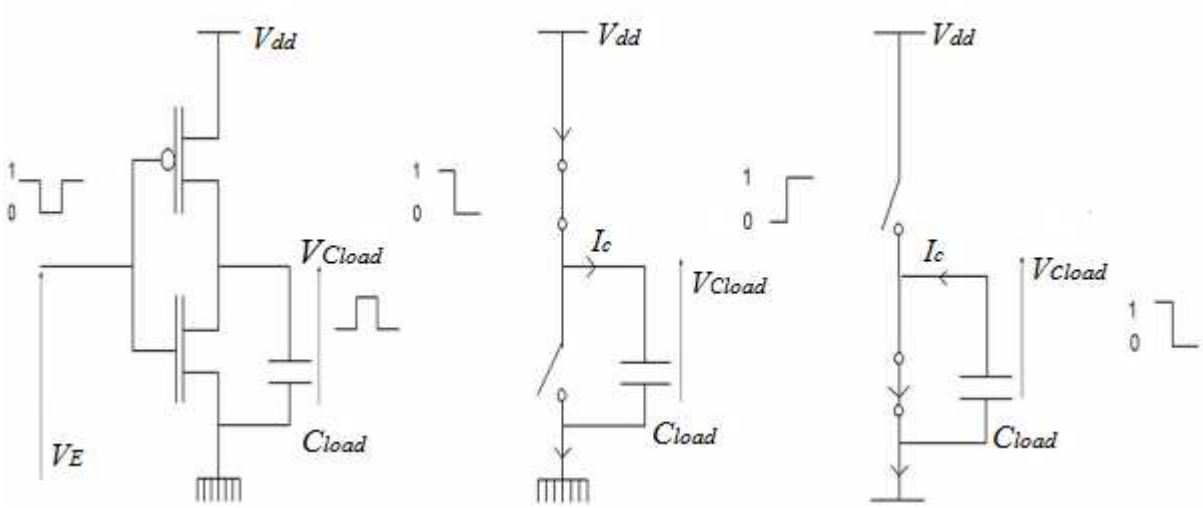


Figure II.2 : Courant de commutation dans un inverseur CMOS ; charge (B) et décharge de la capacité (C)

L'énergie fournie par l'alimentation lors de cette transition montante (E_m) est donc l'intégrale de la puissance sur le temps de montée :

$$E_m = \int_0^{Tm} P(t)dt = V_{dd} \int_0^{V_{dd}} C_l \, dV_C = C_l \times V_{dd}^2 \qquad \text{(II. 6)}$$

L'énergie stockée (et plus tard restituée) par la capacité de charge a pour expression :

$$E_m = \int_0^{Tm} P_{C_l}(t)dt = \int_0^{V_{dd}} C_l \, V_{C_l} * dV_C = \frac{1}{2} C_l \times V_{dd}^2 \qquad \text{(II. 7)}$$

Nous remarquons que l'énergie stockée dans la capacité ne représente que la moitié de l'énergie fournie par l'alimentation, le reste étant dissipé dans les transistors. De ces équations, nous pouvons déduire l'expression de la puissance de commutation dissipée dans les transistors composant les processeurs. Pour un seul transistor on obtiendra :

$$P_{co} = \alpha f C_l V_{dd}^2 \qquad \text{(II. 8)}$$

Tel que :

P_{co} : Puissance de commutation

α : représente le nombre moyen de commutations par cycle d'horloge

f : représente la fréquence d'horloge.

C_l : la capacité de charge.

V_{C_l} : tension au borne de capacité

V_{dd} : Tension d'alimentation

Donc, pour une architecture numérique formée de *n* transistors, la puissance de commutation aura pour expression :

$$P_{co} = \sum_{n}^{i} \alpha_i f C_{l_i} V_{dd}^2 \qquad \text{(II. 9)}$$

Avec α_i le nombre moyen de commutation de la cellule *i* et C_{l_i} sa capacité de charge.

Bilan :

La puissance dynamique étant la somme des deux contributions (court-circuit et commutation), nous obtenons comme expression :

$$P_{dyn} = V_{dd} I_{cc} + \sum_{n}^{1} \alpha f C_l V_{dd}^2 \qquad \text{(II. 10)}$$

La consommation dynamique est donc non seulement influencée par les paramètres technologiques (I_{cc} et C_l), mais aussi par l'utilisation que l'on fait du bloc (fréquence d'horloge, nombre moyen de commutations et tension d'alimentation).

II.4.1.2 Consommation statique

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

On distingue trois types de courant de fuites, comme la montre la (figure II.3) :

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

A température constante, la puissance statique d'un transistor dépend donc principalement de ces trois courants I_{seuil}, I_{diode}, I_{grille} et de la tension d'alimentation V_{dd} selon l'expression (II.10)

$$P_{statique} = I_{fuites} \times V_{dd} = (I_{seuil} + I_{diode} + I_{grille}) \times V_{dd} \qquad \text{(II.10)}$$

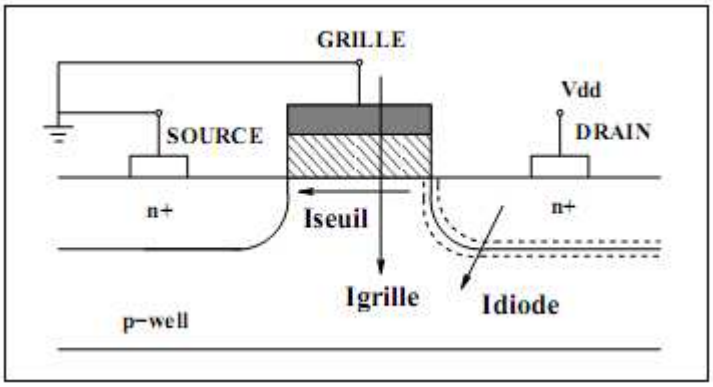


Figure II.3 : courants de fuites

Sachant que :

- Le courant sous le seuil, I_{seuil} est un courant qui circule entre le drain et la source du transistor quand il y une différence de potentiel drain-source et le transistor se trouve en état OFF.
- Le courant de fuites des jonctions,I_{diode} correspond aux fuites dans les diodes des jonctions PN (illustré à la figure II.3).

II.4.2 Méthodes de gestion de la consommation au niveau transistor

Pour permettre une réduction de la puissance dissipée par un transistor, il faut soit modifier ses paramètres physiques (substrat, dimensions...), soit ses paramètres de fonctionnement (tension d'alimentation, tension de polarisation du substrat...). Il est ainsi possible de réduire les deux composants de la puissance consommée : les puissances statique et dynamique.

II.4.2.1 Réduction de la puissance dynamique

L'équation (II.9) montre que la puissance de commutation (composante principale de la puissance dynamique) dépend de la capacité du circuit, de la fréquence de commutation et de la tension d'alimentation. La capacité équivalente C_l dépend entièrement des dimensions physiques du transistor (longueur, largeur etc…).

Pour diminuer la puissance dynamique dissipée, il faut agir sur la tension d'alimentation V_{dd}. Or, comme le montre l'équation (II.11) [11], le temps nécessaire à la commutation d'un transistor T dépend de la tension d'alimentation, de la tension de seuil (V_t) et de la capacité équivalente de charge (C_l).

Ainsi, toute diminution de V_{dd} entraîne une augmentation du temps de commutation. Un compromis devra donc être trouvé entre la tension d'alimentation et la fréquence de fonctionnement.

$$T = \frac{C_l \times V_{dd}}{(V_{dd} - V_t)^{\alpha}} \qquad \text{(II.11)}$$

II.4.2.2 Réduction de la puissance statique

Comme nous le montre l'équation (II.11), le temps de commutation d'un transistor dépend également de la tension de seuil V_t. Si V_{dd} s'approche de cette tension, le délai augmente considérablement. Pour permettre une grande plage admissible de tension et donc une diminution plus importante de la puissance dynamique, la tension de seuil V_t est réduite dans les nouvelles technologies.

Cette diminution entraîne une croissance exponentielle du courant de sous seuil et donc de la puissance statique dissipée. Pour pallier ces effets, les nouvelles technologies silicium permettent d'influer sur les tensions de seuil, et fixer ces dimensions.

II.4.3 Technique de réduction de la consommation énergétique

Cette partie présente un certain nombre de technique permettant de réduire la consommation énergétique. Certaines solutions purement matérielles, d'autre purement logicielle, et d'autre enfin, appelées hybrides, nécessitent une collaboration entre une partie matérielle et une partie logicielle.

Une de ces solutions hybrides est étudiée plus en détails: l'adaptation dynamique de la tension d'alimentation. Cette technique illustre l'impact que peut avoir une technique de minimisation de la consommation d'un processeur.

II.4.3.1 Techniques matérielles

Ces techniques contiennent deux parties essentielles ; la technologie des composants et l'architecture matériel.

II.4.3.1.1 Technologie des composants

La technologie des composants électroniques est un domaine où les progrès se succèdent à un rythme extrêmement rapide. L'augmentation des fréquences de fonctionnement des processeurs, ainsi que la réduction de la taille des circuits permettent une intégration toujours plus fine, concentrant une puissance de plus en plus importante dans un espace toujours plus étroit.

Cette technologie a été conçue dans le but de résoudre les problèmes de surchauffe des composants d'où leurs consommation énergétique.

Pour limiter la consommation des composants, il existe trois différentes techniques :

- *Tension d'alimentation* : dans la section (II.3.1) on a montré que la puissance dissipée dans un composant est :

$$P_{co} = \alpha f C_l V_{dd}^2$$

  On peut conclure que l'abaissement des tensions d'alimentation au niveau de l'effet transistor, et les courants de fuite qui apparaissent auront un impact de plus en plus important sur la consommation et le fonctionnement de circuit.

- *Basculement de bit* : cette technique est basée sur la diminution de taux de basculement des bits, il est dû au paramètre α qui masque plusieurs phénomènes complexe du fonctionnement d'un processeur, sachant que ces phénomènes ont un effet sur la consommation totale.

- *Activation séparées* : une autre méthode pour diminuer la consommation d'un composant est de limiter son alimentation aux blocs nécessaires au traitement en cours.

II.4.3.1.2 Architecteur du matériel

L'optimisation d'un système ne s'effectue pas seulement au niveau de la conception de ses composants, mais également au niveau du choix d'une architecture générale. Il existe deux paramètres au niveau du choix de matériel

- *Choix de périphérique* : le choix d'un périphérique peut devenir un problème critique lorsque celui-ci est destiné à un système dont la gestion de la consommation représente un critère essentiel. Mais au lieu de rechercher le périphérique le plus performant de sa catégorie, il peut parfois être préférable de changer complètement de technologie.

- *Dimensionnement* : Le dimensionnement des périphériques a également une influence sur la consommation. Un des exemples les plus flagrants est le dimensionnement des caches. Plus les caches ont une taille importante plus ils consomment

II.4.3.2 Techniques logicielles

Elles consistent à modifier le code à exécuter d'un programme pour diminuer la consommation induite par l'exécution de ce dernier. Elles interviennent au niveau des outils de développement des applications. Bien que leurs concepts restent relativement simples, il est difficile d'automatiser leur utilisation.

En effet elles requièrent une connaissance très précise du code applicatif ainsi que des caractéristiques de la technologie matérielle. C'est pourquoi nous étudierons les moyens existant pour quantifier la consommation de l'exécution d'un programme avant de décrire quelques optimisations possibles.

II.4.3.2.1 Evaluation de la consommation logicielle

Il est difficile d'évaluer le coût énergétique d'une instruction sur les processeurs actuels. L'utilisation systématique de pipeline ne permet pas d'isoler la consommation d'une instruction parmi toutes celles présentent dans le bus de donnée. Il est cependant possible d'obtenir une valeur moyenne du coût énergétique d'une instruction en exécutant celles-ci un grand nombre de fois.

Cependant, cette méthode a l'inconvénient de ne pas pouvoir prendre en compte les interactions qui peuvent exister entre l'exécution de deux instructions différentes.

II.4.3.2.2 Optimisation du code machine

Bien que la consommation due à l'exécution d'un programme soit difficile à estimer, il existe une relation entre le nombre d'instructions exécutées et la consommation.

D'une manière générale, plus le nombre d'instructions est élevé, plus la consommation est importante. Les techniques d'optimisation logicielles peuvent donc aider à diminuer cette consommation.

La majorité des développements se fond actuellement avec des langages dits de ≪ haut-niveau≫ comme le langage C, java ou Ada [12].

II.4.3.3 Techniques hybrides

Les techniques dites hybrides sont basées sur une collaboration entre composants matériels et logiciels, comme les techniques de mise en veille plus ou moins profondes des composants, ou bien la technique d'adaptation de la vitesse du processeur.

II.4.3.3.1 mise en veille

La mise en veille consiste à désactiver certains périphériques ou certaines parties de périphériques lorsque le système n'en a pas l'usage pendant un certain temps. Les périphériques se contentent généralement d'implanter des mécanismes pour supporter plusieurs états de fonctionnement.

- *Interface avec le système d'exploitation* : La réduction de la consommation au niveau du système d'exploitation est basée sur une collaboration entre, les applications et le système d'exploitation constituant le système. Les applications possèdent des comportements et des contraintes qui influencent l'activité et donc la consommation globale du système. Le système d'exploitation peut exploiter les propriétés des applications pour gérer la consommation d'énergie par des mises en veille ou par l'adaptation de la tension d'alimentation de toute partie du système.

- *Politiques d'endormissement*: Pour réduire la consommation dans le système d'exploitation on cherche à mettre en veille tous les périphériques qui ne contribuent pas au cours de fonctionnement du système.

II.4.3.3.2 Adaptation dynamique de la vitesse du processeur

L'adaptation de la vitesse du processeur peut être vue comme un contrôle plus proche des techniques de mises en veilles décrites précédemment. Les processeurs actuels ont des capacités de traitement de plus en plus importantes, cet accroissement de leur puissance de calcul à provoquer une augmentation de leur consommation. Notons que le fonctionnement d'un processeur n'est pas constant, car il est constitué de périodes de traitement et de périodes d'inactivité c.à.d. là où le processeur n'effectue pas de traitement utile pour l'utilisateur, mais continue à consommer de l'énergie.

Pour régler ce problème, il existe deux méthodes : l'une agit sur le signal d'horloge, et l'autre sur la tension d'alimentation du processeur.

- *Inhibition d'horloge* **:** C'est une méthode qui vise à éviter le traitement des signaux de l'horloge pendant les périodes d'inactivité du processeur. La consommation due à la propagation du signal d'horloge représente environ 40% de la consommation total du processeur. Lorsque la capacité de traitement du processeur est supérieure à la quantité de traitement à effectuer, il est possible de diminuer la fréquence moyenne de l'horloge afin de diminuer la consommation.

- *Adaptation du voltage (tension d'alimentation)* : cette technique permet des réductions de consommation importantes en négligeant la puissance statique ; l'énergie consommée varie au minimum en le carré du voltage dans les technologies CMOS actuelles équation (II.9). De manière générale l'adaptation dynamique de la tension d'alimentation, permet au processeur de modifier à la demande (dynamiquement) sa tension et sa fréquence, donc sa vitesse de traitement et sa consommation énergétique. Ces techniques sont appelées «dynamique voltage Scaling DVS » qu'on utilisera dans le chapitre 3 pour minimiser la consommation d'énergie d'un système a faible puissance.

II.5 Adaptation dynamique de la tension (DVS)

La technique d'ajustement conjoint, en tension et en fréquence est particulièrement distinguée par sa grande efficacité à réduire la consommation des processeurs. Cette technique permet de baisser à la fois la tension d'alimentation et la fréquence, ce qui ralentit l'exécution de la tâche et réduit sensiblement la consommation d'énergie du processeur.

Notons que diminuer la tension d'alimentation V_{dd} impose une fréquence f de fonctionnement plus faible. Comme la puissance dynamique consommée est proportionnelle à V_{dd}^2 et à f, des gains significatifs en énergie consommée sont très vite obtenus comme le montre le tableau (III.3) du chapitre 3.

Les algorithmes qui utilisent cette technique de variation de tension, ont besoin des données qui suivent pour déterminer l'intervalle de temps d'exécution de chaque tâche :

- Une date de lancement au plus tôt donnant l'instant où l'exécution de la tâche peut commencer,
- Une date d'échéance indiquant à quel moment l'exécution de la tâche doit être terminée.

II.5.1 Nature des techniques DVS

Les techniques DVS ont des hypothèses très importantes à considérer avant de réaliser un programme pour diminuer la consommation énergétique, et elles sont comme suit :

- Ces techniques sont appliquées à des systèmes temps réel à contraintes souples ou strictes :

Les techniques de DVS temps réel souple sont généralement basées sur la prédiction des temps d'exécution à partir des exécutions précédentes pour adapter la fréquence et la tension de fonctionnement du processeur. L'efficacité de ces techniques en termes de réduction de la consommation est liée au modèle de prédiction utilisé. Le non respect de quelques échéances est dans ce cas sans conséquences graves sur le fonctionnement du système.

Pour les techniques de DVS à temps réel strict, le respect de toutes les échéances des tâches est obligatoire pour assurer le bon fonctionnement du système. Des techniques de ce type sont présentées ci-dessous.

- Intra-DVS ou Inter-DVS : les techniques Intra-DVS utilisent le deadline (date limite) mais non consommé par la tâche courante pour réduire la vitesse d'exécution de cette tâche, alors que la technique Inter-DVS utilise le deadline de la tâche courante pour réduire la vitesse d'exécution des tâches suivantes.

- en ligne ou hors ligne: les prises de décisions d'ajustement de la tension et de la fréquence de fonctionnement du processeur sont faite soit hors ligne, lors de la phase de conception c.à.d. avant l'exécution, soit en ligne c'est-à-dire en cours d'exécution.

- dynamique ou statique : les techniques statiques de DVS déterminent une vitesse fixe de fonctionnement qui reste inchangée tout au long de l'exécution, alors que les techniques dynamiques adaptent la vitesse du processeur en fonction de plusieurs paramètres comme les temps d'exécution des tâches, le taux d'utilisation du processeur.

Conclusion :

Dans ce chapitre, nous avons présenté l'origine et les différentes sources de consommation énergétique liées à la technologie de fabrication des processeurs. Plusieurs techniques de réduction de cette consommation sont abordées pour résoudre ce problème de consommation ont été décrites.

Cependant, on a présenté trois types de techniques : matérielles, logiciels et hybrides. Leurs utilisations permettent d'obtenir une diminution importante de la consommation énergétique dûe aux traitements effectués.

Pour notre cas, on s'intéresse aux techniques hybrides, basées sur une collaboration entre un module logiciel et des fonctionnalités matérielles qui permettent d'ajuster au mieux la consommation d'un système à son niveau d'activités. Plus particulièrement la technique d'ajustement de la tension d'alimentation (DVS) qu'on utilisera dans le chapitre trois.

Chapitre III

Introduction

Minimiser la consommation d'énergie dans les systèmes de faible puissance est devenue un critère primordial, en particulier en vue des systèmes embarqués en temps réel. L'augmentation de la durée de vie des portables et mobiles dépend fortement de la gestion de la batterie. Dans des systèmes de faible puissance, le processeur consommerait de 18 à 30% de l'ensemble de la consommation d'énergie et dépasse souvent les 50%. Le contrôle de la fréquence et de la tension fournit les moyens pour réguler la consommation d'énergie du processeur conduisant à la technique DVS.

Notre objectif dans cette partie est de minimiser la consommation énergétique d'un processeur pour un système à faible puissance. Les tâches à effectuer sont non préemptif et apériodiques.

L'objectif est de contrôler le temps de traitement des tâches et les ordonnancer d'une manière à satisfaire toutes les contraintes temporelles, pour avoir les résultats de la consommation d'énergie en variant les différentes valeurs de la tension d'alimentation. Ce problème d'optimisation est résolu sous Matlab.

III.1 Position du problème

Notre approche est basée sur un problème d'optimisation qui consiste en la minimisation d'une fonction objectif qui est la consommation d'énergie en fonction de temps de traitement des tâches (fréquence de fonctionnement).

La structure du problème conduit à des propriétés intéressantes, tel que : la convexité de la fonction objectif, que nous exploitons pour formuler la fonction à optimiser (fonction objectif) qui est soumise à des contraintes temporelles basées sur les caractéristiques des tâches (date d'activation, l'échéance…etc.).

Pour résoudre ce problème d'optimisation, on a proposé un programme sous Matlab qui nous permet d'avoir l'évolution de la fréquence de fonctionnement et la consommation d'énergie en diminuant la tension d'alimentation.

III.2 Nature des tâches

Dans ce problème, nous considérons les tâches de traitement du système non préemptives et apériodiques. Nous supposons également que l'ordre dans lequel les tâches doivent êtres exécutées est donné à priori selon une politique d'ordonnancement. Le problème traité est hors-ligne, c'est à dire que les dates d'activation des tâches et leurs délais (deadline) sont connues.

Notre objectif est d'affecter des délais de traitement (ce qui revient a la vitesse de processeur, par le biais de contrôle de tension) des tâches de façon à minimiser une fonction de consommation d'énergie totale, tout en garantissant qu'aucune exécution de tâche donnée ne dépasse les délais.

III.3 Les outils utilisés

III.3.1 Problème d'optimisation

Etant donnée une fonction $f : S \rightarrow \mathbb{R}$,
Un problème d'optimisation consiste, à trouver:

1) son minimum *v* (resp. son maximum) dans *S*

2) un point $x_0 \in$ S qui réalise ce minimum (resp. maximum) i.e.

$f(x_0) = v$.

Vocabulaire

– *f est la fonction objectif*

– *v est la valeur optimale*

– x_0 *est la solution optimale*

– *S = {solutions réalisables du problème}*

– *écriture du problème :* $min_{x \in S} f(x)$ *resp.* $max_{x \in S} f(x)$

III.3.2 La convexité

Pourquoi la convexité ?

En pratique il n'est pas crucial de déterminer l'expression exacte de la puissance car la puissance dynamique dissipée reste toujours une fonction convexe croissante de la fréquence. Beaucoup de résultats énoncés en économie d'énergie (minimisation d'énergie) sont valables pour toute fonction convexe [13].

La forte convexité est le cadre agréable pour de nombreux problèmes d'optimisation, car elle donne facilement l'existence, l'unicité, et les algorithmes de calcul performent.

Définition **:** Un ensemble $K \subset R^n$ est dit convexe si pour tout couple (x, y) $\in K^2$ et $\forall \lambda \in [0,1]$ on a :

$$\lambda x + (1-\lambda)y \in K \qquad \text{(III.1)}$$

Cette définition peut s'interpréter en disant que le segment reliant *x* et *y* doit être dans *K*. Elle se généralise de la façon suivante : on dira qu'un vecteur *y* est une combinaison convexe des points $\{x^1 \dots \dots x^p\}$ si on a :

$$y = \sum_{i=1}^{p} \lambda_i x_i \qquad \text{(III. 2)}$$

$$\text{avec :} \quad \lambda_i \geq 0 \quad et \quad \sum_{i=1}^{p} \lambda_i = 1$$

III.3.3 Fonction convexe

Définition **:** On dit qu'une fonction $f: K \to \mathbb{R}$, définie sur un ensemble convexe *K*, est convexe si elle vérifie

$$\forall (x,y) \in K^2, \forall \lambda \in [0,1], \quad f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda) f(y). \qquad \text{(III.3)}$$

On dira que *f* est strictement convexe si

$$\forall (x,y) \in K^2, x \neq y, \forall \lambda \in [0,1], \quad f(\lambda x + (1-\lambda)y) < \lambda f(x) + (1-\lambda) f(y). \qquad \text{(III.4)}$$

Lorsque $n = 1$ cette définition s'interprète bien géométriquement : le graphe de la fonction est toujours en dessous du segment reliant les points $(x, f(x))$ et $(y, f(y))$.

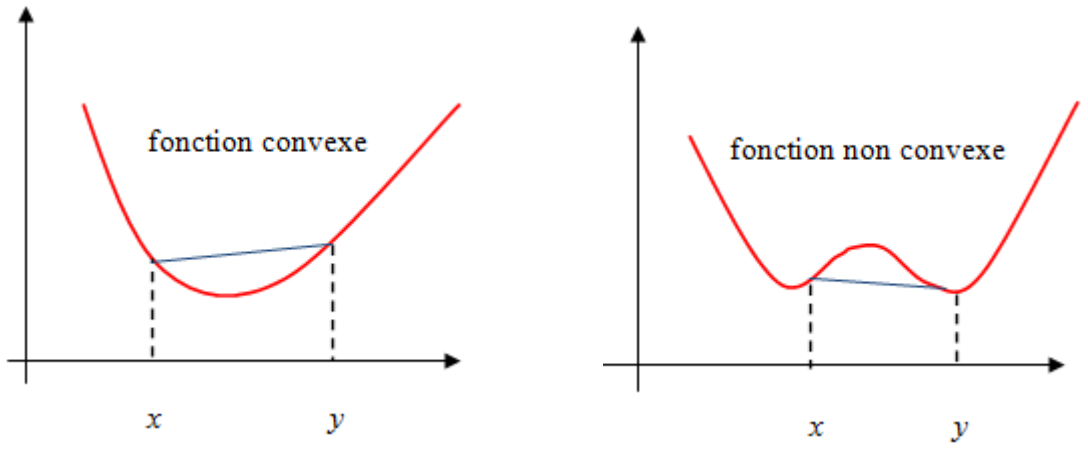


Figure III.1 : Exemple d'une fonction convexe

III.4 Formulation du problème

Tel que mentionné dans le chapitre deux, la consommation d'énergie totale E_{Total} dans les processeurs CMOS comprend essentiellement deux parties : l'énergie dynamique consommée E_{dyn} causée par la tension d'alimentation V_{dd} et la consommation statique d'énergie $E_{statique}$ causée par les courants de fuite.

Tel que :

$$E_{Total} = E_{dyn} + E_{statique}, \qquad E_{dyn} = C_1 \times V_{dd}^2 \tag{III.5}$$

La fréquence de traitement (la vitesse du processeur lors du traitement d'une tâche) est donnée par :

$$f = \frac{(V_{dd} - V_t)^{\alpha}}{C_2 \times V_{dd}} \tag{III.6}$$

Où C_1 , C_2 et $\alpha \in [1,2]$ sont des constantes dépendantes de la caractéristique de dispositif, $V_t$est la tension de seuil, de tel sorte que $V_{dd} \geq V_t$.

Remarque :

Dans les étapes qui suivent, on travail avec $\alpha = 1$.

c.-à-d : $$f = \frac{(V_{dd} - V_t)}{C_2 \times V_{dd}} \tag{III.7}$$

III.4.1 Fonction de consommation énergétique (fonction objectif)

En ce que concerne la fonction de consommation d'énergie $\theta(T_i)$, il s'agit d'une fonction strictement convexe de T_i (temps de traitement) et monotone décroissante.

Cependant, nous pouvons obtenir plus d'information sur $\theta(T_i)$ en utilisant les relations (III.5) et (III.6) à partir de laquel nous pouvons écrire :

$$\theta(T_i) = E_{total} = C_1(h^{-1}(T_i))^2 + E_{statique} \tag{III.8}$$

Où $E_{dyn} = C_1(h^{-1}(T_i))^2$; $h^{-1}(T_i)$ sa fonction inverse.

Pour $\alpha = 1$, on détermine l'expression de V_{dd} :

$$h(V_{dd}) = T_i = \frac{1}{f_i} = \frac{C_2 V_{dd}}{(V_{dd} - V_t)} \qquad \Rightarrow \qquad V_{dd} = \frac{T_i V_t}{T_i - C_2} \qquad \text{(III. 9)}$$

Vérification de la convexité de h^{-1}:

Pour tout: T_1 ,$T_2 \in \varphi$ et $\beta \in [0,1]$

alors: $\beta T_1 + (1-\beta)T_2 \in \varphi$

h(V_{dd}) est la fonction de temps de traitement en foncton de la tension d'alimentation :

$$\beta T_1 + (1-\beta)T_2 = \beta h\left(h^{-1}(T_1)\right) + (1-\beta)h\left(h^{-1}(T_2)\right)$$
$$\geq h[\beta h^{-1}(T_1) + (1-\beta)h^{-1}(T_2)] \qquad \text{(III.10)}$$

Puisque *h(V_{dd})* est monotone, alors h^{-1} (T_i) est également monotone, ce qui implique, pour tout $\beta \in [0,1]$:

$$h^{-1}(\beta T_1 + (1-\beta)T_2) \leq h^{-1}(h(\beta h^{-1}(T_1) + (1-\beta)h^{-1}(T_2))) \qquad \text{(III.11)}$$
$$= \beta h^{-1}(T_1) + (1-\beta)h^{-1}(T_2),$$

Donc :

$$T_i = h(V_{dd}) = \frac{C_2 V_{dd}}{(V_{dd} - V_t)}$$

$$\Rightarrow \qquad h^{-1}(T_i) = V_{dd}$$

Définition d'une fonction monotone :

C'est une fonction dont le sens de variation ne change pas. Une fonction monotone sur un intervalle est une fonction qui reste croissante ou qui reste décroissante sur cet intervalle.

Alors la fonction de la consommation énergétique devient :

$$\theta(T_i) = C_1.\left(\frac{T_i.V_t}{T_i - C_2}\right)^2 + E_{statique} \quad \text{(III. 12)}$$

avec : $T_i \neq C_2$

En raison de la présence de $E_{statique}$, la consommation $\theta(T_i)$ est généralement pas convexe ou monotone décroissant puisque $E_{statique}$ augmente en diminuant V_{dd}. Cependant, il existe une limite de fonctionnement inférieure V_{min} sur V_{dd} tels que $E_{statique}$ peut être considéré comme une petite constante dans la plage de fonctionnement.

Puisque $h^{-1}(T_i)$ est strictement convexe par rapport à T_i, $\theta(T_i)$ est convexe et une fonction monotone décroissante en T_i avec la limite supérieure correspondante exprimée comme suit :

$$T_{max} = \frac{C_2 V_{min}}{(V_{min} - V_t)} \quad \text{(III. 13)}$$

Même la tension d'alimentation V_{dd} est physiquement limitée par une borne supérieureV_{max}, c'est-à-dire il y a une borne inférieure correspondante pour T_i , exprimé comme suit :

$$T_{min} = \frac{C_2 V_{max}}{(V_{max} - V_t)} \quad \text{(III. 14)}$$

Pour résumer nous avons : $T_{min} \leq T_i \leq T_{max}$

III.4.2 Les contraintes

Pour atteindre l'objectif, il y a deux questions que nous devons poser :

- Les tâches de processeur sont à quel rythme ?
- La façon de commander ces tâches ?

Puisque le problème est non préemptif apériodique, on peut fixer l'ordre de traitement des tâches et le modèle de leur dynamique de files d'attente grâce à l'équation de Lindley [14] comme suit:

$$x_i = \max\ (x_{i-1}, a_i) + \mu_i T_i \quad \text{(III. 15)}$$

Qui peut être développé comme suit :

$$\begin{cases} x_i \leq x_{i-1} + \mu_i T_i \\ x_i \leq a_i + \mu_i T_i \end{cases} \quad \text{(III. 16)}$$

Sachant que $x_i \leq d_i$; c'est-à-dire la date de début d'une tâche est inférieur ou égale à la date limite (deadline) de cette tâche.

Tel que:

x_i: Date de départ de la tâche *i*

a_i: Date d'arriver de la tâche *i*

μ_i : Nombre de coup de l'horloge qu'il faut pour réaliser une tâche.

T_i : Temps de traitement d'une tâche (la variable à contrôler)

III.4.3 La forme du critère à optimiser

Après avoir déterminer la fonction objectif, les variables de décision et les contraintes, le critère peut s'écrire sous la forme suivante :

$$\min_{T_1,\ldots,T_N} \left\{ J = \sum_{i=1}^{N} \mu_i \, \theta(T_i) \right\} \quad \text{(III. 17)}$$

Sujet à:

$$x_i = \max(x_{i-1}, a_i) + \mu_i \tau_i \qquad i = 1\ldots,N$$

$$x_i \leq d_i \qquad i = 1\ldots,N$$

$$T_{min} \leq T_i \leq T_{max}$$

Avec :

N : nombre total des tâches

d_i : Date limite de la tâche *i* (deadline)

T_i : Temps de traitement de la tâche *i*

T_{max} : Temps de traitement maximal

T_{min} : Temps de traitement minimal

$\theta(T_i)$: Fonction de consommation énergétique.

III.4.4 Développement du problème

Dans ce problème $E_{statique}$ est considérée comme une petite constante par rapport à l'énergie dynamique E_{dyn} (négligeable devant E_{dyn}) c'est-à-dire l'énergie totale prend la forme suivante :

$$E_{total} = \theta(T_i) = C_1.(\frac{T_i.V_t}{T_i - C_2})^2 \quad \text{(III. 18)}$$

On travail avec 5 tâches ; N=5 (nombre de tâches à réaliser),

On considère $\mu_1 = \mu_2 = \mu_3 = \mu_4 = \mu_5 = 1$. C'est-à-dire un coup d'horloge pour l'ensemble de tâches

Suivant la nature de la fonction objectif $\theta(T_i)$ et les contraintes, le problème d'optimisation correspond à un problème non linéaire, suite a la présence de l'operateur non linéaire '*max*'. Le problème est développé comme suit :

$$\min_{T_1,\dots,T_N} \left\{ \theta(T_i) = C_1 . \left(\left(\frac{T_1 . V_t}{T_1 - C_2}\right)^2 + \left(\frac{T_2 . V_t}{T_2 - C_2}\right)^2 + \left(\frac{T_3 . V_t}{T_3 - C_2}\right)^2 + \left(\frac{T_4 . V_t}{T_4 - C_2}\right)^2 + \left(\frac{T_5 . V_t}{T_5 - C_2}\right)^2 \right) \right\}$$

Sujet à :

$$\begin{gathered}
x_1 - T_1 \leq a_1 \\
-x_1 + x_2 - T_2 \leq 0 \\
x_2 - T_2 \leq a_2 \\
-x_2 + x_3 - T_3 \leq 0 \\
x_3 - T_3 \leq a_3 \\
-x_3 + x_4 - T_4 \leq 0 \\
x_4 - T_4 \leq a_4 \\
-x_4 + x_5 - T_5 \leq 0 \\
x_5 - T_5 \leq a_5 \\
x_1 \leq d_1 \\
x_2 \leq d_2 \\
x_3 \leq d_3 \\
x_4 \leq d_4 \\
x_5 \leq d_5 \\
T_1 \leq T_{max} \\
-T_1 \leq -T_{min} \\
T_2 \leq T_{max} \\
-T_2 \leq -T_{min} \\
T_3 \leq T_{max} \\
-T_3 \leq -T_{min} \\
T_4 \leq T_{max} \\
-T_4 \leq -T_{min}
\end{gathered} \qquad \text{(III. 19)}$$

$$-T_5 \leq -T_{min}$$

Le problème est sous la forme suivante :

$$\min_x f(x)$$

S à :

$$Ax \leq b$$

Tel que :

$$X = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ T_1 \\ T_2 \\ T_3 \\ T_4 \\ T_5 \end{pmatrix}$$

Les dates de départ des 5 tâches. Pour vérifier le respect des contraintes

Les temps de traitement des 5 tâches. Ce sont les variables contrôlables (variable de décision)

$$b = \begin{pmatrix} a_1 \\ 0 \\ a_2 \\ 0 \\ a_3 \\ 0 \\ a_4 \\ 0 \\ a_5 \\ d_1 \\ d_2 \\ d_3 \\ d_4 \\ d_5 \\ T_{max} \\ -T_{min} \\ T_{max} \\ -T_{min} \\ T_{max} \\ -T_{min} \\ T_{max} \\ -T_{min} \\ T_{max} \\ -T_{min} \end{pmatrix} ; \quad A = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \end{pmatrix}$$

III.5 Programmation sous Matlab

Puisque le système étudié est hors ligne et strict, les tâches sont non préemptifs et apériodiques (on connaît leurs dates d'activation et leurs deadline), on pose a_i et d_i d'une manière à satisfaire toutes les contraintes. Pour répondre au problème de minimisation d'énergie consommée, on diminue la tension d'alimentation qui agit sur la fréquence de fonctionnement (technique DVS).

Les dates d'arrivée et les échéances (deadlines) des 5 tâches sont données dans le tableau suivant :

Tableau III.1 : Dates d'arrivées et de l'échéance des 5 tâches

	a_i	d_i
Tâche 1	4	9
Tâche 2	5	10
Tâche 3	6	15
Tâche 4	7	20
Tâche 5	8	25

Pour un bon fonctionnement, on a choisi les valeurs suivantes pour les différents paramètres : $C_1 = 1, C_2 = 0.1,$ et $V_t = 0.5$

Ces paramètres sont motivés par le circuit CMOS. Le programme qui permet de répondre à ce problème d'optimisation est programmé sous Matlab.

III.5.1 Résultats de simulation

Après avoir simulé le programme sur Matlab, nous avons obtenu les résultats suivants:

Tableau III.2 : Dates de départs et les temps de traitements des 5 tâches après simulation

	Tâche 1		**Tâche 2**		**Tâche 3**		**Tâche 4**		**Tâche 5**	
$Tension$ **(volts)**	x_1	T_1	x_2	T_2	x_3	T_3	x_4	T_4	x_5	T_5
$V_{dd} = 5$	3.200	**0. 200**	3.400	**0.200**	3.600	**0.200**	3.800	**0.200**	4.000	**0.200**
$V_{dd} = 4$	3.225	**0.225**	3.450	**0.225**	3.675	**0.225**	3.900	**0.225**	4.125	**0.225**
$V_{dd} = 3$	3.2667	**0.2667**	3.5333	**0.2667**	3.800	**0.2667**	4.0667	**0.2667**	4.3333	**0.2667**
$V_{dd} = 2$	3.350	**0.3500**	3.700	**0.3500**	4.050	**0.3500**	4.050	**0.3500**	4.750	**0.3500**
$V_{dd} = 1$	3.600	**0.600**	4.200	**0.600**	4.800	**0.600**	5.400	**0.600**	6.000	**0.600**

L'évolution de l'énergie consommée et le temps de traitement en fonction de la variation de la tension d'alimentation pour une tâche (tâche5) est représentée dans le tableau suivant :

Tableau III.3: L'évolution de $\theta(T_i)$ et T_i en diminuant V_{dd}

Tension d'alimentation (v)	**Temps de traitement (s)**	**Energie consommée (joule)**
5.0000	0.2000	2.4149
4.0000	0.2250	2.3805
3.0000	0.2667	2.3282
2.0000	0.3500	2.2381
1.0000	0.6000	2.0454

Les résultats de simulation graphique sont présentés dans les figures (III.1) et (III.2)

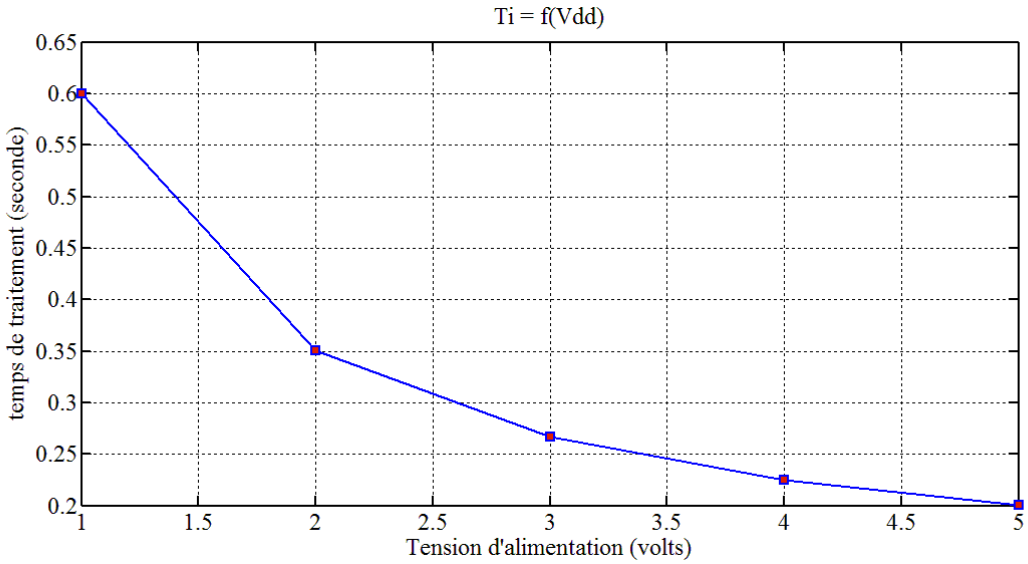


Figure III.1 : temps de traitement en variant la tension d'alimentation

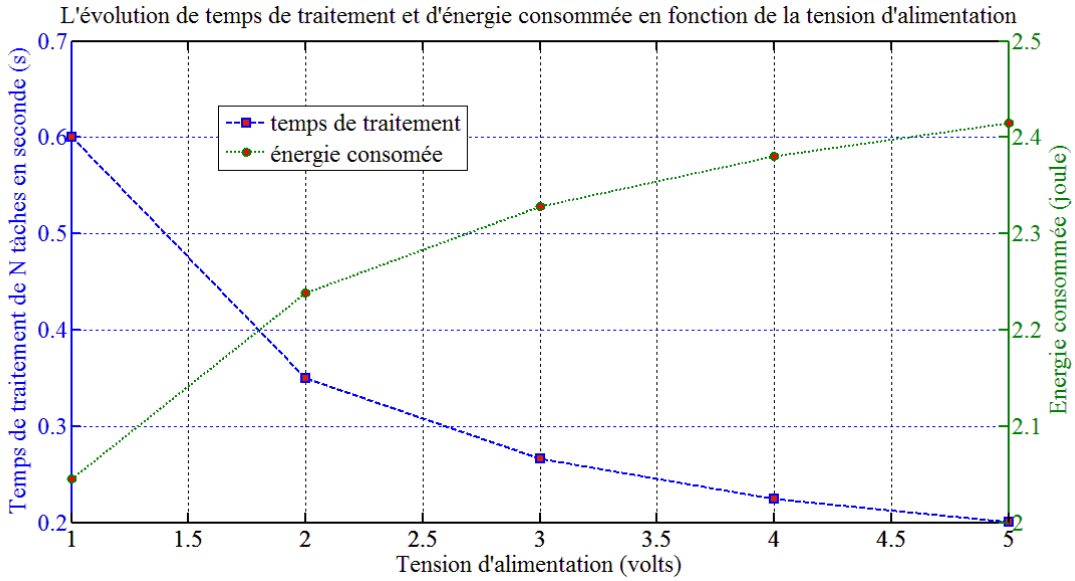


Figure III.2 : Effets de la tension d'alimentation

Interprétation des résultats :

Pour interpréter les résultats de simulation, on doit d'abord vérifier le respect des contraintes posées. Après avoir choisi des valeurs pour les différents paramètres du modèle. Par exemple avec les tâches 5 et 4 pour V=5v, on a :

$x_5 = 13,\ x_4 = 11,\ T_5 = 2,\ a_5 = 11,\ d_4 = 20$ et $d_5 = 25$

On vérifie les contraintes suivantes :

$-x_4 + x_5 - T_5 \leq 0 \quad \rightarrow \quad -11 + 13 - 2 = 0$ (vérifie)

$x_5 - T_5 \leq a_5 \quad \rightarrow \quad 13 - 2 = 11$ (vérifie)

$x_4 \leq d_4 \quad \rightarrow \quad 11 \leq 20$ (vérifie)

$x_5 \leq d_5 \quad \rightarrow \quad 13 \leq 25$ (vérifie)

Après vérification de toutes les contraintes et qu'aucune tâche ne dépasse sont délais, et en observant la relation liant le temps de traitement et la consommation à la tension d'alimentation, on conclu que diminuer cette dernière a un impact important sur la consommation énergétique.

On conclu les résultats suivants :

- On diminuant la tension, le temps de traitement augmente c'est-à-dire la fréquence de fonctionnement diminue ($T = \frac{1}{f}$) et la consommation d'énergie diminue (Tableau III.3).

- Il faut faire attention à ne pas trop se rapprocher de la tension de seuil pour éviter d'augmenter de manière exponentielle le temps de traitement. Les résultats montrent qu'il existe une tension d'alimentation optimale se situant autour de 2 Volts (Figure III.2).

- Le problème d'ordonnancement qu'on a résolu consiste non seulement à déterminer l'ordre dans lequel exécuter les tâches du système mais également à fixer la fréquence de fonctionnement du processeur au cours du temps (même temps de traitement pour chaque tâche).

- Si aucune contrainte de temps n'est spécifiée alors la meilleure stratégie vis-à-vis de la consommation est de mettre le processeur en veille ce qui naturellement est incompatible avec le niveau de performances minimales attendu.

Conclusion

La méthode DVS étudiée dans cette partie vise à diminuer l'énergie consommée d'un processeur en faisant diminuer sa tension d'alimentation, et par conséquent sa fréquence de fonctionnement.

Le modèle construit nous a permis d'obtenir des profils de consommation, en fonction des paramètres de notre système, (tension/fréquence).

Afin d'évaluer la consommation des systèmes temps réel, et de déterminer la méthode optimale pour la minimiser, nous devons disposer d'un outil de modélisation permettant d'évaluer la consommation du circuit dans le cadre de la minimisation de la consommation énergétique des processeurs,

Conclusion générale

Le travail qu'on a traité dans ce mémoire est divisé en deux partie essentielles : Dans la première partie, les notions de tâches temps réel, ordonnancement, ordonnançabilité ont été abordés. Des algorithmes permettent de donner des solutions d'ordonnancement temps réel ont été présentés. Ces algorithmes permettent de prendre en charge des tâches : périodique/apériodique, préemptif /non préemptif et peuvent être implémenté en hors-ligne où en ligne. Un logiciel qui permet de prendre en charge des problèmes d'ordonnancement temps réel sur processeur donnée est présenté vers la fin du premier chapitre.

Dans la deuxième partie de ce travail, le problème de minimisation de la consommation énergétique d'un processeur donnée dans un système à faible puissance temps réel est traité. Nous avons montré qu'il existe plusieurs manières de réduire la consommation énergétique pour processeur. Parmi ces méthodes, la technique basée sur l'adaptation dynamique de la tension d'alimentation nous a paru la plus intéressante. Ceci est dû au fait, qu'elle utilise à la fois le logiciel et le matériel pour réduire la facture énergétique du système étudie. Nous avons donc, utilisé cette technique (DVS) pour ordonnancer en hors-ligne un problème temps réel strict à 5 tâches élémentaires (nécessitant un temps d'horloge pour sa réalisation).

En conclusion, on constate que les problèmes d'ordonnancement en temps réel et la minimisation d'énergie revêtent un caractère primordial dans la conception des systèmes temps réel à faible puissance. Elle fait l'objet de nombreux travaux de recherche tant au niveau matériel que logiciel. Toutefois, les efforts ont été jusqu'à présent consentis soit sur le matériel, soit sur le logiciel, mais très rarement sur les deux simultanément. Afin d'évaluer la consommation des systèmes temps réel, et de déterminer la méthode optimale pour la minimiser, nous devons disposer d'un outil de modélisation permettant d'évaluer la consommation du circuit dans le cadre de la minimiser de la consommation énergétique un processeur.

Bibliographie

Bibliographie

a[01] J. Elloy. « Les contraintes du temps réel dans les systèmes industriels répartis ». RGE N2/91, pages 26–34, Février 1991

[02] A. Girault, H. Kalla, and Y. Sorel. « Une heuristique d'ordonnancement et de distribution tolérante aux pannes pour systèmes temps réel embarqués. Modélisation des Systèmes Réactifs », MSR'03, Metz, France. Hermes, pages 145–160, Octobre 2003

[03] C. L.Liu and J. W. Layland. «Scheduling algorithms for multiprogramming in a hard-real-time environment». Journal of the ACM, 1973

[04] J. Leung and J. Whitehead. « On the complexity of fixed-priority scheduling of periodic real-time tasks ». Performance Evaluation, 2(237-250), 1982

[05] A.K. Mok. «Fundamental Design Problems for the Hard Real-Time Environments ». PhD thesis, MIT, 1983.

[06] A. K. Mok and M. L. Detouzos. «Multiprocessor scheduling in a hard real-time environment ». In 7th IEEE Texas Conf. on Computing Systems, USA, 1978. IEEE.

[07] F. Cottet, J. Delacroix, C. Kaiser, and Z. Mammeri. «Ordonnancement temps réel - Cours et exercices corrigés ». Hermès, janvier 2000.

[08] POUWELSE J., LANGENDOEN K., SIPS H., « Dynamic voltage scaling on a low-power microprocessor », Proceedings of the 7th annual international conference on Mobile computing and networking (MobiCom'01), New York, NY, USA, ACM Press, p. 251–259, 2001.

[09] ZENG H., FAN X., ELLIS C., LEBECK A., VAHDAT A., « ECOSystem : Managing Energy as a First Class Operating System Resource », Proceedings of the tenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS X), octobre 2002.

[10] AYDIN H., MELHEM R., MOSSÉ D., MEJIA-ALVAREZ P., « Power-Aware Scheduling for Periodic Real-Time Tasks », IEEE Transactions on Computers, vol. 53, n˚5, p. 584-600, 2004.

[11] S.Kamal and Khouri, «Leakage power analysis and reduction during behavioral synthesis », in Transaction on VLSI Systems 2002, Dec. 2002.

[12] RANNOU Robert, OGOR Robert., « Langage Ada et algorithme », 2ème éd. Paris : Hermès, 1993, 462 p. (Traité des Nouvelles Technologies. Série Informatique), ISBN 2-86601-338-7.

[13] Rozenn Texier-Picard « CONVEXITE ET APPLICATIONS », ENS Cachan Bretagne /Université Rennes 1.

[14] C.G. Cassandras and S. Lafortune, Introduction to Discrete Event Systems. Kluwer Academic Publishers, 1999.

[15] Jianfeng Mao «Optimal Dynamic Voltage Scaling in Energy-Limited Nonpreemptive Systems with Real-Time Constraints», IEEE transactions on mobile computing, vol. 6, no. 6, june 2007.

[16] D. Chillet, "Basse consommation dans les systèmes embarqués," in ROSCOFF Ecole thématique, Apr. 2003.

[17] Francis Cottet, Joëlle Delacroix, Claude Kaiser, Zoubir Mammeri, «Ordonnancement temps réel». Edition kermes Science, Janvier 2000

[18] David Decotigny; «Bibliographie d'introduction à l'ordonnancement dans les systèmes informatiques temps réel»; Novembre 2002.

[19] PARAIN F., BANÂTRE M., CABILIC G., HIGUERA T., ISSARNY V. , LESOT J.-P., «Techniques de Réduction de la Consommation dans les Systèmes Embarqués Temps-Réel, » Rapport n˚3932, IRISA, mai 2000

[20] F. Cottet. « Etude de l'application temps réel embarquée» : mission path finder. actes de l'école d'été temps réel, ETR99, pages 109–117, 13-16 sept 1999

[21] P. Guitton, "Estimation et optimisation de la consommation lors de la conception globale des systèmes autonomes," Thèse de doctorat de l'université de Nice Sophia-Antipolis, p. 39, 2004.

[22] Gilbert Cabillic, «Minimisation de la consommation énergétique à l'aide du système d'exploitation», Ecole thématique Architectures des systèmes matériels enfouis et méthodes de conception associées, Roscoff (Finistère) 31 mars - 4 avril 2003

[23] M. A. Cirit, «Estimation dynamic power consumption of CMOS circuits», IEEE Int. Conf. on CAD, Nov. 1987, p.534-537.

[24] Bruno Gaujal, Nicolas Navet, «Ordonnancement sous contraintes de temps et d'énergie», Ecole d'été Temps Réel-Qualité de Service, Tou-louse, 9-12 Septembre 2003, organisée par l'Institut de Recherche en Informatique de Toulouse