

Cycle de formation des ingénieurs en Télécommunications

OPTION

Architecture des Systèmes de Télécommunications

RAPPORT DE PROJET DE FIN D'ETUDES

IMPLEMENTATION D'UN CODEUR VIDÉO H.264 SUR UN MICRO OS EMBARQUÉ

Elaboré par :

Farh ZARROUG

Encadré par :

M. Fethi TLILI

M. Khaled GRATI

Travail proposé et réalisé en collaboration avec



Année universitaire : 2005/2006

Dédicaces

A mes parents

A mes deux sœurs

A ma famille

A mes amis

A tous ceux qui m'aiment et que j'aime

Je dédie ce travail.

...✉ Farh

Avant propos

Ce travail a été effectué dans le cadre de mon projet de fin d'études pour l'obtention du diplôme d'Ingénieur Diplômé en télécommunications option Architecture des Systèmes de télécommunications à l'Ecole Supérieure des Communications de Tunis (SUP'COM). Il a été réalisé en collaboration avec la société EBSYS partenaire technologique de la firme américaine ANALOG DEVICES, Inc (ADI).

Au terme de ce projet, Je tiens à remercier et à exprimer ma profonde gratitude à mes encadreurs Mr. Khaled GRATI et Mr. Fethi TLILI pour leur aide précieuse, leurs conseils et leurs suggestions avisées qui m'ont aidé à mener à bien ce travail.

Je tiens à remercier également tous les ingénieurs et le personnel d'EBSYS, spécialement Mr. Walid BARREH pour ses aides et sa collaboration.

Enfin, je remercie tous ceux qui m'ont aidé de près ou de loin pour effectuer ce travail. Mes remerciements s'adressent aussi à tous mes enseignants pour la qualité de l'enseignement qu'ils nous ont prodigué durant nos études.

Table des matières

Liste des abréviations	vi
Table des figures	viii
Liste des tableaux	ix
Introduction Générale	1
Chapitre I : Etude des OSs Embarqués	3
I.1. Introduction -----	4
I.2. Les systèmes embarqués-----	4
I.2.1. Historique du processeur -----	5
I.2.2. Classes des systèmes embarqués -----	6
I.2.3. Architecture générale des systèmes embarqués -----	7
I.2.4. Co-design des systèmes embarqués-----	8
I.2.5. Contraintes de temps -----	9
I.2.5.1. Systèmes temps-réel strict -----	10
I.2.5.2. Systèmes temps-réel souple-----	10
I.2.5.3. Performance temporelle -----	10
I.2.6. Consommation énergétique -----	11
I.2.7. Mémoire -----	11
I.2.8. Tolérance aux fautes -----	12
I.3. Les OSs embarqués -----	12
I.3.1. Définition d'un OS -----	12
I.3.2. Critères de choix d'un RTOS -----	13
I.3.3. Synchronisation et communication inter-threads -----	15
I.3.3.1. Mécanisme de synchronisation-----	16
I.3.3.2. Mécanisme de communication -----	18
I.3.4. Méthodologie de partitionnement de l'application en threads -----	19
I.3.4.1. Nécessité d'utilisation des threads -----	19
I.3.4.2. Méthodologie de partitionnement-----	19
I.3.5. Ordonnancement des tâches-----	21
I.3.5.1. Mécanisme d'ordonnancement-----	22
I.3.5.2. Algorithmes traditionnels -----	23
I.4. Cas d'étude : VDK (Visual DSP Kernel) -----	25
I.4.1. Le thread -----	26
I.4.2. Les signaux -----	26

I.4.2.1. Les sémaphores-----	27
I.4.2.2. Les messages-----	27
I.4.2.3. Les Events et les Event bits-----	27
I.4.2.4. Les Devices Flags-----	28
I.5. Conclusion-----	28
Chapitre II : Codeur Vidéo H.264	29
II.1. Introduction -----	30
II.2. Compression vidéo H.264-----	31
II.2.1. Historique de compression vidéo-----	31
II.2.1.1. Les normes de L'UIT-T -----	31
II.2.1.2. Les normes de L'ISO/IEC -----	32
II.2.1.3. Les normes communes -----	32
II.2.2. Description technique de la norme H.264-----	33
II.2.2.1. Division en macroblochs dans l'inter-prédiction-----	34
II.2.2.2. Les types d'images et structure du GOP-----	34
II.2.2.3. Prédiction Intra -----	36
II.2.2.4. Prédiction temporelle -----	38
II.2.2.5. Transformation entière -----	39
II.2.2.6. Quantification et parcours des données -----	40
II.2.2.7. Le codage entropique-----	41
II.2.2.8. Le filtrage de boucle -----	41
II.3. Solution Codec sur DSP BF561 -----	41
II.3.1. Architecture de la Plateforme Panview 2.1 -----	41
II.3.2. Description de la solution existante-----	43
II.3.2.1. Architecture globale d'un codec vidéo-----	43
II.3.2.2. Acquisition de la vidéo -----	45
II.3.2.3. Processus de codage H.264-----	46
II.4. Conclusion-----	49
Chapitre III : Conception et mise en œuvre de la solution codeur H.264 sur VDK	
III.1. Introduction -----	51
III.2. Configurations possibles du partitionnement-----	51
III.3. Choix de la configuration optimale -----	55
III.3.1. Conditions de test-----	55
III.3.2. Optimisation des ressources-----	56
III.3.3. Taux d'utilisation du processeur -----	57
III.4. Mise en oeuvre de la nouvelle solution -----	61
III.4.1. Type et ordonnancement des tâches -----	61
III.4.2. Description fonctionnelle-----	62
III.4.2.1. Tâches effectuées par le contrôleur DMA-----	63

III.4.2.2. Tâches effectuées par le processeur -----	65
III.4.3. Mécanisme de communication et de synchronisation entre les threads -----	66
III.5. Evaluation des performances de la solution adoptée -----	67
III.5.1. Conditions et hypothèses de la simulation-----	67
III.5.2. Simulation et analyse des résultats-----	69
III.6. Conclusion-----	71
Conclusion Générale	72

Liste des abréviations

A

- AVC: Advanced Video Coding.
- API: Application Programming Interface.
- ARPS: Adaptive Rood Pattern Search.
- ARP: Adaptive Rood Pattern.

C

- CABAC: Context Adaptative Binary Arithmetic Coding.
- CAO: Conception Assistée par Ordinateur.
- CAVLC: Context Adaptative Variable Length Coding.
- CIF: Common Intermediate Format.
- CPU: Central Processing Unit.

D

- DM: Deadline Monotonic.
- DSP: Digital Signal Processor.
- DVB: Digital Video Broadcasting.
- DVD: Digital Versatile Disc.
- DMA: Direct Memory Access.

F

- FPGA: Field Programmable Gate Array.
- FSDRAM: Frame SDRAM.

G

- GOP: Group Of Pictures.

I

- IP: Internet Protocol.
- ISO/IEC: International Standards Organisation / International Electrotechnical Commission.
- ISR: Interrupt Service Routine.

J

- JVT: Joint Video Team.

L

- LCD: Liquid Crystal Display.
- LLF: Least Laxity First.

M

- MIPS: Million of Instructions Per Second.
- MPEG: Motion Picture Experts Group.
- MP3: MPEG-1 LAYER 3.
- MB: Macro Bloc.

N

- NTSC: National Television Standard Commitee.
- NALU: Network Abstraction Layer Unit.

O

- OS: Operating System.

P

- PC: Program Counter register.
- PDA: Personnel Digital Assistant.
- PAL: Phase Alternating Line.

- PIXEL: PI(X)cture ELement.
- PPI: Parallel Peripheral Interface.

Q

- QCIF: Quarter Common Intermediate Format.

R

- RMS: Rate Monotonic Scheduling.
- RNIS: Réseau Numérique à Intégration de Service.
- ROM: Read Only Memory.
- RTC: Réseau Téléphonique Commuté.
- RTOS: Real Time OS.

S

- SAD: Sum of the Absolute Differences.
- SIF : Source Intermediate Format.
- SJF: Shortest Job First.
- SP: Stack Pointer register.
- SECAM: Séquentiel Couleur A Mémoire.
- SDRAM: Synchronous Dynamic Random Access Memory.

U

- UIT-T: Union Internationale de Télécommunications, standardisation du secteur des Télécommunications.
- UVLC: Unified Variable Length Coding.
- URP: Unit-size Rood Pattern.

V

- VDK: Visual DSP Kernel.
- VHS: Video Home System.
- VLC: Variable Length Coding.

Table des figures

<i>Figure I. 1: Architecture de la machine à carte performée</i>	5
<i>Figure I. 2: Topologie d'un Système Embarqué</i>	7
<i>Figure I. 3: Les étapes du Co-Design</i>	8
<i>Figure I. 4: Les états d'un thread</i>	16
<i>Figure I. 5: Problème de cinq philosophes</i>	18
<i>Figure I. 6: Schéma d'un codeur vidéo H.264</i>	20
<i>Figure I. 7: Les paramètres temporels d'une tâche</i>	23
<i>Figure II. 1: Les principaux standards normalisés par l'UIT-T et par l'ISO/IEC</i>	31
<i>Figure II. 2: Schéma fonctionnel d'un codeur vidéo H.264</i>	33
<i>Figure II. 3: Les différentes partitions et sous-partitions dans la norme H.264</i>	34
<i>Figure II. 4: Structure générale du GOP</i>	35
<i>Figure II. 5: Prédiction d'un bloc 4x4 en mode intra</i>	37
<i>Figure II. 6: Modes d'intra-prédiction 16x16 luma</i>	38
<i>Figure II. 7: Estimation de mouvement</i>	39
<i>Figure II. 8: Balayage en zigzag</i>	40
<i>Figure II. 9: Architecture de la plateforme vidéo</i>	42
<i>Figure II. 10: Organigramme de l'application</i>	44
<i>Figure II. 11: Flux de données avec l'extérieur</i>	45
<i>Figure II. 12: Lecture et écriture des images brutes</i>	46
<i>Figure II. 13: Processus de codage H.264</i>	48
<i>Figure III. 1: Schéma du codeur selon l'architecture 1</i>	52
<i>Figure III. 2: Schéma du codeur selon l'architecture 2</i>	53
<i>Figure III. 3: Schéma du codeur selon l'architecture 3</i>	54
<i>Figure III. 4: Schéma du codeur selon l'architecture 4</i>	54
<i>Figure III. 5: Les ressources utilisées dans la première architecture</i>	56
<i>Figure III. 6: Les ressources utilisées dans la troisième architecture</i>	56
<i>Figure III. 7: Les ressources utilisées dans les architectures 2 et 4</i>	57
<i>Figure III. 8: Visualisation de l'historique sous ADSP</i>	58
<i>Figure III. 9: Charge du processeur dans la configuration 1</i>	58
<i>Figure III. 10 : Charge du processeur dans la configuration 2</i>	59
<i>Figure III. 11: Charge du processeur dans la configuration 3</i>	59
<i>Figure III. 12: Charge du processeur dans la configuration 4</i>	60
<i>Figure III. 13: Graphe de dépendance entre les tâches dans un codeur H.264</i>	61
<i>Figure III. 14: Schéma synoptique d'un codeur vidéo H.264</i>	62
<i>Figure III. 15: Organisation de SDRAM</i>	64
<i>Figure III. 16: Ordre d'exécution des threads</i>	68
<i>Figure III. 17: Ordre d'exécution des threads sous VDK</i>	69

Liste des tableaux

<i>Tableau III. 1: Les temps de transfert des MBs</i> -----	64
<i>Tableau III. 2 : Mécanisme de communication et de synchronisation entre les threads</i> -----	66

Introduction Générale

L'émergence de la technologie des microprocesseurs et l'apparition de plusieurs organismes de standardisation dans le domaine de compression vidéo telque l'ISO/IEC et l'UIT-T ont constitué un tournant technologique important qui a permis aux systèmes embarqués de communication audio-visuelle de devenir une réalité très vite perceptible. Les développements technologiques ont rendu ces microprocesseurs plus flexibles et moins chers. De ce fait, les systèmes embarqués multimédias à base de microprocesseurs ont été introduits dans de nombreux domaines d'application tels que la télévision numérique, la vidéoconférence, la visiophonie, la télé-médecine, les terminaux de communication sans fils, les systèmes de vidéosurveillance, etc. Ils sont devenus dans ces dernières années un produit de grand public. Leur succès est en grande partie dû à leurs facilités d'utilisation, aux nombreux services qu'ils offrent, à la baisse de prix et à l'utilisation optimale et efficace des ressources et d'énergie.

Pour minimiser la consommation en énergie, optimiser l'utilisation des ressources et améliorer les performances du système, deux approches complémentaires sont possibles :

- Améliorer les performances du matériel (augmenter la taille de la batterie et de la mémoire, augmenter la capacité de traitement du processeur, etc.).
- L'optimisation du logiciel afin de diminuer le coût énergétique, optimiser l'utilisation de la mémoire et enfin la conception de stratégies logicielles exploitant les fonctionnalités du matériel.

L'optimisation du logiciel consiste à privilégier les instructions moins gourmandes en énergie afin de diminuer la consommation énergétique de l'exécution du programme.

Dans la deuxième approche s'intègre notre projet de fin d'études intitulé « implémentation d'un codeur vidéo H.264 sur un micro OS embarqué » en collaboration avec EBSYS. L'objectif de ce projet est d'étudier la plateforme Panview d'ANALOG DEVICES sur laquelle est implémenté un codeur vidéo H.264, étudier les méthodologies de partitionnement pour trouver le meilleur compromis logiciel/matériel afin d'augmenter les performances du codeur et de rendre sa mise à jours plus pratique et concevoir et mettre en œuvre la nouvelle solution sur VDK.

Ce rapport est organisé en trois chapitres, dans le premier nous allons étudier les OSs embarqués en évoquant les critères de choix d'un système d'exploitation temps réel tout en

respectant la fonctionnalité globale du système et les contraintes imposées par l'environnement. Le second chapitre sera consacré à la présentation des spécificités de la norme du codage vidéo H.264 et la description de la solution existante. Le dernier chapitre a pour objet de mettre en œuvre et d'évaluer les performances de la solution codeur sur VDK.

Chapitre I

Etude des OSs Embarqués

I.1. Introduction

Parmi les domaines de l'informatique qui connaissent un regain d'intérêt figurent les systèmes embarqués, qui font partie intégrante de notre vie quotidienne sans qu'on en ait toujours conscience. En tant que particulier, ils nous permettent les communications sans fil (téléphonie, etc.), nous assistent dans la conduite de nos véhicules, contribuent à notre confort en régulant intelligemment les différents paramètres de notre environnement (température, humidité, etc.). Dans les entreprises, ces systèmes contribuent à la surveillance des installations à risques, permettent le suivi temps réel de production, contrôlent les différents flux de données ou de matières des usines, assurent une traçabilité complète de la production.

Cette diversité des applications et la précision des résultats fournis par les systèmes embarqués, apportent aux concepteurs énormément de problèmes lors du co-design surtout dans la phase de la sélection matériel et logiciel pour que ce système soit capable d'effectuer convenablement ses tâches dédiées.

Dans ce contexte s'intègre notre premier chapitre, qui consiste à faire une étude sur les systèmes embarqués commençant par connaître son histoire, ses caractéristiques et les étapes de co-design.

Une deuxième partie sera consacrée, en premier lieu, à présenter les critères de choix d'un OS temps réel en respectant la fonctionnalité globale du système et les contraintes imposées par l'environnement. En deuxième lieu nous allons présenter le mécanisme d'ordonnancement temps réel, la synchronisation, la communication et la méthodologie de partitionnement de l'application en threads.

Dans la dernière partie nous entamerons l'étude d'un système d'exploitation multi-threads (VDK Visual DSP Kernel) intégré dans l'environnement ADSP++ d'Analog Devices.

I.2. Les systèmes embarqués

Le terme système embarqué dénote un système autonome, disposant de l'ensemble des éléments physiques nécessaires à son fonctionnement (processeur, mémoire, périphériques d'entrées/sorties), utilisant conjointement du logiciel (système s'exploitation, drivers etc.) pour mettre en œuvre une fonction spécifique. Il est en forte interaction avec l'environnement extérieur dans lequel il évolue.

Les contraintes lors de la construction de tels systèmes sont relatives au temps, imposées par l'interaction du système avec son environnement et aussi au caractère limité et/ou périssable des ressources dont dispose le système (énergie, mémoire, etc.)[1]. À ces contraintes s'ajoutent des contraintes de sûreté de fonctionnement plus ou moins importantes selon le type d'utilisation du système embarqué.

I.2.1. Historique du processeur

L'histoire des systèmes électroniques a commencé en 1833, lorsque Babbage invente le concept des calculateurs programmables. Mais cette architecture présente plusieurs inconvénients (capacité de calcul, grande taille, etc.), ceci encourage plusieurs travaux de recherche pour améliorer les performances de l'architecture actuelle, parmi lesquels, les travaux de Von Neumann qui mènent en 1945 à l'introduction d'une nouvelle architecture du processeur programmable qui était plus souple et plus efficace en terme de capacité de traitement.

En 1946, ENIAC introduite la première machine électronique constituée de 18.000 tubes électroniques. En 1949, IBM invente la première machine de calcul utilisant des cartes perforées. Les périphériques de cette machine se résumaient en un lecteur de cartes, un performateur de cartes et une imprimante. Le mode de fonctionnement était assez simple, vu de l'utilisateur, puisque l'exécution d'un programme consistait à mettre dans le lecteur de cartes un paquet contenant la forme binaire du programme suivie des données. Après la lecture du programme, l'ordinateur en lançait l'exécution, les résultats étant obtenus sous forme d'un paquet de cartes perforées, ou de lignes imprimées (figure I.1).

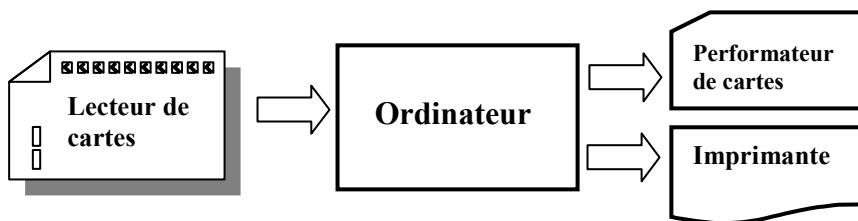


Figure I. 1 : Architecture de la machine à carte perforée

Les systèmes numériques embarqués ont vu leur importance progresser au rythme de l'importance prise par les microprocesseurs, en 1971, Intel invente le premier microprocesseur 4 bits 4004 vendu à 200\$ [2], ses performances étaient trop limitées puisqu'il fonctionne à une fréquence qui ne dépasse pas une centaine de KHz (92.5 KHz) [2]. Comme une deuxième génération de microprocesseurs d'Intel, était le 8086 (famille x86) inventé en 1978, la taille de mot « Size Word » qu'il peut interpréter est 16 bits et sa fréquence de fonctionnement est de 4.77 MHz.

En 1981, IBM a mis sur le marché le premier IBM-PC dans lequel intégré un microprocesseur 8088.

La baisse des prix de semi-conducteur et la maîtrise de la technologie de microélectronique, ont encouragé la formation des plusieurs firmes internationales qui sont entrées en masse sur le marché des processeurs (Motorola, Zilog, Texas Instruments, Analog Devices, Thomson, Philips,...).

Le 21^{ème} siècle ou le siècle de l'informatique a vu une explosion du marché des téléphones portables et de l'Internet, et le développement des applications multimédia qui a alléché des nouvelles compagnies d'entrer sur le marché.

Lois d'évolution des technologies : sont vérifiées depuis 1970 jusqu'à 2000

- Loi de Gordon Moore : Pour une surface de silicium donnée, on double le nombre de transistors intégrés tous les 18 mois.
- Loi de Joy : La puissance CPU en MIPS double tous les 2 ans.
- Loi de Ruge : On a besoin d'une bande passante de 0.3 à 1 Mb/s par MIPS.

I.2.2. Classes des systèmes embarqués

On distingue l'existence de quatre familles, chaque type se différencie des autres par ses fonctionnalités [2] :

- Calculateur générique (General Computing) : Dédié pour le traitement de données avec interface utilisateur interactive, similaire à une application de bureau mais empaquetée dans un système embarqué.

Exemple : Jeu vidéo, PDA (Annexe A).

- Système de commande (Control System) : Permettant le contrôle de systèmes en temps réel.

Exemple : Moteur d'automobile, procès chimique, procès nucléaire, système de navigation aérien.

- Système de traitement du signal (Signal Processing) : Permettant le calcul sur des grandes quantités de données à fort débit.

Exemple : Compression vidéo, radar, sonor, etc.

- Communication et réseau (Communication & Networking) : Ce type de système permettant la transmission des informations et la commutation.

Exemple : Commutateurs, routeurs, téléphone, etc.

I.2.3. Architecture générale des systèmes embarqués

Un ordinateur se compose de trois couches: Application, Système d'exploitation (OS : Operating System) et Matériel. De même, un système embarqué dispose de trois couches, comme le montre la figure I.2. Chaque couche a la même fonctionnalité qu'un système normal. Mais, Il y a des différences de sous composants du système. Deux premières couches, Il s'agit du logiciel, il est possible de modifier ces composants et d'ajouter ou supprimer des modules au besoin dépendant de but du système. Cependant, ce n'est pas un système qui contient tous les composants comme le système complet car le but de conception est de servir quelques tâches spécifiques et de concentrer à un unique travail. Le système d'exploitation est une couche logicielle sur laquelle on va placer l'ensemble des applications lancées par les utilisateurs, il comprend les bibliothèques pour le développement, des drivers permettant aux applications d'accéder à des périphériques évolués. La conception d'un OS se base forcément sur le matériel visant à optimiser la performance.

La dernière couche est la couche matérielle composée d'un ensemble des éléments physiques employés pour le traitement de données. Les composants matériels sont limités et sont classés en quatre classes suivantes :

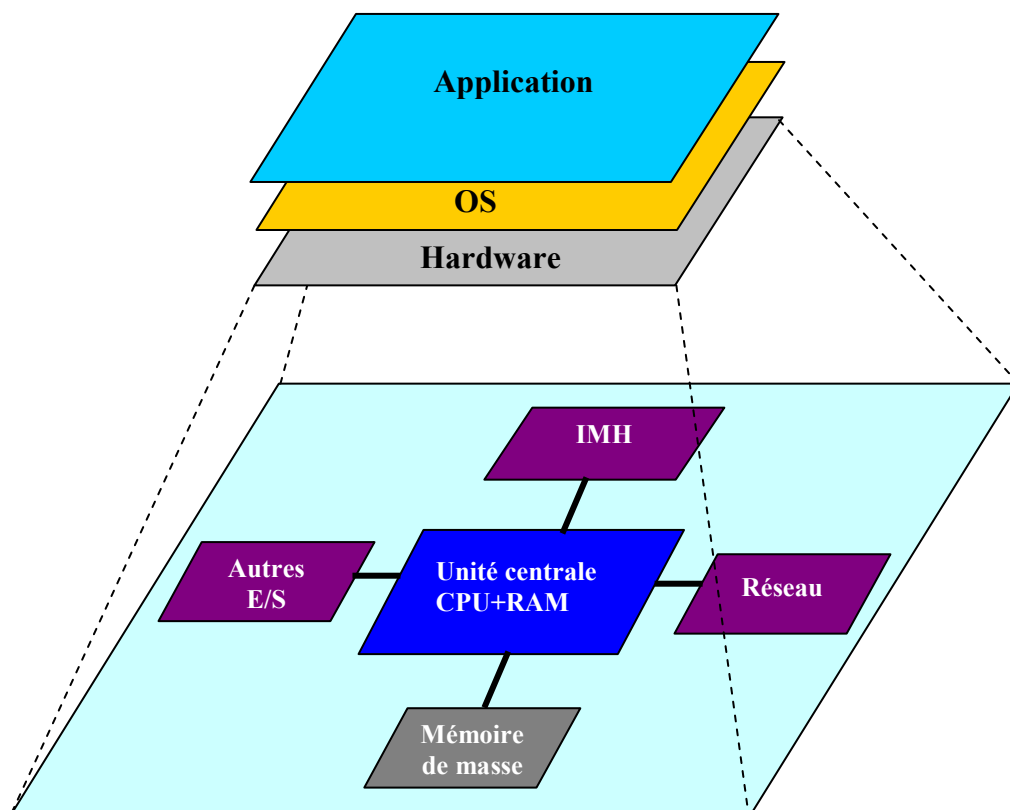


Figure I. 2 : Topologie d'un Système Embarqué

- Entrées :
 - Les capteurs/convertisseurs (pression, audio, température), les lecteurs de codes barres.
 - Les claviers, les boutons poussoirs ou télécommandes (infrarouge, bluetooth, etc.).
- Sorties :
 - Le système d'alarme ou de synthèse vocale, les écrans et les afficheurs LCD (Annexe A).
 - L'imprimante en tous genres comme papier, étiquettes, photos.
- Mémoire de masse :
 - Le disque dur : Microdrive à la taille environ 2,5-3,5 inches.
 - La mémoire flash : FlashDisk, CompactDisk, DiskOnChip, SDCard (Annexe A).
 - L'utilisation de ROM : Disque virtuel CD, DVD, disquette.
- IHM (Interface Homme-Machine) : Un dispositif qui sert à communiquer entre l'homme et la machine. Un exemple réaliste de IHM est l'écran avec les dispositifs «TouchScreen» qui est visé aux PDAs (Personal Digital Assistant).

I.2.4. Co-design des systèmes embarqués

Le terme "co-design" est apparu au début des années 1990 pour marquer une nouvelle façon de penser à la conception des circuits intégrés et des systèmes [3], [2]. La "conception conjointe du logiciel et du matériel" était devenue nécessaire pour répondre aux exigences du marché des systèmes intégrés. En effet, l'émergence des systèmes multimédia (téléphones portables, consoles de jeu, etc.) entraînait une plus grande complexité de la partie électronique et la concurrence économique imposait un temps de conception encore plus court.

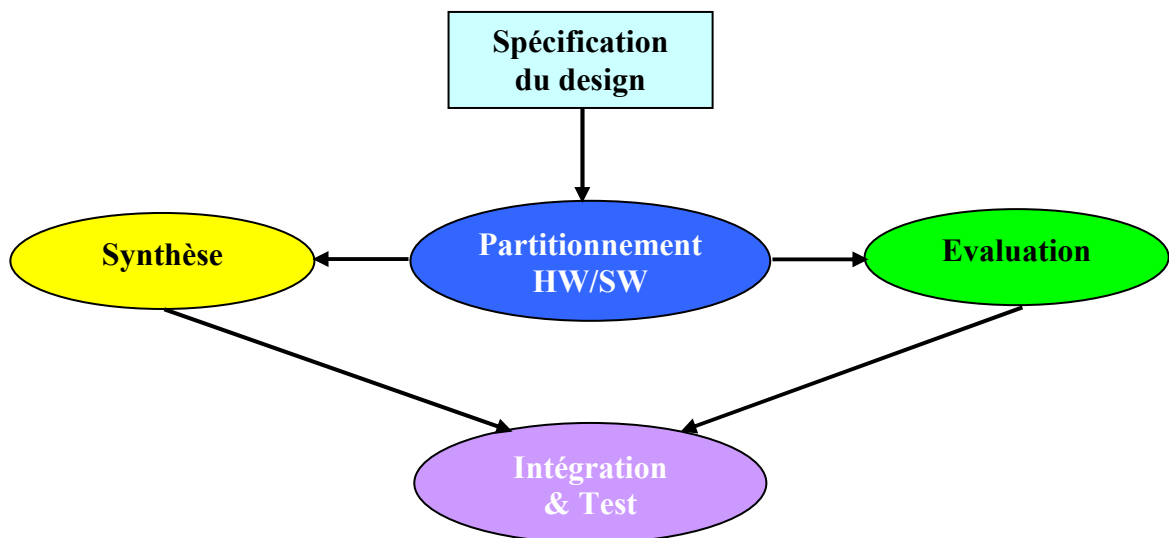


Figure I. 3 : Les étapes du Co-Design

La figure I.3 montre les étapes du co-design :

- Spécification du design : Savoir la liste des fonctionnalités du système d'une façon abstraite.
- Partitionnement HW/SW : Le découpage ou partitionnement logiciel/matériel est une phase importante de la conception de systèmes (et plus particulièrement de la phase d'exploration d'architectures) et consiste à rechercher le meilleur compromis logiciel/matériel. C'est dans cette phase, que sont effectués les choix menant à une réalisation soit matérielle, soit logicielle des différentes parties constituant le système en respectant les paramètres suivantes :
 - Les contraintes statiques qui sont : Le coût financier, la consommation énergétique, la surface de silicium maximale, la taille du code et de la mémoire.
 - Les contraintes dynamiques qui sont essentiellement les contraintes temporelles.
 - La sûreté du fonctionnement, vérifier le bon fonctionnement du système.
 - La testabilité qui va ajouter des composants matériels supplémentaires ou l'ajout des instructions.
 - La réutilisation liée au logiciel, consiste à l'utilisation d'un code dans une autre application semblable.

En règle générale, le logiciel est utilisé pour réduire les coûts de conception et le matériel pour augmenter les performances.

- Synthèse : Synthèse matérielle, choix du matériel et CAO électronique.
- Evaluation : Compilation logicielle.
- Intégration et test : Réalisation matérielle et implémentation des modules logiciels, tests expérimentaux et debugage.

I.2.5. Contraintes de temps

Beaucoup de systèmes embarqués interagissent directement avec leur environnement via des capteurs/actionneurs ou un réseau de communications sans fil. Ces interactions contraignent les temps de réponse du système embarqué de manière plus ou moins forte selon le domaine d'applications visé. On parle alors d'un système temps-réel, dans le sens où le temps de livraison des résultats d'un calcul fait partie intégrante de la spécification de ce dernier, au même titre que le résultat lui-même.

I.2.5.1. Systèmes temps-réel strict

Dans les systèmes temps-réel strict, ou dur, le non respect des contraintes temporelles du système, le plus souvent exprimées sous la forme d'échéances de terminaison, constitue une défaillance de l'application. Dans le cadre d'applications critiques, telles que par exemple le contrôle de centrales nucléaires, une telle défaillance peut avoir des conséquences catastrophiques, telles que la mise en danger de vies humaines ou des pertes financières importantes.

Étant données les conséquences de non respect d'une échéance dans les systèmes temps-réel strict, il est nécessaire pour ces systèmes de pouvoir vérifier avant leur exécution que toutes les échéances seront toujours respectées. Cette vérification est le rôle des méthodes d'analyse d'ordonnabilité (théorie de l'ordonnancement), qui requièrent une connaissance du pire comportement temporel du système [1].

I.2.5.2. Systèmes temps-réel souple

Dans les systèmes temps-réel souple, bien que l'instant de livraison d'un résultat soit important, la violation des contraintes temporelles du système est acceptable si elle reste rare. Cette tolérance est acceptée car les applications concernées ne relèvent pas du domaine des applications critiques. Les exemples typiques d'applications ayant des contraintes temps-réel souples sont les applications multimédias à flux continus. Le système vise au respect des contraintes temporelles dans la délivrance des flux de données afin de garantir la qualité des images et du son ; toutefois, les contraintes temporelles peuvent être adaptées puisque une dégradation de la qualité des données ne sera que faiblement perçue par l'utilisateur.

I.2.5.3. Performance temporelle

À cause de la demande toujours croissante de la puissance du traitement des applications, la performance globale temporelle du système embarqué est également un critère de conception important. Aussi, dans la conception d'un système embarqué la performance temporelle des mécanismes est une préoccupation constante. Au niveau logiciel ce travail porte à la fois sur les algorithmes afin de diminuer leur complexité. Parfois, l'utilisation de processeurs spécialisés peut également permettre d'atteindre la performance désirée. Une autre approche consiste à proposer un ordonnancement qui permette de réduire le temps de réponse des traitements temps-réel.

I.2.6. Consommation énergétique

Une grande majorité des systèmes embarqués (téléphones cellulaires, ordinateurs de poche, etc.) sont confrontés au problème de l'autonomie. Aussi, afin d'étendre l'autonomie de fonctionnement d'un système deux approches complémentaires sont possibles : augmenter la capacité de stockage des batteries ou réaliser un système embarqué à faible consommation énergétique. Dans le cadre de cette dernière approche, plusieurs méthodes sont alors envisagées qui touchent à la fois le domaine de l'électronique et du logiciel : la conception de composants électroniques consommant le minimum d'énergie, l'optimisation du logiciel afin de diminuer le coût énergétique de son exécution et enfin la conception de stratégies logicielles exploitant les fonctionnalités du matériel.

L'optimisation du logiciel consiste à privilégier les instructions moins gourmandes en énergie afin de diminuer la consommation énergétique de l'exécution du programme. La conception d'une stratégie logicielle exploitant des fonctionnalités du matériel afin de diminuer la consommation énergétique du système touche principalement à l'ordonnancement. Tout d'abord, certains processeurs offrent différents modes d'exécution. Outre le mode d'exécution nominal, un mode veille permet à la fois d'endormir à faible coût le processeur et également de le réveiller de manière rapide. Le système d'exploitation aura donc dans ce cas la responsabilité de décider, de passer le processeur en mode veille lors des périodes d'inactivité du système. Une autre approche possible consiste à adapter dynamiquement la vitesse d'exécution du processeur en agissant par exemple sur sa tension d'alimentation. En effet, le fait de réduire la tension du processeur réduit considérablement la consommation énergétique.

I.2.7. Mémoire

La mémoire est une ressource limitée dans un grand nombre de systèmes embarqués (de quelques Kilo-octets dans une carte à puce à quelques Méga-octets dans un téléphone portable) et par conséquent une bonne utilisation de la ressource mémoire est cruciale pour ces systèmes. Les méthodes permettant d'utiliser des systèmes à faible capacité mémoire vont de l'utilisation de codes interprétés compacts, à l'utilisation d'algorithmes de compression, en passant par des algorithmes d'allocation dynamique de mémoire optimisés pour limiter le morcellement de la mémoire, ou fragmentation.

I.2.8. Tolérance aux fautes

Certains systèmes embarqués doivent pouvoir remplir leurs fonctions malgré la présence de fautes, qu'elles soient d'origine physique ou humaine. Les moyens pour la sûreté de fonctionnement et plus spécifiquement les méthodes de tolérance aux fautes, permettant au système de remplir ses fonctions en dépit des fautes pouvant affecter ses composants. Par exemple, en ce qui concerne les fautes d'origine physique, il est nécessaire de détecter les erreurs, par l'utilisation de méthodes telles que les codes détecteurs d'erreurs, puis d'effectuer un recouvrement d'erreur permettant au système de continuer à remplir ses fonctions malgré l'erreur. En particulier, dans les systèmes temps-réel strict, il est nécessaire d'intégrer les mécanismes de tolérance aux fautes dans l'analyse d'ordonnabilité du système.

I.3. Les OSs embarqués

I.3.1. Définition d'un OS

Un OS (Operating System) ou système d'exploitation est un programme qui assure la liaison entre le matériel et les applications (traitement de texte, jeux, etc.) lancées par l'utilisateur. Le système d'exploitation fournit un certain nombre d'outils pour gérer la machine. Il assure l'initiation du système après une mise sous tension.

Grâce à des drivers (gestionnaires de périphériques) le système d'exploitation peut gérer les périphériques, en assurant des opérations aussi que l'affichage des caractères sur l'écran ou bien lecture de clavier. La communication avec le système d'exploitation s'établit par l'intermédiaire d'un langage de commandes et un interpréteur de commandes, cela permet à l'utilisateur de piloter les périphériques en ignorant tout les caractéristiques du matériel qu'il utilise.

On distingue plusieurs types de systèmes d'exploitation, parmi lesquels nous citons:

- Système multitâche : Est un OS multitâche permet de partager le temps du processeur entre plusieurs programmes, ainsi ceux-ci sembleront s'exécuter simultanément. Pour réaliser ce processus, les applications sont découpées en séquence d'instructions que l'on appelle tâches ou threads/processus. Ces tâches seront exécutées par ordre et séquentiellement selon la priorité et son emplacement dans la file d'attente du processeur (Ready Queue).
- Système préemptif : Est un système d'exploitation multi-tâches ou chaque tâches en cours d'exécution peut être interrompu au milieu de l'exécution, généralement sont les OSs temps réel.

- Système coopératif : Est un système multitâches ou l'exécution d'une tâche ne peut pas être suspendu, exemple Windows 3.x et Windows 95/98.
- Systèmes multiprocesseurs : Ces systèmes sont nécessairement multitâches puisqu'on leur demande d'une part de pouvoir exécuter simultanément plusieurs applications, mais surtout d'organiser leur exécution sur les différents processeurs (qui peuvent être identiques ou non). Ces systèmes peuvent être soit architecturés autour d'un processeur central qui coordonne les autres processeurs, soit avec des processeurs indépendants qui possèdent chacun son système d'exploitation, ce qui leur vaut de communiquer entre eux par l'intermédiaire de protocoles.
- OS embarqué : Est un système d'exploitation léger dédié pour une application précise, il est différent des systèmes d'exploitations généraux et ne possède pas toujours un système de fichier.
- RTOS (Real Time Operating System) : Est un système d'exploitation léger permettant de garantir la disponibilité des ressources matérielles pour des tâches définies, dans des limites temporelles précises. Dans un système d'exploitation temps réel, les temps d'acquisition et de traitement de l'information doivent être inférieurs aux temps d'occurrence de cette information. La majorité des OSs temps réel sont des OSs embarqués.

I.3.2. Critères de choix d'un RTOS

Les systèmes temps réel ont depuis toujours acquis une importante place dans le développement des systèmes informatiques. Ils recouvrent un spectre d'applications très étendu qui va du simple contrôle de systèmes de commande, au contrôle du trafic aérien, des télécommunications, en passant par les systèmes de défense militaire, installations nucléaires et bien d'autres. Les systèmes temps réel sont des systèmes de traitement dont la validité est conditionnée non seulement par la correction des résultats mais surtout par les dates auxquelles ceux-ci sont délivrés. En effet, ces systèmes sont essentiellement caractérisés par des contraintes de temps sur les actions à entreprendre, qu'il faut respecter de manière plus ou moins critique.

Grand nombre d'applications temps réel nécessitent pour leur exécution, des systèmes très stricts d'une fiabilité incontestable car la moindre erreur dans le respect des contraintes temporelles peut entraîner la catastrophe. Pour ce faire, aussi bien la machine que son système d'exploitation doit être adapté à ce type d'applications. Ainsi, les systèmes qui supportent ce genre d'applications doivent offrir les caractéristiques suivantes :

- Fiabilité (reliability) : Faculté de pouvoir être utilisé sans intervention extérieure s'exprime en fonction de la disponibilité, dépend du hardware autant que du software.
- Prédicibilité (predictability) : Le degré de confiance que l'on peut avoir dans la prévision du temps de réponse.
- Performances : Caractérise la vitesse à laquelle le système va fournir sa réponse, dépend du hardware autant que du software.
- Compacité de l'empreinte (compactness) : Utilisation optimale des ressources.
- Possibilité d'appliquer un facteur d'échelle (scalability) : Utilisation optimale des ressources en cas de modification de la taille du système et optimisation des conditions de développement.

Un RTOS [4], est souvent organisé autour d'un noyau fournissant des algorithmes pour l'ordonnancement et la gestion des ressources, il permet d'exécuter toutes les tâches et offre éventuellement les services suivants :

- Communication entre les tâches : L'échange des données entre les tâches est réalisé soit par des tampons partagés soit par le mécanisme de messages.
- Synchronisation des tâches : Grâce aux sémaphores et aux verrous on peut synchroniser deux threads.
- Gestion et ordonnancement des tâches : L'ordonnanceur (scheduler en anglais) permet l'ordonnancement des tâches en respectant ces échéances pour satisfaire la condition de temps réel.
- Gestion de la mémoire : Gérer d'une façon appropriée et optimisée l'utilisation de la mémoire afin de réduire le coût de réalisation du système.
- Gestion des interruptions et E/S physiques : Garantir de servir toutes les interruptions pour synchroniser le système avec l'environnement extérieur.
- E/S logiques et gestion des périphériques : Etre capable d'accéder aux différents Entrées/sorties et permettant grâce aux drivers de gérer les périphériques.
- Traitement des erreurs et des exceptions : Les exceptions sont des interruptions internes générées en cas d'erreur, par exemple une division par 0, dépassement de la capacité, accès à des zones interdites dans la mémoire. Un bon RTOS doit traiter rapidement et effectivement ces exceptions.
- Gestion du temps : Diminuer le temps de changement du contexte, le temps d'accès aux périphériques et à la mémoire.
- Comportement prévisible : Le comportement d'OS devrait être connu et prévisible.

Exemples des OSs temps réel

- QNX est un micro-noyau temps réel, développé par la société canadienne QNX Software. QNX est un logiciel commercial conforme à la norme POSIX (Annexe A), qui se rapproche du monde de l'open source [5].
- Windows CE et Embedded Windows NT, sont conçus par Microsoft. Leur utilisation est actuellement limitée à l'équipement de nombreux PDAs ou assistants personnels.
- Jaluna, est un système d'exploitation temps réel français, construit autour du noyau libre C5. La dernière version 5 de Chorus a été abandonnée par Sun.
- ECOS (Embedded Cygnus Operating System), initialement développé par la société Cygnus et rattaché aujourd'hui à la société Red Hat Software. Ce système est bien adapté aux applications embarquées.
- VxWorks, est un exécutif temps réel qui nécessite peu d'espace mémoire, développé par la société Wind River, pour des besoins de temps réel dans les systèmes embarqués.
- RTEMS (Time Executive for Multiprocessor Systems), est un système de compilation croisée à la norme Ada 95 dédié aux systèmes embarqués temps réel. En fait, c'est un projet qui a été abandonné par l'armée Américaine.

I.3.3. Synchronisation et communication inter-threads

Tout d'abord nous allons définir un thread, c'est un programme responsable à la réalisation d'une tâche, il fonctionne indépendamment des autres threads mais il peut se communiquer et se synchroniser avec eux grâce aux plusieurs mécanismes de communication et de synchronisation (sémaphore, message, etc.). Un thread possède quatre états, comme illustré dans la figure I.4:

- état prêt : lorsque les ressources qu'il a demandé sont libres.
- état bloqué : lorsque les ressources qu'il a demandé ne sont pas disponibles.
- état d'exécution : lorsqu'il est en cours d'exécution
- état inexistant : le thread n'est pas encore créé.

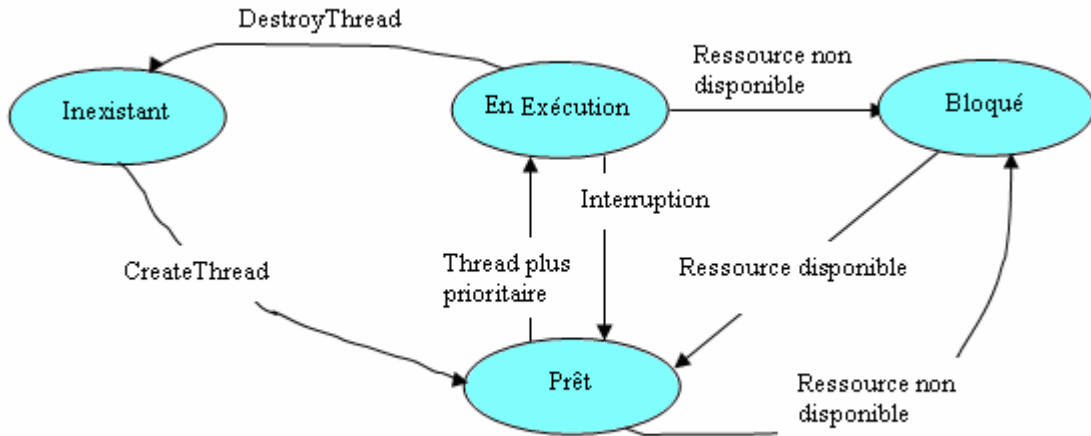


Figure I. 4: Les états d'un thread

I.3.3. 1. Mécanisme de synchronisation

a- Les verrous :

C'est un mécanisme proposé pour permettre de résoudre l'exclusion mutuelle d'accès à une ressource. Un verrou est un objet système sur lequel deux opérations sont définies. Un tel objet peut s'assimiler à une ressource logicielle, les opérations permettant d'acquérir ou de libérer cette ressource sont :

- Verrouiller (v) : permet au processus d'acquérir le verrou v s'il est disponible. S'il n'est pas disponible, le processus est bloqué en attente de la ressource.
- Déverrouiller (v) : permet au processus de libérer le verrou v qu'il possédait. Si un ou plusieurs processus étaient en attente de ce verrou, un seul de ces processus est réactivé et reçoit le verrou.

b- Les sémaphores :

Un sémaphore est un mécanisme proposé par E.W.Dijkstra en 1965 et qui est un peu plus général que le verrou. Il se présente comme un distributeur de jetons, mais le nombre de jetons est fixe et non renouvelable, les processus doivent restituer leur jeton après utilisation. S'il y a un seul jeton en circulation, on retrouve le verrou. Deux opérations pour acquérir ou libérer un sémaphore:

- Down (s) : permet à un processus d'obtenir un jeton, s'il y en a de disponibles. Si aucun n'est disponible, le processus est bloqué.
- Up (s) : permet à un processus de restituer un jeton. Si des processus étaient en attente de jeton, l'un d'entre eux est réactivé et le reçoit.

Notons qu'un sémaphore peut être vu comme un couple constitué d'un entier, encore appelé le niveau du sémaphore et d'une file d'attente de processus. Le niveau est le nombre de jetons encore disponibles. Il est évident que si le niveau est positif, la file d'attente est vide. Parfois nous utilisons les valeurs négatives du niveau pour représenter le "déficit" en jetons, c'est à dire le nombre de processus en attente de jeton.

À la création d'un sémaphore, il faut décider du nombre de jetons dont il dispose. On voit que si une ressource est à un seul point d'accès (critique), le sémaphore doit avoir initialement un jeton, qui sera attribué successivement à chacun des processus demandeurs. Si une ressource est à n points d'accès, c'est-à-dire, peut être utilisée par au plus n processus à la fois, il suffit d'utiliser un sémaphore initialisé avec n jetons. Dans les deux cas, un processus qui cherche à utiliser la ressource, demande d'abord un jeton, puis utilise la ressource lorsqu'il a obtenu ce jeton et enfin le rend lorsqu'il n'a plus besoin de la ressource.

c- La notion d'interblocage :

Comme les utilisations de ressources critiques sont souvent imbriquées, il peut en résulter une défaillance du système par interblocage.

Supposons que les tâches T1 et T2 ont besoin chacune de deux ressources exclusives R1 et R2 et que ces ressources ne peuvent pas être retirées à une tâche qui les utilise en section critique. Si T1 demande R1 et R2 et si T2 demande au contraire R2 puis R1, les tâches peuvent attendre mutuellement indéfiniment. Prenant l'exemple de cinq philosophes (figure I.5), si par exemple chaque philosophe prend la fourchette qui se situe à son droit, en fait il ne reste plus de fourchette sur la table et par conséquent aucune personne ne peut manger. En effet chaque philosophe a besoin des deux fourchettes pour qu'il puisse manger et donc ils vont attendre indéfiniment.

Mais la majorité des systèmes d'exploitation offrent des régions protégées dans lesquelles un thread et/ou processus peut occuper le processeur durant un nombre de ticks pour qu'il puisse prendre les ressources qu'il ait besoin simultanément (sans préemption). Dans cette région l'ordonnanceur ne peut pas intervenir et les autres threads ou processus restent dans un état passif.

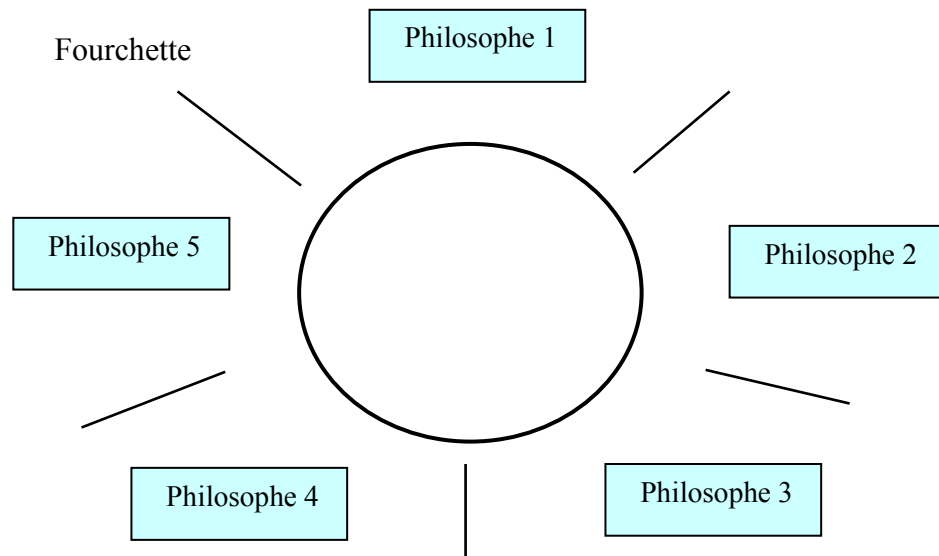


Figure I. 5: Problème de cinq philosophes

I.3.3. 2. Mécanisme de communication

Dans la majorité des OSs temps réel, les threads ou/et les processus coopèrent souvent pour traiter un même problème. Ces processus s'exécutent en parallèle sur un même ordinateur ou bien sur des ordinateurs différents. Ils doivent alors s'échanger des informations.

Il existe plusieurs moyens de communication interprocessus. Nous pouvons citer, entre autres, les messages et les tampons partagés.

a- Les messages :

Les messages forment un mode de communication privilégié entre les threads et/ou les processus. Les communications par messages se font à travers deux opérations fondamentales : envoie (message) et reçois (message). Les opérations d'envoi et de réception peuvent être soit directes entre les processus, soit indirectes par l'intermédiaire d'une boîte aux lettres. Les messages sont de tailles variables ou fixes.

b- Les tampons partagés :

On peut concevoir des applications qui communiquent à travers un segment de mémoire partagée. Le principe est le même que pour l'échange d'informations entre deux processus par les messages. Dans le cas d'une zone de mémoire partagée, on devra déclarer une zone commune par une fonction spécifique, car la zone mémoire d'un processus est protégée.

I.3.4. Méthodologie de partitionnement de l'application en threads

On entend toujours le terme « partitionnement », mais son utilisation est trop vaste, par exemple, si on parle de co-design on parle de partitionnement hardware/software qui consiste à trouver le meilleur compromis logiciel/matériel pour construire un système de haute performance.

Dans notre projet, le terme « partitionnement » est une technique utilisée pour organiser l'exécution d'un programme et le rendre plus efficace et plus lisible, elle consiste à diviser l'application en un nombre des threads ou chacun d'eux responsable d'une tâche.

I.3.4.1. Nécessité d'utilisation des threads

Citons quelques raisons pour lesquelles nous employons des threads en concevant les systèmes d'exploitation :

- Puisque les threads peuvent partager des données communes, ils n'ont pas besoin d'employer la communication interprocessus.
- Par la nature des choses, les threads peuvent prendre avantages pour un système multiprocesseur.
- Les threads sont moins cher dans le sens suivant :
 - Ils ont besoin seulement d'une pile et le stockage dans les registres, donc les threads sont plus faciles à créer.
 - Les threads utilisent des ressources minimales dans un operating system avec lequel on travaille. Donc ils n'ont pas besoin de nouvelle espace mémoire, variables globales, code de programme ou les ressources de système d'exploitation.
 - Le contexte de commutation est plus rapide avec la notion des threads puisque il suffit seulement de sauver et/ou reconstituer le PC, SP et les registres.

I.3.4.2. Méthodologie de partitionnement

Dans la littérature la notion de partitionnement reste encore un thème floue, puisque le terme partitionnement n'est pas trop utilisé dans les applications standards, mais il est fréquemment utilisé lors de la conception des systèmes d'exploitation.

A ce niveau, nous allons introduire, en se basant sur l'exemple d'un codeur vidéo H.264, telque l'illustre la figure I.6, quelques critères de partitionnement ainsi que les paramètres d'évaluation de performance.

Critères de partitionnement

- **Selon la fonctionnalité** : Ce critère consiste à regrouper les blocs qui conduisent à la réalisation d'une même tâche. Par exemple, notre application regroupe six blocs ou chacun d'eux est occupé à la réalisation d'une tâche (figure I.6).

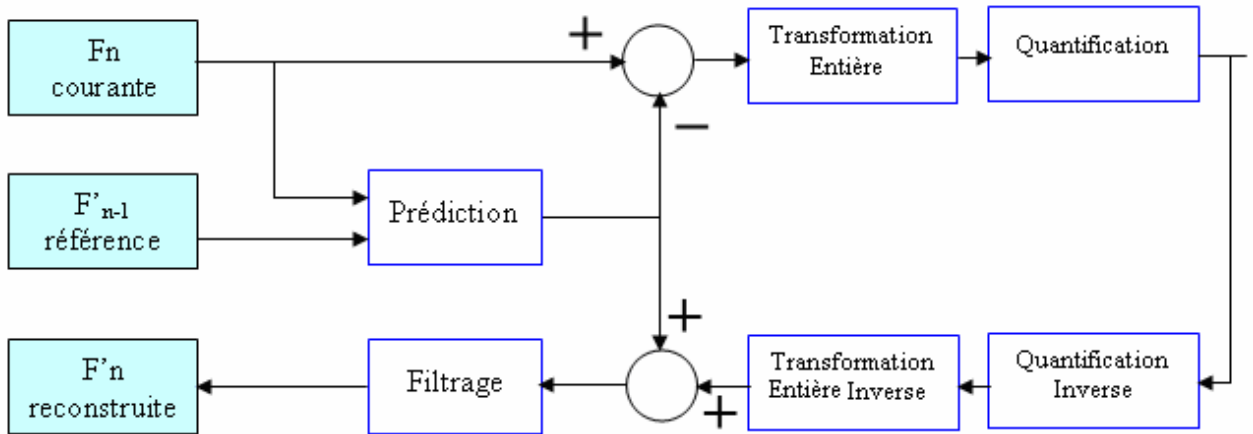


Figure I. 6 : Schéma d'un codeur vidéo H.264

En fait le bloc prédiction est divisé en deux sous blocs (prédiction intra et inter), mais les deux sous blocs consistent à réaliser la tâche de prédiction. Donc il sera mieux de regrouper le deux sous blocs pour construire un seul thread qu'on appelle « prédiction ». On aura donc cinq autres threads (Transformation entière, Quantification, Quantification Inverse, Transformation entière Inverse et Filtrage).

Si on prend la fonctionnalité à une échelle plus grande, on peut partitionner l'application en deux threads seulement. Le premier thread consiste à réaliser le codage (sens direct) et la deuxième est chargée de réaliser la fonction de reconstruction (sens inverse).

- **Selon la nature des données (Inputs)** : Ce critère consiste à regrouper les blocs qui ont les mêmes caractéristiques des Inputs (type, taille, etc.) même s'ils ont des fonctions différentes. Donc à travers ce critère, nous avons deux architecture cibles, la première consiste à regrouper les blocs de Transformation entière, Quantification, Quantification Inverse et Transformation entière Inverse en un seul thread appelé « Transformation&Quantification » et conservant les deux autres threads (Prédiction et Filtrage). La deuxième architecture fusionne les threads de « Filtrage » et celui de « Transformation&Quantificati-

on », pour construire un nouveau thread de « Transformation-Quantification&Filtrage » noté souvent « TQF ».

Chaque architecture possède des avantages et des inconvénients, mais l'architecture optimale doit respecter les deux critères précédents pour augmenter les performances suivantes :

- Rapidité d'exécution et Taux d'utilisation de processeur : diminuer le temps de traitement en augmentant le taux d'utilisation de processeur (idéale 100%) et éviter la famine du processeur en diminuant l'utilisation des ressources communes qui sont la cause d'interblocage.
- Optimisation des ressources : les ressources sont la mémoire, les registres et les périphériques, sachant qu'un thread demande moins des ressources qu'un processus.
- Maintenance et test du code : partitionnement de l'application en threads facilite le développement et le test du code et rendre la mise à jours des différents modules plus pratique.
- Coût d'implémentation : le coût d'implémentation dépend de nombre des heures nécessaires pour le développement de tous les modules, avec cette technique de partitionnement on peut diminuer le temps de réalisation de l'application car elle permet de diminuer le temps de développement du code et facilite le test.

I.3.5. Ordonnancement des tâches

Les systèmes temps réel sont conçus pour interagir avec l'environnement extérieur en garantissant le respect des contraintes pour que les réactions ou la capture des données arrivent à bon escient. Le non respect de ces contraintes par les tâches, considéré comme une faute grave du système, résulte le plus souvent de conflits survenus entre les tâches au moment de leur exécution. Ces conflits sont de deux ordres [6], ils portent sur les processeurs et les ressources :

- Conflits sur les processeurs : Plusieurs tâches peuvent réclamer simultanément le processeur afin de respecter leur contraintes. Ces conflits peuvent être résolus par des mécanismes d'ordonnancement gérant l'accès au processeur. Les ordonnanceurs qui s'en chargent n'ont qu'une vision très générale des tâches car ils ne considèrent pas les traitements qu'elles effectuent mais utilisent des paramètres définis à l'avance sur les tâches telsque le temps d'exécution, l'échéance. Ordonnancer des tâches sur un processeur consiste à déterminer une séquence d'exécution des tâches sur le processeur qui garantisse leurs contraintes.

• Conflits sur les autres ressources : Ceux-ci surviennent lorsqu'une tâche demande l'accès à une ressource ou donnée non partageable (unité d'entrée/sortie, fichiers, disques...). Des mécanismes de verrouillage peuvent être employés pour résoudre ces conflits. Toutefois, des mécanismes d'ordonnancement de ces ressources avec des considérations de temps doivent également être utilisés, afin d'éviter que pour des besoins d'intégrité de ressources, certaines tâches tardent à y avoir accès et voient leurs contraintes temporelles violées.

Un mécanisme d'ordonnancement temps réel permet de mettre en œuvre l'exécution d'un ensemble de tâches sur un ensemble de ressources rattachées à un processeur. Dans un modèle dynamique, ceci s'accompagne d'une gestion dans le temps où le mécanisme d'ordonnancement a pour charge de suivre cette exécution et de la modifier en fonction des précisions obtenues sur certains paramètres et des nouvelles tâches créées.

I.3.5.1. Mécanisme d'ordonnancement

L'algorithme utilisé pour ordonnancer l'ensemble des tâches peut être de plusieurs sortes [2], [7]:

Préemptif : on permet aux tâches d'être interrompues pendant leur exécution. Le contexte d'exécution (la valeur des différents registres du processeur, le compteur programme, ...) doit alors être sauvegarder en mémoire puis restauré lors de la remise en route de la tâche en question. Ceci introduit bien évidemment un temps de latence supplémentaire appelé latence de dispatch ainsi que des complications pour la réalisation pratique du système d'exploitation.

Temps-réel : lorsque l'on travaille en temps réel rigide, on veut être sur qu'une tâche sera terminée avant un certain temps. On introduit alors la notion d'échéance : chaque tâche t_i est modélisé par trois paramètres qui sont :

- La date de délivrance D_i qui est la date de mise en service ; pas forcément la date ou la tâche va effectivement commencer à s'exécuter mais la date à partir de laquelle l'exécution est permise.
- Le temps d'exécution X_i qui est le temps que mettrait la tâche pour s'exécuter si elle était seule sur le processeur.
- L'échéance E_i qui est la date avant laquelle on veut que le résultat soit fournie.

On notera désormais (D_i, X_i, E_i) une tâche ayant ces paramètres, comme l'illustre la figure I.7.

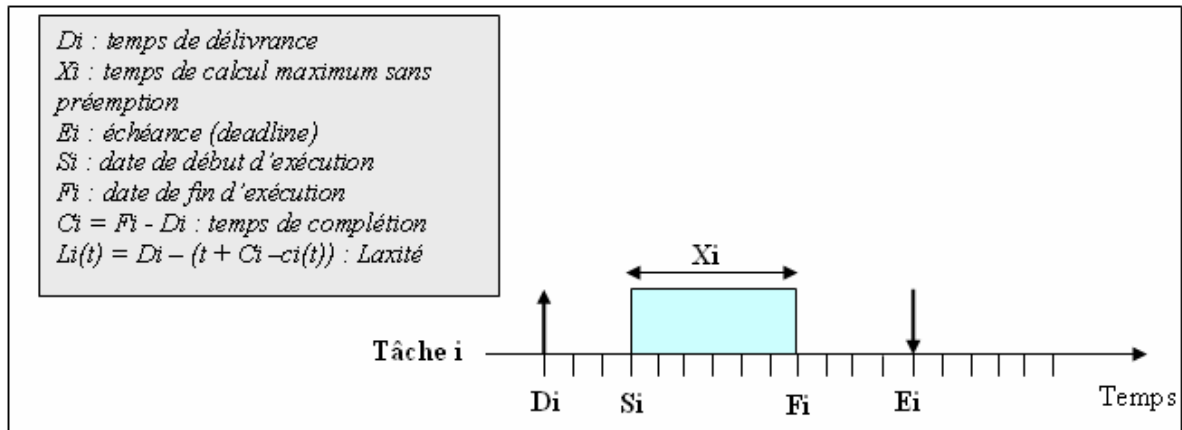


Figure I. 7 : Les paramètres temporels d'une tâche

Un algorithme d'ordonnancement Temps-réel tente, pour un ensemble de n tâches $T = \{t_i / i \in [0..n-1]\}$, de respecter toutes les échéances, c'est-à-dire qu'on veut que $\forall i < n, D_i + C_i \leq E_i$ avec C_i le temps de complétion qui n'est généralement pas égale à X_i ; en effet le temps de complétion est la différence entre la date de délivrance et la date de fin d'exécution et dépend donc des autres tâches qui peuvent éventuellement préempter ou différer l'exécution.

A priorité dynamique : l'ordonnanceur fonctionne couramment en assignant des priorités. Lorsque le choix de la tâche à exécuter doit être fait, il suffit de prendre la tâche de priorité la plus forte. La différence entre un algorithme à priorité statique et un algorithme à priorité dynamique est que le premier ne modifie pas les priorités des tâches. Si, à un instant donné, une tâche a une priorité plus élevée qu'une autre, alors cet ordre sera toujours respecté. Dans le cas des algorithmes à priorités dynamiques par contre, les priorités sont recalculées à chaque fois qu'un choix doit être fait. De ce fait, si une tâche passe avant une autre à un moment précis, cela n'implique pas que ce ne sera pas le contraire plus tard lorsque le contexte aura changé.

I.3.5.2. Algorithmes traditionnels

Ce sont ceux qui sont habituellement implémentés sur les machines:

FIFO (First In First Out) : les tâches candidates à l'exécution sont exécutées dans leur ordre d'arrivée. Ceci n'est pas vraiment un algorithme multitâche puisque la tâche en cours doit se finir entièrement avant qu'une autre finisse mais cet aspect peut tout de même être simulé si les tâches

se cèdent mutuellement la priorité. C'est par exemple le mode opératoire des systèmes d'exploitation coopératifs comme Windows 98.

Round Robin : c'est un algorithme préemptif ; chaque nouvelle tâche est placée dans une file d'attente circulaire (d'où le nom qui veut dire tourniquet) et on alloue à chacune d'elles un quantum de temps. A la fin de ce quantum, la tâche est automatiquement préemptée et on passe à la suivante. Chaque tâche obtient ainsi la même fraction de processeur et si le temps alloué est suffisamment court on a l'illusion de travailler avec plusieurs processeurs en parallèle.

SJF (Shortest Job First) : SJF est un algorithme non préemptif. Il consiste à exécuter les tâches dans l'ordre de leurs temps d'exécution. SJF est utilisé pour les applications où le temps d'exécution de chaque tâche est connu de l'avance.

RMS (Rate Monotonic Scheduling) : c'est un algorithme à priorités statiques. Le choix du nom RMS vient du fait que la fonction de distribution des priorités est monotone (Monotonic) par rapport aux fréquences (Rate) des tâches : plus la fréquence est élevée et donc plus la période est courte et plus la priorité est élevée. Si deux tâches ont la même période alors le choix est arbitraire mais doit rester le même durant toute la durée de fonctionnement de l'appareil. Cet algorithme a beaucoup de succès auprès des concepteurs de systèmes temps-réel car il est très facile à mettre en place.

DM (Deadline Monotonic) : c'est un algorithme d'ordonnancement préemptif à priorité fixe, cet algorithme impose que la priorité d'une tâche est inversement proportionnelle à son échéance relative (conflit résolu arbitrairement). DM est optimale dans la classe des algorithmes à priorités fixes pour des tâches périodiques indépendantes à échéances inférieure à la période.

EDF (Earliest Deadline First) : c'est le cas typique d'un algorithme préemptif à priorité dynamique. Il a été décrit pour la première fois par Liu et Layland mais il a connu moins de succès. Cet algorithme donne la priorité élevée pour la tâche (parmi les tâches prêtes) dont l'échéance absolue est la plus proche, il est optimal dans la classe des algorithmes préemptifs pour des configurations de tâches périodiques indépendantes avec échéances inférieure à la période.

LLF (Least Laxity First) : c'est un algorithme préemptif à priorité dynamique, il donne la priorité la plus élevée à la tâche (parmi les tâches prêtes) qui possède la laxité $L_i(t)$ la plus petite. Cette politique est la meilleure du point de vue de la faisabilité et permet le taux d'utilisation processeur le plus élevé [8].

I.4. Cas d'étude : VDK (Visual DSP Kernel)

VDK (Visual DSP Kernel) est une librairie intégrée dans VisualDSP++, elle est supportée par les familles de processeurs suivantes ADSP-219x, ADSP-21xxx, ADSP-TSxxx et Blackfin.

VDK optimise l'utilisation du potentiel du processeur, permet à un seul DSP d'effectuer l'exécution de plusieurs tâches, offre le mode instrumenté ; historique d'événements et offre une couche d'abstraction du matériel permettant la programmation haut niveau. En VDK, les threads encapsulent les algorithmes et leurs données associées et les types de threads ne sont pas utilisés directement que s'ils étaient instanciers. Donc VDK est un système d'exploitation multi-threads et préemptif offrant le traitement parallèle des données [9], [10].

Ordonnement : L'ordonnanceur est l'organe responsable pour décider si un thread devrait être arrêté ou remis en marche, il alloue le processeur au thread ayant la priorité la plus haute sachant qu'il est en état ready (les ressources qu'il demande sont libres). Une référence pour chaque thread est stockée dans une file d'attente nommée ready queue.

Priorité : En VDK, chaque thread possède une priorité limitée à 15 ou 13 niveaux selon l'architecture, cette priorité peut être modifiée dynamiquement. Le nombre de threads ayant la même priorité est limité seulement par la mémoire du système. L'affectation des priorités est une tâche assez difficile dans un système temps réel, dans les systèmes temps réel critique (contrôle d'un centre nucléaire) le dépassement d'une échéance d'une tâche peut causer une catastrophe.

Préemption : Un thread s'exécute jusqu'à ce qu'il demande une ressource, ou bien il existe des autres threads prêts possédant des priorités plus élevées, elle continue l'exécution s'il est le plus prioritaire et toutes les ressources demandées sont valides. Si les signaux ne sont pas valides, il sera bloqué et retiré de ready queue.

Un thread peut être interrompu et le Kernel cède la main au « hardware interrupt controller », une fois l'interruption traitée le thread le plus prioritaire dans le ready queue reprend l'exécution.

Régions protégées : Le Kernel fournit deux niveaux de protection pour les codes qui doivent s'exécuter d'une façon séquentielle, le première région dite « Unscheduled region » dans cette région du code l'ordonnanceur du VDK est suspendu (accès à des variables globales). Ce type de région est utilisé dans des applications où l'exécution d'une tâche ne doit pas être interrompu, par exemple thread qui contrôle les exceptions. La deuxième région « Critical region » est très

dangereuse puisque dans cette région du code l'ordonnanceur et les interruptions sont suspendus, ceci peut causer une latence à l'interruption.

I.4.1. Le thread

Comme les autres systèmes d'exploitation, chaque thread a ses propres paramètres (pile, état, etc.) et fonctionne indépendamment des autres et peut communiquer avec eux à travers plusieurs mécanismes de communication (messages, tampons partagés...).

Paramètres d'un thread : Quand un thread est créé, le système alloue un espace dans le tas « heap » pour y stocker ses paramètres requis par le Kernel et spécifiés par l'utilisateur. Chaque thread dispose de sa propre pile. La taille de la pile est laissée sous la responsabilité de l'utilisateur. Une pile sous dimensionnée ne génère pas d'exceptions et cause des difficultés de débbugage. Chaque thread possède une priorité par défaut, elle peut être changée dynamiquement par les APIs SetPriority() et ResetPriority().

Les fonctions requises d'un thread : Chaque thread requise les cinq fonctions suivantes :

- CreateThread() : c'est la fonction responsable de la création d'un thread. Elle fait appel à la fonction Initfunction().
- Initfunction() : c'est la fonction responsable de l'allocation des ressources systèmes nécessaires pour le thread.
- Runfunction() : c'est la fonction équivalente au main().
- Errorfunction() : elle est appelée par le Kernel en cas d'erreur commise par le thread.
- Destroyfunction() : c'est la fonction responsable de la destruction d'un thread et la libération des ressources utilisées. La destruction peut ne pas être immédiate elle est effectuée par le Idle thread.

I.4.2. Les signaux

Les signaux (sémaphores, messages, Events et Event bits et Devices Flags) est un mécanisme offert par la plupart des OSs utilisés pour:

- Le contrôle d'accès aux ressources partagées.
- La synchronisation.
- La communication entres les threads.
- Le compte des occurrences système.

Un thread attend jusqu'à ce que le signal soit valide ou le timeout soit écoulé (optionnel).

I.4.2.1. Les sémaphores

En VDK, un sémaphore est caractérisé par un count initiale (*SI*) et un compte max (*SM*). Ces paramètres sont définis au moment de la création du sémaphore. Si plusieurs threads demandent un sémaphore et que ce dernier est supérieur à zéro, le thread le plus prioritaire continue l'exécution après la décrémentation du sémaphore, si le sémaphore est égal à zéro, le thread qui le demande sera bloqué et si le temps spécifié écoulé, le thread continue son exécution dans sa fonction d'erreur.

Les sémaphores sont des ressources globales accessibles par tout les threads et si un thread libère un sémaphore ce dernier sera incrémenté. Un sémaphore peut être incrémenté périodiquement par le Kernel chaque *n* ticks.

I.4.2.2. Les messages

En VDK, chaque message est déclaré comme une variable locale identifiée par, un MessageID, un type des données, la taille des données à transférer et un pointeur vers un buffer de données sous forme de « Payload ». La communication entre les threads se fait à travers quinze canaux logiques. Quand un message attend sur un canal, la seule opération possible sur celui-ci est de faire un «Pend».

Un thread crée un message puis il l'envoie, il continue son exécution sauf si son message enclenche un thread de plus haute priorité. Un thread destination peut écouter sur un ou plusieurs canaux. Les messages se mettent en file d'attente jusqu'à ce qu'ils soient retirés par le thread « destinataire ». Si un thread attend un Message et que le Timeout s'écoule, le thread continue son exécution dans son ErrorFunction(). Si des messages restent en attente sur un canal alors que le destinataire a été détruit, le système détruit ces messages sans pour autant libérer leurs Payload.

I.4.2.3. Les Events et les Event bits

Events et Event bit sont des signaux utilisés pour régler l'exécution des threads basée sur l'état du système, un event bit est utilisé pour signaler qu'un certain élément du système est dans un état indiqué. Un event est opération booléenne effectuée sur l'état de tout les event bits. Quand la combinaison booléenne des event bits est telle que l'event soit juste, tous les threads qui pendent

sur cet event sont déplacés dans la file d'attente du processeur (Ready Queue) et l'event reste vrai. Tout threads demandant un event non valide seront bloqués et éliminer de la file d'attente du processeur (Ready Queue).

Le nombre des Events et d'event bits est limité par la taille du mot « Word size » du processeur. Par exemple, dans l'architecture 16 bits, il y a 15 Events et event bits, mais dans l'architecture 32 bits on trouve 31 Events et event bits.

I.4.2.4. Les Devices Flags

Les flags (drapeaux) sont des signaux utilisés dans la majorité des systèmes d'exploitation pour contrôler les interruptions afin de les servir. Les Devices Flags sont par défaut "false" et sont utilisés spécialement dans les Devices Drivers pour la synchronisation entre soft et hard. Un thread qui demande un Device Flag se bloque automatiquement. Si le Device Flag est libéré tous les threads qui le demandent se mettent en ready queue. La routine PendDeviceFlag() doit être appelée à partir d'une zone critique.

I.5. Conclusion

Dans ce chapitre, nous avons introduit les systèmes embarqués en spécifiant les différents éléments qui interviennent dans la phase de conception. Ensuite, nous avons illustré quelques critères pour le choix d'un bon RTOS (Real Time Operating System) dont les performances attendues devraient respecter tous contraintes imposées par l'environnement extérieur. Dans la troisième partie, nous avons fait une étude sur le système d'exploitation VDK intégré dans l'environnement ADSP++.

Dans la suite de notre travail nous allons nous intéresser à la compression vidéo en spécifiant les différents éléments qui interviennent dans ce processus.

Chapitre II
Codeur Vidéo H.264

II.1. Introduction

La vidéo Numérique (Annexe B) est adoptée comme un choix qui a de plus en plus de prolifération d'applications s'étendant de la téléphonie visuelle à la vidéoconférence et de DVD à la TV numérique. L'adoption de la vidéo numérique dans plusieurs applications a été accompagnée par le développement de plusieurs standards de codage vidéo. Ces normes fournissent les moyens requis pour réaliser l'interopérabilité entre les systèmes conçus par différents fabricants pour n'importe quelle application donnée, par conséquent facilitant la croissance du marché visuel.

L'Union Internationale de Télécommunications, standardisation du secteur des Télécommunications (UIT-T) est l'un de deux organismes qui développent les normes de codage vidéo, l'autre c'est l'ISO/IEC (International Standards Organisation / International Electrotechnical Commission). Les travaux techniques de l'ISO/IEC sont menés au sein du groupe MPEG (Motion Picture Experts Group) pour définir les standards MPEG-1, MPEG-2 et MPEG-4 pour des applications aussi variées que la télévision ou le multimédia. En parallèle des activités de MPEG, le groupe vidéo de l'UIT-T s'intéresse principalement à la définition de recommandations techniques destinées aux applications de visiophonie et de visioconférence (normes H.261 et H.263 et H264).

Actuellement, ces deux organismes travaillent à l'élaboration d'une norme commune H.264 / MPEG-4 « AVC (Advanced Video Coding) » dont les performances attendues devraient permettre de réduire de moitié le débit de transmission ou de stockage pour une qualité visuelle équivalente aux normes précédentes.

L'objet de ce chapitre est de présenter brièvement un codeur vidéo H.264. Une première partie sera consacrée pour les spécificités de la norme de compression actuelle. Une deuxième partie décrit la solution existante implémentée sur la plateforme Panview2.1 d'ANALOG DEVICES Inc.

II.2. Compression vidéo H.264

II.2.1. Historique de compression vidéo

Les standards de compression vidéo ont joué un rôle important pour le développement des technologies numériques. En effet le choix de la technologie, en termes d'efficacité et d'interopérabilité permet d'assurer un large déploiement des services et produits associés à un coût optimal pour les constructeurs et les utilisateurs. La figure II.1 résume l'évolution des principales normes et standards normalisés par l'UIT-T et par l'ISO/IEC.

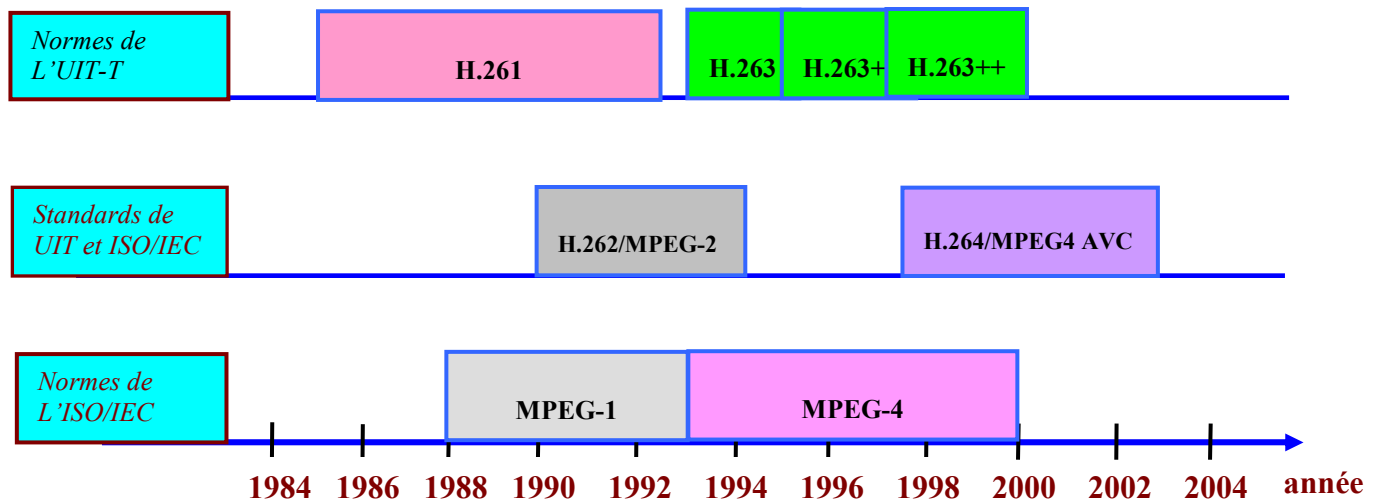


Figure II. 1 : Les principaux standards normalisés par l'UIT-T et par l'ISO/IEC

II.2.1.1. Les normes de L'UIT-T

La première norme vidéo est la norme H.261, adoptée en 1993. Elle est destinée pour les applications de visiophonie pour le réseau RNIS (Réseau Numérique à Intégration de Service) à des débits multiples de 64kbit/s [11]. Les formats d'image traités sont le QCIF (Quarter Common Intermediate Format) et le CIF (Common Intermediate Format) (Annexe B).

En 1995, une première version de la norme H.263 apparaît. Elle vise les applications de visiophonie et de visioconférence sur RTC et RNIS [12]. Cette norme repose sur les principes de la norme H.261. Les formats d'images sont des multiples et sous-multiples du CIF. Un codeur H.263 peut effectuer des compensations en mouvement avec une précision au demi-pixel, ceci améliore grandement la qualité de la vidéo en réduisant fortement les effets dits "moustiques" [12].

La deuxième version de la recommandation H.263 adopté en 1998 est appelée H.263+ [13]. Elle met en oeuvre douze options supplémentaires permettant d'améliorer fortement la qualité et la robustesse aux erreurs. La dernière version de H.263 publiée en 2000, appelée H.263++ [14], ajoute trois autres options et une spécification à la version antérieure, elle prend mieux en compte la transmission vidéo temps réel sur des réseaux à qualité de service non garantie (IP et mobiles).

II.2.1.2. Les normes de L'ISO/IEC

Première norme de compression vidéo développée par L'ISO/IEC et publiée en 1993, MPEG-1 supporte principalement le codage vidéo allant jusqu'à 1.5 Mbits/s [15], donnant une qualité similaire au VHS (Video Home System). Cette norme a été construite sur la base de H.261 dont elle reprend ses principes en les améliorant : compensation de mouvement au $\frac{1}{2}$ pixel, images de types I (Intra), P (Prédite) ou B (Bidirectionnelle), quantification optimisée par l'utilisation de matrices de quantification, prédiction spatiale du coefficient DC (Direct Current) pour les images I. MPEG-1 n'est plus guère utilisée aujourd'hui si ce n'est en compression du son avec le format MP3 pour le stockage de la musique.

La norme MPEG-4, connue sous la désignation "ISO/IEC 14496-2" a été terminée en octobre 1998 et publiée comme un standard international au début de l'année 2000, elle représente une norme générique de compression destinée à la manipulation d'objets multimédia [16]. Elle permet le codage d'une grande variété de format vidéo (taille, résolution, fréquence image) mais aussi le codage d'objets vidéo de forme arbitraire. De ce fait cette norme, adresse à une large gamme d'applications audiovisuelles allant de la visioconférence à la production visuelle en passant par le streaming sur Internet.

MPEG-4 combine les outils de MPEG-2 et H.263 ainsi que de nouveaux outils lui conférant une plus grande efficacité en compression tels que des modes de scalabilité plus adaptés à une transmission sur des réseaux à débit fluctuant ainsi qu'une plus grande robustesse aux erreurs de transmission.

II.2.1.3. Les normes communes

Dans la plupart des cas, les deux organismes ont travaillé indépendamment au cours de normalisation, la première norme commune H.262/MPEG-2 développée en 1994, elle vise les applications liées à la TV numérique [17]. La norme MPEG-2 reprend les principes de MPEG-1 en

ajoutant des outils indispensables pour les applications télévisuelles. Ce standard a été adopté pour le DVB (Digital Video Broadcasting) pour les services de TV numérique par voie hertzienne terrestre (DVB-T) et satellite (DVB-S). Il est également utilisé comme format de codage du DVD (Digital Versatile Disc).

Récemment, les deux groupes ont approuvé, de nouveau, le rapprochement de leurs équipes vidéo pour la définition d'un nouveau standard de compression H.264 en mars 2003, cette décision a conduit les deux groupes à fusionner sous le nom de JVT (Joint Video Team) le 6 décembre 2001.

Le but de cette nouvelle entité est de standardiser un codeur vidéo basé sur la norme H.26L qui a commencé en 1998, elle promet d'être environ 50% plus efficace que MPEG-4, 66% plus efficace que MPEG-2 [18], [19]. Cette norme constitue une nette amélioration par rapports aux normes vidéo antérieurs sur le plan de la qualité de l'image, de la compression, de l'efficacité et de robustesse.

II.2.2. Description technique de la norme H.264

La norme actuelle H.264 n'apporte pas de rupture technologique par rapport aux autres normes de codage antérieures. En réalité, les éléments communs à toutes les normes de codage vidéo sont retenus comme l'illustre la figure II.2. Les différences se situent à petite échelle à chaque étape du codage (prédiction, transformé, quantification, etc.).

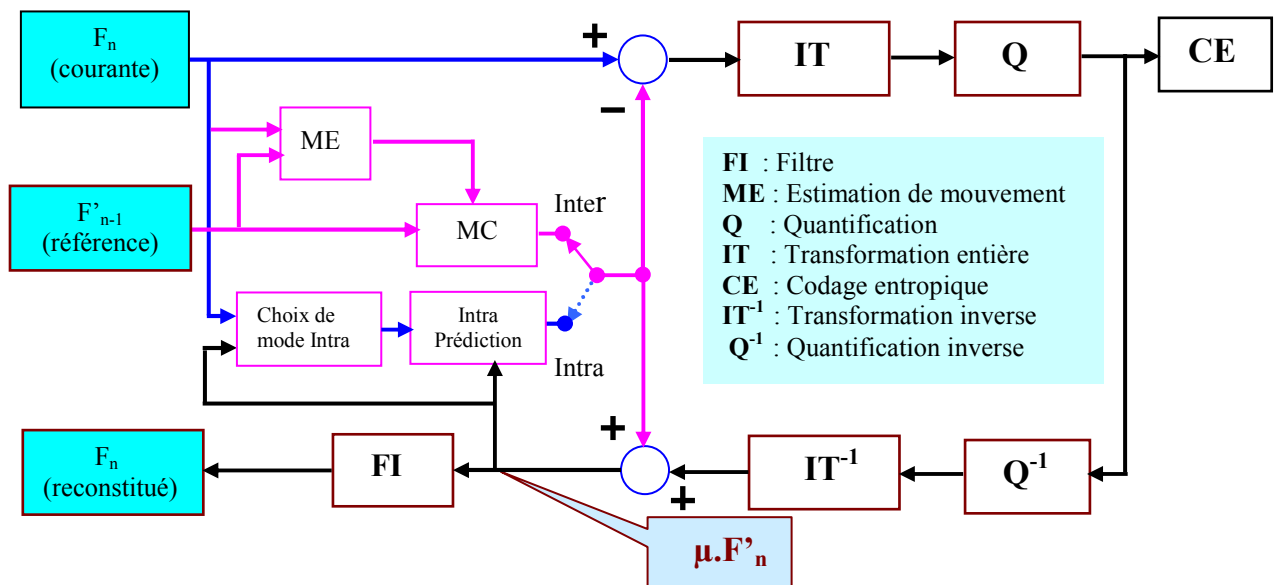


Figure II. 2 : Schéma fonctionnel d'un codeur vidéo H.264

II.2.2.1. Division en macroblocs dans l'inter-prédiction

Dans un flux de vidéo, chaque image est partitionnée en macroblocs de dimension fixe qui couvrent une zone rectangulaire de l'image de 16x16 échantillons de luminance et de 8x8 échantillons des deux composantes de chrominance (Annexe B). H.264 nous permet aussi le passage aux sous-partitions [21]. Il s'agit de blocs pouvant avoir des tailles variées comme l'illustre la figure II.3.

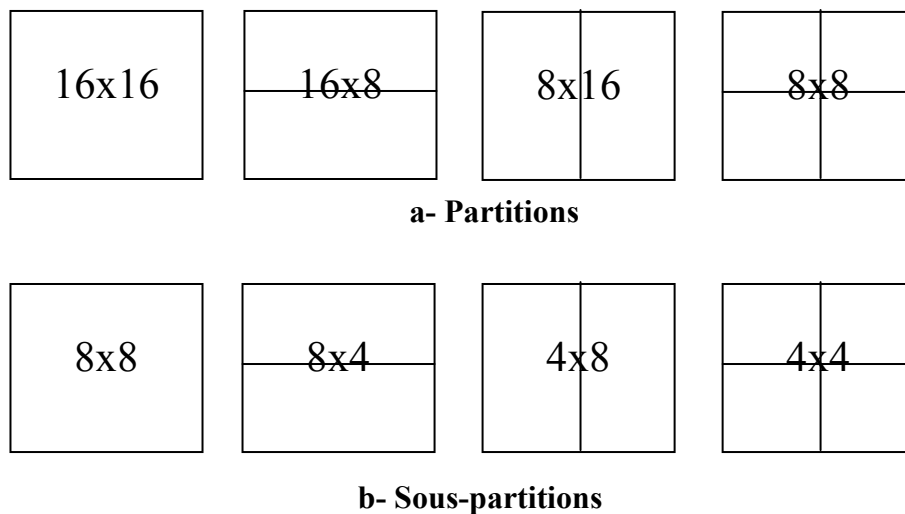


Figure II. 3: Les différentes partitions et sous-partitions dans la norme H.264

II.2.2.2. Les types d'images et structure du GOP

On distingue dans un flux H.264 plusieurs types d'images selon les modes de prédiction utilisés :

- Image I (Intra Picture) : Image codée (codage intra) indépendamment des autres, elle fournit les points d'accès dans le train binaire. Elle sert au codage des images de type P et B. Son taux de compression est moyen.
- Image P (Predictive Picture) : Cette image peut être prédite par compensation de mouvement à partir d'une image "I" ou "P". Elle sert aussi au codage des images de type "B". Le taux de compression est supérieur à celui de type "I".
- Image B (Bi-directionally predictive Picture) : Image peut être prédite soit par simple compensation de mouvement avant ou arrière à partir d'images "P" ou "I" soit par double compensation de mouvement avant et arrière.

En plus de ces trois types d'images la norme H.264 introduit deux nouveaux types (Image SP (Switching P) et Image SI (Switching I)) pour une commutation efficace entre les flux codés à différents débits.

Le codage prédictif rend les images dépendantes entre elles puisque lors du décodage, il est nécessaire d'avoir l'image précédente pour décoder l'image en cours. Cette dépendance est gênante si une image est perdue lors d'un transfert, le décodage du reste de la séquence devient impossible. C'est pourquoi le processus de compression regroupe les images en GOP (Group Of Pictures) composés de trois types d'image.

Un GOP commence généralement par une image I qui constitue l'image-clé du GOP, cette image est suivie par une suite d'images P, intercalées par des images de type B, comme le montre la figure II.4.

La fréquence des images P et la taille du GOP sont des paramètres qui influent sur le taux de compression et sur le temps de décodage. En effet, plus le GOP sera long, plus la corrélations entre les images sera importantes, mais en diminue la robustesse.

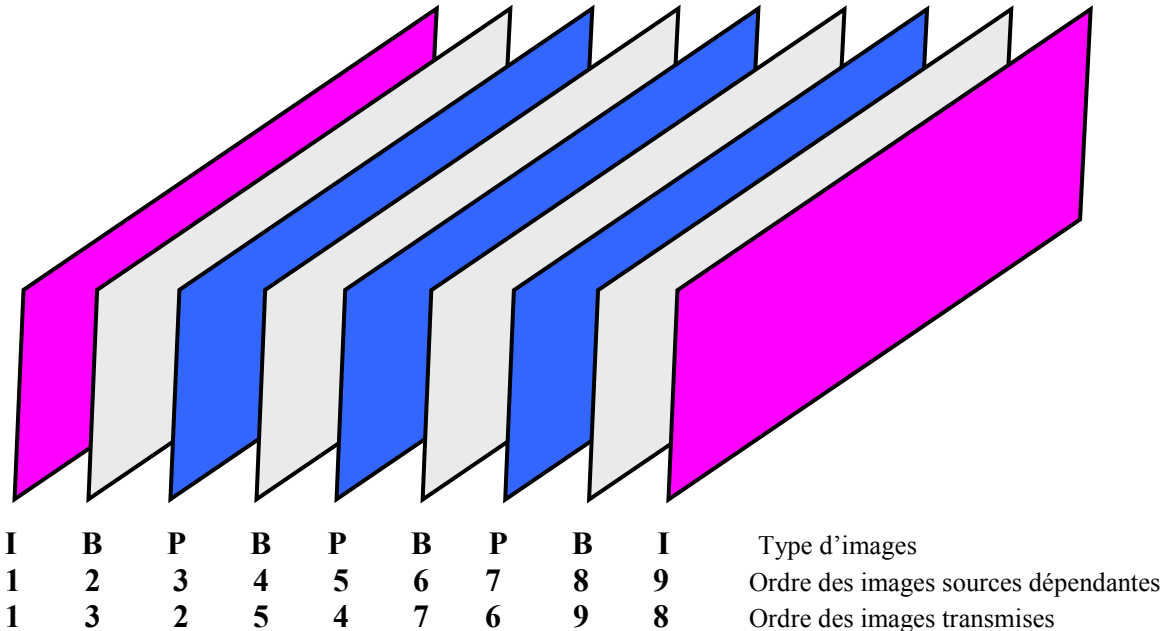


Figure II. 4 : Structure générale du GOP

On distingue deux types de GOP:

- GOP fermé : il n'existe pas des images références appartenant au GOP précédent pour limiter la propagation des erreurs en cas de perte des images.

- GOP ouvert : il fait référence au GOP précédent pour assurer la continuité des références.

II.2.2.3. Prédiction Intra

Une image codée en intra peut être construite par le décodeur sans faire recours aux autres images références. Pour le codage en intra la norme H.264 définit deux tailles des blocs (4x4 et 16x16), le codage de chaque bloc se fait en plusieurs modes [20]. Il y a neuf modes possibles de prédiction pour chaque bloc de luminance de taille 4x4, comme l'illustre dans la figure II.5, quatre modes possibles pour un bloc de luminance 16x16 comme le montre la figure II.6 et aussi quatre modes qui sont toujours appliqués à chaque bloc de chrominance 8x8.

a- Prédiction 4x4 en luminance

La prédiction d'un bloc de luminance 4x4 se réfère aux échantillons au-dessus et à gauche qui ont été précédemment codés et reconstruits. Ils sont donc disponibles dans le codeur et le décodeur pour former une référence de prédiction.

Les neuf modes de prédictions sont les suivantes :

- Mode 0 (Vertical): Les échantillons sont prédites à partir des échantillons du haut par une simple interpolation vertical.
- Mode 1 (horizontal): Les échantillons du gauche I, J, K et L sont extrapolés horizontalement.
- Mode 2 (DC): Les échantillons sont obtenus en faisant la moyenne entre la verticale et l'horizontale.
- Mode 3 (Down-Left Diagonal): Les échantillons sont interpolés à un angle de 45° entre haut-droite et bas-gauche.
- Mode 4 (Direct Diagonal): Les échantillons sont extrapolés à un angle de 45° vers le bas et vers la droite.
- Mode 5 (Vertical-Left): Extrapolation sous un angle, approximativement, de 26.6° du haut gauche au bas droit.
- Mode 6 (Horizontal-Down): Extrapolation sous un angle, approximativement de 26.6° au-dessous de l'horizontale.

- Mode 7 (Vertical-Right): Interpolation sous un angle approximativement de 26.6° à la droite de la verticale.
- Mode 8 (Horizontal-Up): Interpolation sous un angle approximativement de 26.6° au-dessus de l'horizontale.

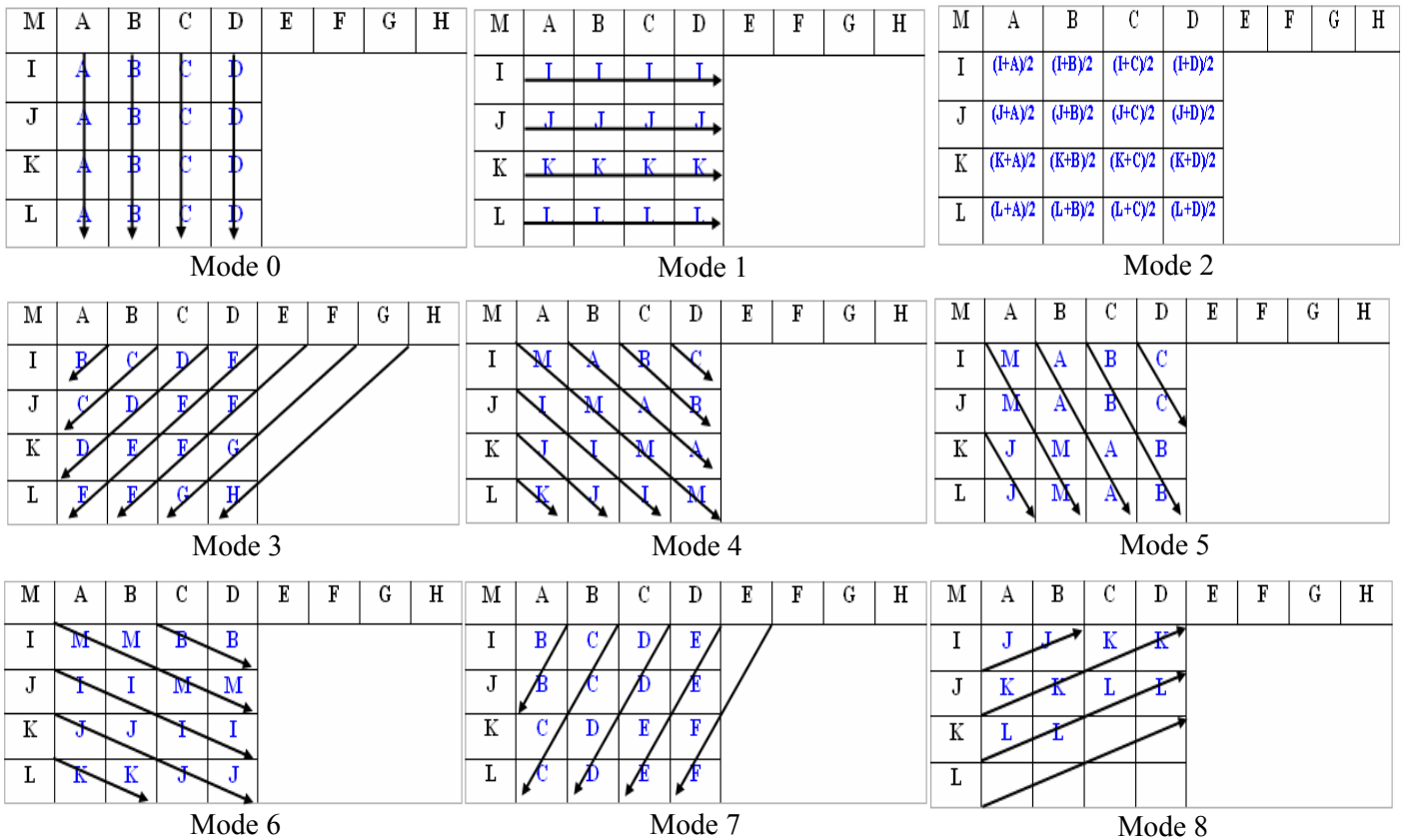


Figure II. 5 : Prédiction d'un bloc 4x4 en mode intra

b- Prédiction 16x16 en luminance

Similairement aux blocs luma de taille 4x4, un macro-bloc luma 16x16 peut être prédit selon quatre mode (0 à 3), décrites dans la figure II.6 :

- Mode 0 (vertical) : Interpolation verticale.
- Mode 1 (horizontal) : Interpolation horizontale.
- Mode 2 (DC) : les échantillons sont obtenus en faisant la moyenne entre l'horizontale et la verticale.
- Mode 3 (plane) : les échantillons situés dans la partie supérieure par rapport à la diagonale sont interpolés à un angle de 45° entre haut-droite et bas-gauche, les autres sont interpolés à un angle de 45° de bas-gauche à haut-droite.

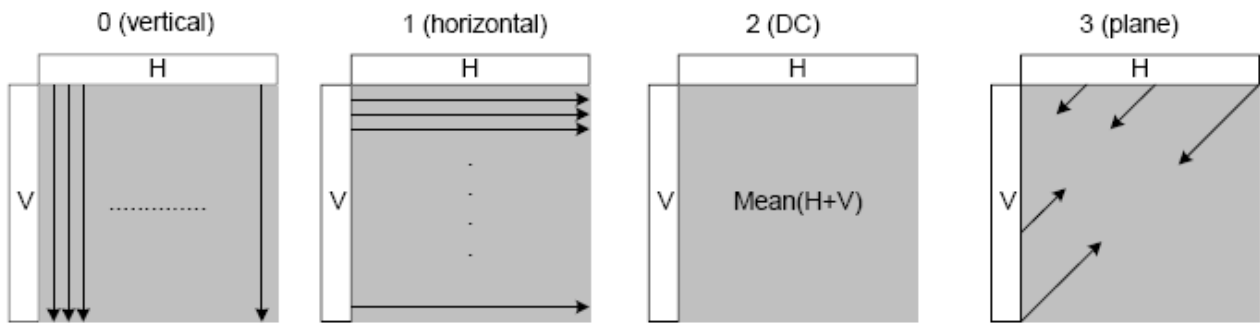


Figure II. 6: Modes d'intra-prédiction 16x16 luma

c- Mode de prédiction 8x8 en chroma

chaque composante de chroma 8x8 d'un macro-bloc est prédite à partir des échantillons au dessus et/ou à gauche qui ont précédemment codés et reconstruits. Les quatre modes de prédiction sont très similaires aux modes utilisés pour le codage luma 16x16: vertical (mode 0), horizontal (mode 1), DC (mode 2) et plane (mode 3).

II.2.2.4. Prédiction temporelle

L'inter-prédiction est basée sur l'estimation et la compensation du mouvement en profitant des redondances temporelles qui existent entre les images consécutives. L'efficacité de H.264 dérive de l'estimation du mouvement. Il garde la plupart des méthodes utilisées dans les normes antérieures mais il ajoute la flexibilité et d'autres fonctionnalités.

- a- **L'estimation de mouvement** : En règle général le mouvement, dans une séquence vidéo ne peut pas se modéliser par une simple translation sauf le cas d'un simple déplacement d'un objet indéformable. A chaque macrobloc de l'image, on associe une information de mouvement. L'opération d'estimation de mouvement permet de déterminer dans l'image de référence le macrobloc qui rassemble le plus au macrobloc à coder. Cet algorithme de recherche n'est pas normalisé et son efficacité a une influence fondamentale sur la performance du codeur. La méthode la plus utilisée est le *block-matching* comme indique la figure II.7 : le macrobloc est comparé avec les macroblocs pointés par les vecteurs testés dans la zone de recherche de l'image de référence. Le vecteur est en général déterminé avec une précision d'un demi-pixel jusqu'au $\frac{1}{4}$ pixel. La sélection est faite sur le macrobloc qui minimise la somme des valeurs absolues des différences entre les pixels du bloc courant et le bloc de référence.

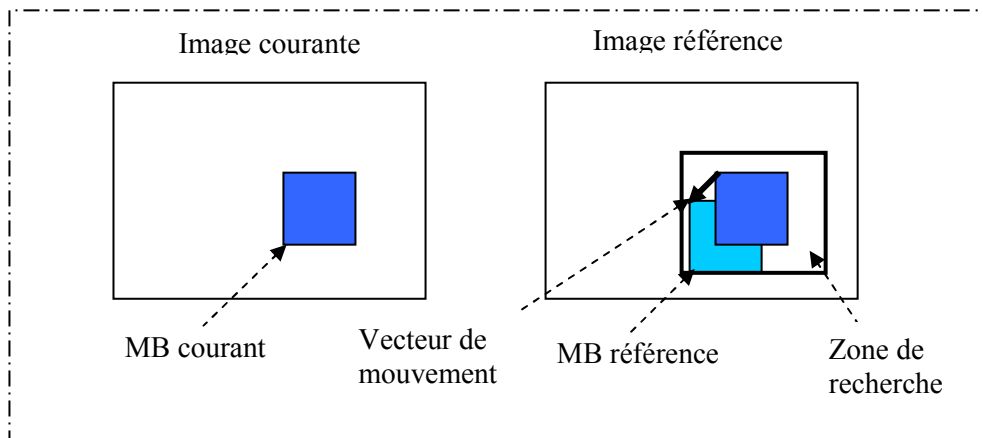


Figure II. 7 : Estimation de mouvement

b- La compensation de mouvement : C'est la formation du nouveau MB à partir de l'image référence de façon à permettre la même opération de compression dans le décodeur où seules les images décodées sont disponibles. Il diffère des normes précédentes en proposant une grande variété de formes et de tailles de blocs (16x16, 16x8, 8x4, 4x4). En effet un macrobloc de taille 16x16 peut être divisé en quatre partitions de tailles 16x16, 16x8, 8x16, 8x8 comme le montre la figure II.3-a. Si le mode 8x8 est choisit, chacune des quatre partitions 8x8 peut être partitionnée de nouveau pour obtenir des sous-partitions de tailles 8x4, 4x8 ou 4x4 comme l'indique la figure II.3-b. Cette caractéristique permet de s'adapter plus finement au contenu spatial et au mouvement des images, elle permet de diminuer plus que 16% [18] le débit binaire comparé à celui utilisant seulement les blocs 16x16.

Ainsi, le choix de type de la partition a un impact important sur les performances du codeur. En effet, les partitions de grande taille sont choisies pour les surfaces homogènes et les arrière plans, par contre les partitions de petite taille sont utilisées pour les zones qui présentent beaucoup de détails.

II.2.2.5. Transformation entière

La transformation entière opère sur des macroblocs d'erreur provenant de l'intra-prédiction ou inter-prédiction à une taille fixe 4x4 contrairement aux normes antérieurs qu'elles traitent des blocs de taille 8x8. La transformation entière a pour but de décorréliser les blocs originaux ou différentielles, de concentrer l'énergie aux fréquences basses et d'éliminer les redondances

spatiales. Par rapport à la transformée de Fourier rapide, la transformée entière présente l'avantage d'être efficace en terme de décorrélation et d'être facile à implémenter.

La transformée inverse de H.264 est définie de manière exacte (précision entière) pour éviter les divergences dues aux arrondissements.

II.2.2.6. Quantification et parcours des données

La quantification a pour but de discrétiser les amplitudes des coefficients issus de la transformée entière. La quantification la plus simple est la quantification uniforme qui consiste à diviser tous les coefficients par la même valeur. Toutefois, il est judicieux d'adapter ce quantificateur aux valeurs obtenues après la transformée. On utilise alors des tables de seuillage de quantification qui préservent l'importance des coefficients de la transformée. Il suffit de diviser les coefficients les plus énergétiques (basse fréquence) par des seuils peu élevés et les coefficients les moins énergétiques (haute fréquence) par des seuils élevés. Chaque table de seuillage est caractérisée par la donnée d'un paramètre QP (Pas de Quantification). Dans la norme H.264, cinquante deux quantificateurs sont prévus [22].

Le parcours des données quantifiées consiste à mettre les coefficients les plus énergétiques au début du train et placer les données nulles à la fin pour minimiser la taille des informations à coder, comme l'indique la figure II.8.

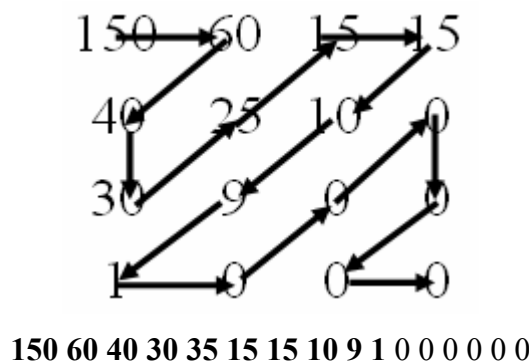


Figure II. 8 : Balayage en zigzag

II.2.2.7. Le codage entropique

Le codage entropique peut être réalisé de trois manières différentes dans H.264. Une première méthode utilise une table universelle de mot de code (UVLC -Unified Variable Length Coding). Cette table est utilisée pour coder la plupart des éléments de synchronisation comme les en-têtes. Les deux autres méthodes sont utilisées pour coder presque tous les autres éléments syntaxiques (coefficients, vecteurs mouvements). Il s'agit d'une part d'un codage VLC adaptatif au contexte (CAVLC - Context Adaptive Variable Length Coding) et d'un codage arithmétique adaptatif contextuel (CABAC - Context Adaptive Binary Arithmetic Coding) d'autre part.

Dans la norme H.264/AVC, le moteur fondamental de codage arithmétique et son estimation de probabilité connexe sont définis comme des méthodes de faible complexité, sans multiplication, qui utilisent uniquement les commutations et des vérifications de tables. Par rapport au CAVLC, le CABAC garantit en général une réduction du débit binaire de 10 à 15% [18] lors du codage de signaux TV d'une même qualité.

II.2.2.8. Le filtrage de boucle

La norme H.264/AVC définit un filtre de «déblocage» adaptatif en boucle qui améliore l'efficacité de compression et la qualité visuelle des séquences vidéo en gommant certains effets indésirables du codage tels que les effets de bloc. En même temps, le filtre réduit le débit binaire en général de 5 à 10% [23] tout en produisant la même qualité objective que la vidéo non filtrée.

II.3. Solution Codec sur DSP BF561

II.3.1. Architecture de la Plateforme Panview 2.1

La plateforme d'implémentation du codeur vidéo H.264 *Panview* est un outil d'ANALOG DEVICES utilisée pour les communications audio-visuelles. Son architecture est donnée par la figure II.9. Le noyau de cette carte est l'ADSP BF561, c'est un processeur dual-core qui opère sur 500 MHz et appartient à la famille Blackfin.

Les tâches principales effectuées par cette carte sont :

- Capture vidéo : Les données vidéo peuvent être acquises selon deux modes : soit directement à partir d'une caméra numérique, soit en utilisant une caméra analogique et les données seront numérisées selon la format 4 :2 :2 (Annexe B) à l'aide du circuit ADV71781.

- Capture audio : Le même principe que la capture vidéo, les données audio sont captées à partir d'un microphone sous forme d'un signal analogique, ensuite ils seront numérisés.
- Filtrage spatial et temporel : Exécutée par le circuit FPGA (Field Programmable Gate Array) dans le but de préfiltrer les images qui seront stockées dans la mémoire.

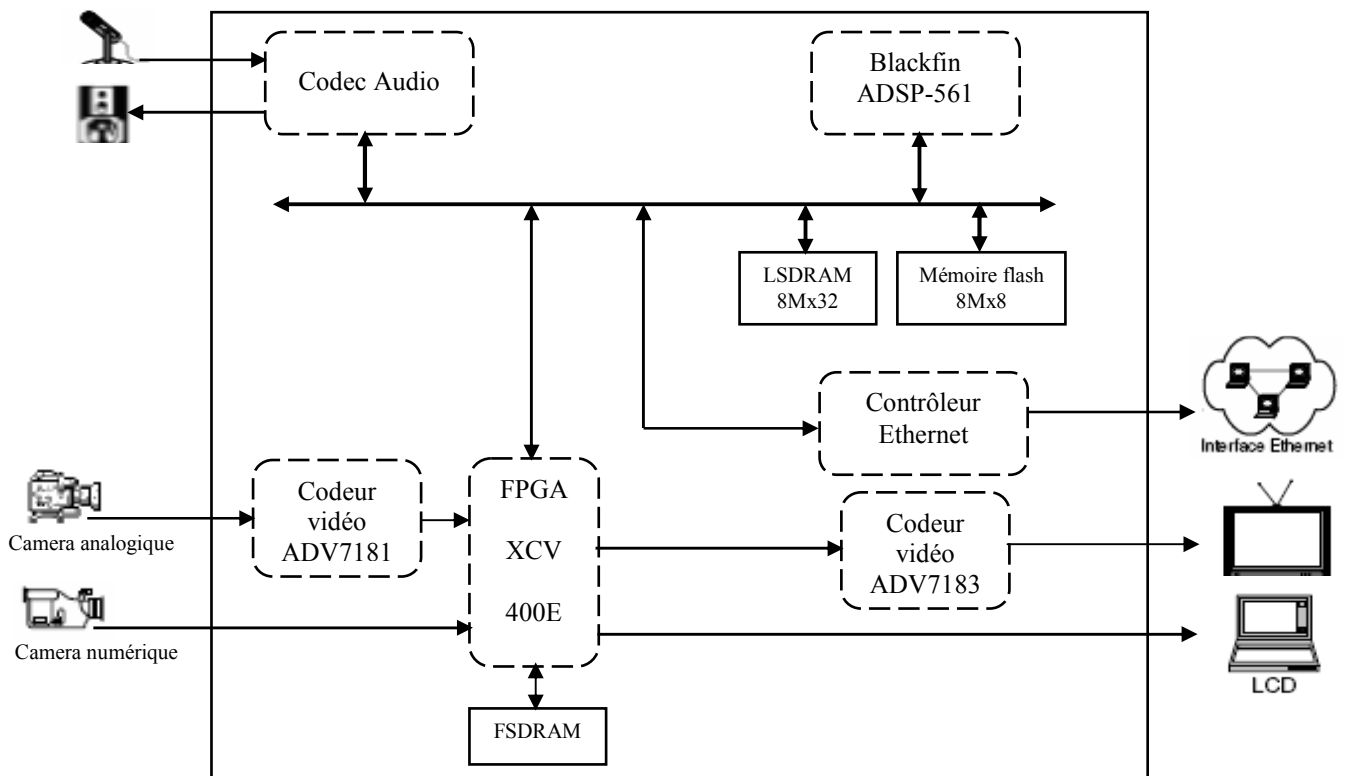


Figure II. 9 : Architecture de la plateforme vidéo

- COMpression DECompression (CODEC) : C'est la tâche principale de la carte. Le processus de compression est réalisé par le codeur H.264, le décodeur consiste à décompresser le flux des données provenant d'un autre codeur distant.
- Affichage vidéo : La carte Panview permet l'affichage suivant plusieurs standards (PAL (Phase Alternating Line), SECAM (Séquentiel Couleur A Mémoire), NTSC (National Television Standard Committee)) selon la configuration des registres du circuit de codeur vidéo ADV7183.
- Emission/Réception des données : L'échange des données avec le réseau se fait à travers une interface Ethernet (RJ 45).

- Les transferts DMA : Le contrôleur DMA prend la charge sur le DSP pour transférer les données soit entre la mémoire interne (mémoire cache) et les interfaces, soit entre les caches et le SDRAM (Synchronous Dynamic Random Access Memory).

II.3.2. Description de la solution existante

II.3.2.1. Architecture globale d'un codec vidéo

La figure II.10 rappelle l'architecture globale d'une plateforme de compression vidéo H.264, la première tâche consiste à capturer la vidéo à partir d'une camera qui est connectée à la carte. La numérisation de la vidéo ou la conversion au format CIF 4:2:2 se fait au niveau de PPI_{in} et de module ADV7181. Les algorithmes de H.264 sont conceptuellement divisés en deux couches : une première couche de codage vidéo (VCL - Video Coding Layer) qui s'occupe de la représentation efficace du contenu vidéo et une couche d'adaptation au réseau (NAL - Network Adaptation Layer) qui s'intéresse plus particulièrement à une mise en forme des données vidéo, adaptée au support de transmission, comme l'illustre la figure II.10.

La réception des données binaires et l'extraction de la charge utile se fait au niveau du décodeur pour afficher les images au format analogique NTSC. La conversion de la vidéo du format numérique CIF 4:2:2 (Annexe B) au format analogique est réalisé par le circuit ADV71783 et par le port PPI_{out} .

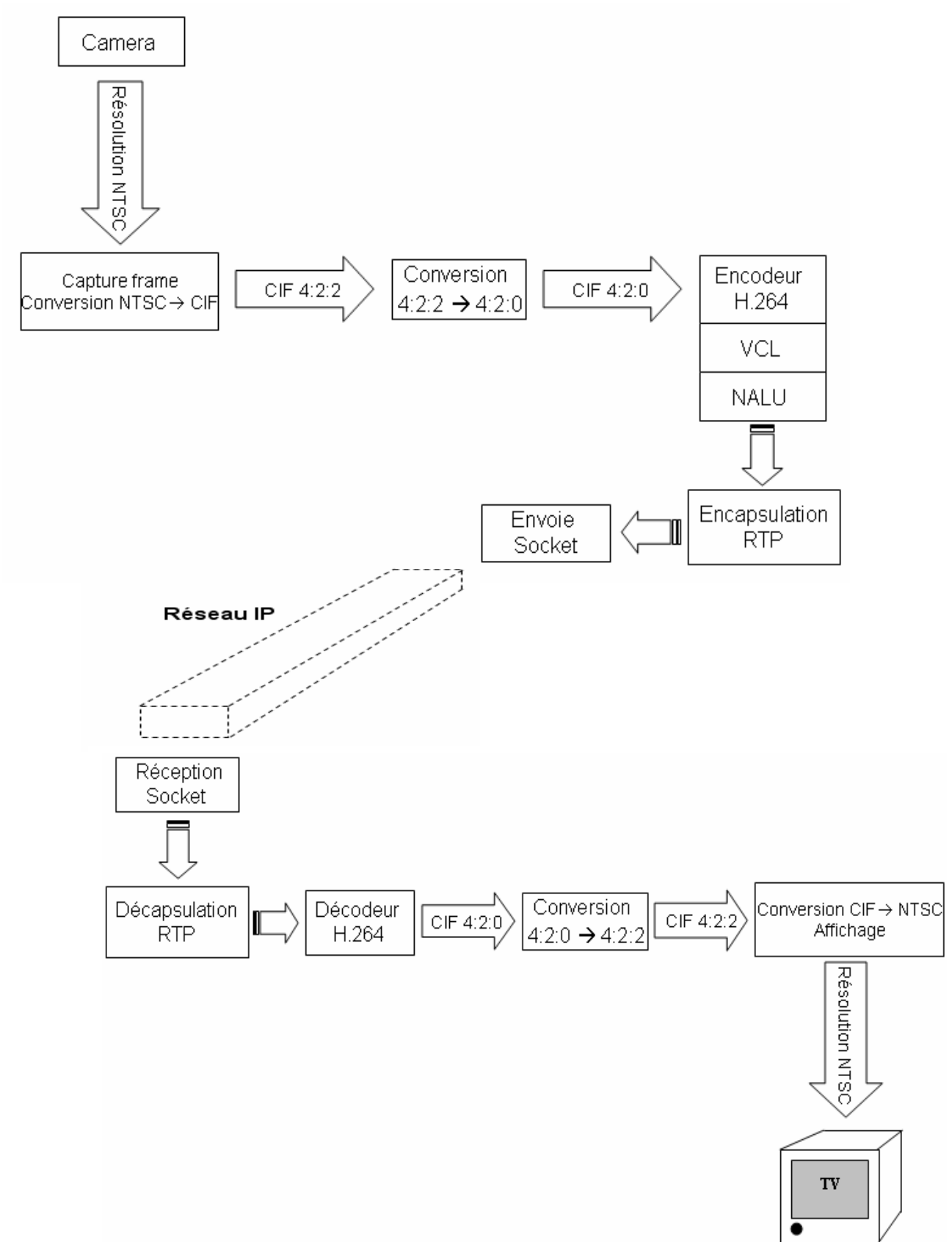


Figure II. 10: Organigramme de l'application

II.3.2.2. Acquisition de la vidéo

La procédure de capture vidéo à partir d'une caméra se compose de deux tâches principales comme l'illustre la figure II.11.

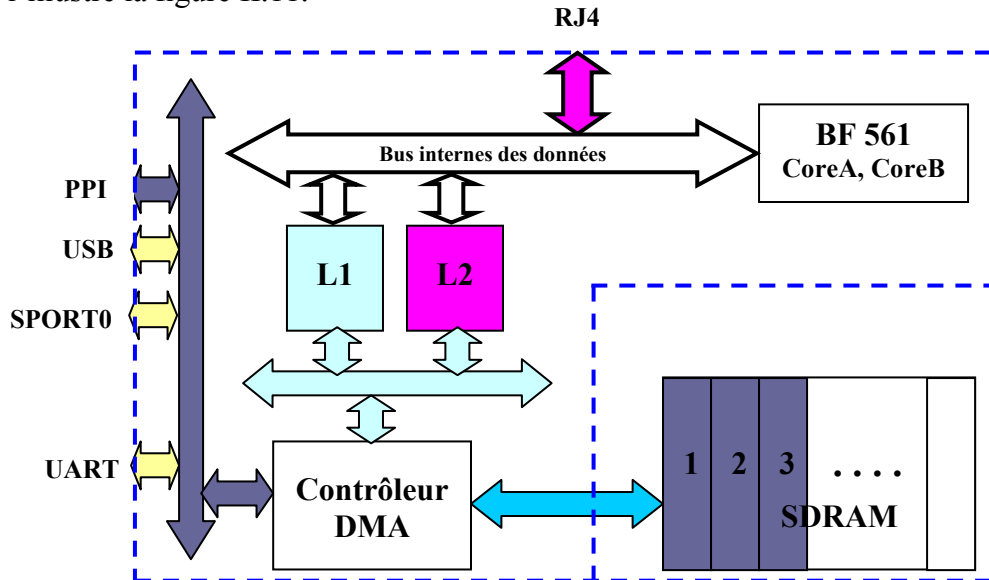


Figure II. 11 : Flux de données avec l'extérieur

- Capture vidéo : À travers l'interface PPI (Parallel Peripheral Interface) le codeur capte les données vidéo selon deux modes. La première consiste à capter les images à partir d'une camera NTSC (30 images/seconde), le contrôleur DMA permet le transfert d'une ligne de 720 pixels au format 4:2:2 dans chaque cycle. Cette ligne est enregistrée dans le cache L1. Le deuxième mode consiste à enregistrer la vidéo dans L1 à partir d'une camera PAL (25 images/seconde). Dans chaque cycle de transfert le contrôleur DMA permet de transférer aussi une ligne de 720 pixels au format 4:2:2.
- Stockage dans SDRAM : En raison du faible dimension de L1, le contrôleur DMA doit transférer les données à partir du cache L1 à la mémoire externe (SDRAM) qui est extensible. Dans cette mémoire il y a trois buffers pour l'enregistrement des images brutes, cette technique de partitionnement de la mémoire fait que la lecture et l'écriture s'exécutent d'une façon parallèle (pipeline) comme l'indique dans la figure I.12. On ne peut pas utiliser un seul tampon parce que la lecture et l'écriture n'ont pas le même rythme et le processeur doit ignorer (skipping) quelques images pour que l'application soit temps réel. En effet il est moins rapide que la cadence d'arrivée des MBs.

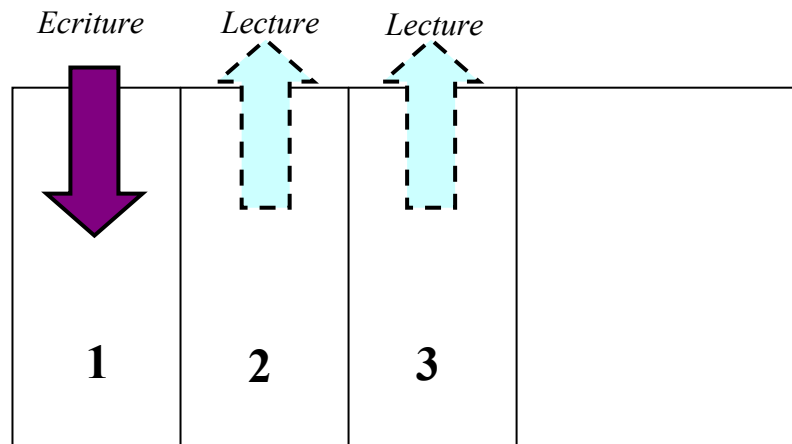


Figure II. 12 : Lecture et écriture des images brutes

II.3.2.3. Processus de codage H.264

La norme H.264 exige l'utilisation des images au format 4:2:0 à l'entrée du codeur, c'est à dire un MB au format 4:2:0 se compose d'un MB en luminance et de deux blocs 8x8 en chrominance, sa taille est 384 octets. Pour ce faire la carte effectue les opérations suivantes ;

- Lecture de nouveaux MBs : La lecture des données vidéo à partir du SDRAM et l'écriture dans la mémoire cache L1 se fait MB par MB au format 4:2:2, c'est à dire chaque MB comporte une composante en luminance et une composante en chrominance dont la taille totale est 256x2 octets. A titre d'indication la largeur et la fréquence d'un bus de donnée DMA sont 64 bits et 133 MHz, ces chiffres sont utiles pour calculer la durée nécessaire de transfert d'un MB.
- Reformatage des images : C'est une étape critique, elle permet la conversion des images numériques du format CIF 4:2:2 au format CIF 4:2:0, ce dernier est exploitable par le codeur H.264. Dans des nouvelles plateformes la conversion se fait au niveau de port PPI. Le bloc de reformatage est constitué de 5 MIPS (5 Million instructions par seconde).
- Cycle de codage : Le codage est constitué de 8 étapes (Prédiction Inter, Prédiction Intra, Transformation, Quantification, Quantification inverse, Transformation inverse, Filtrage et codage entropique), qui s'exécutent d'une façon séquentielle, comme l'illustre la figure II.13. La tâche de prédiction est formée de 120 MIPS, les tâches de Transformation-quantification, transformation-quantification inverse, filtrage et codage entropique sont constituées chacune de 60 MIPS. Les MBs reconstruits sont enregistrés dans le cache L1

ensuite, elles sont stockées dans le SDRAM. Ces images reconstruites sont utilisées ensuite comme des références pour le codage des images récentes. Les données quantifiées et codées seront enregistrées dans la mémoire partagée L2 pour que le deuxième core A puisse la traiter et l'adapter au réseau (formation de bitstream).

Le transfert des images de références dépend de la taille de la zone de recherche et le nombre de références sachant que la norme H.264 autorise l'utilisation de 17 références pour le codage inter.

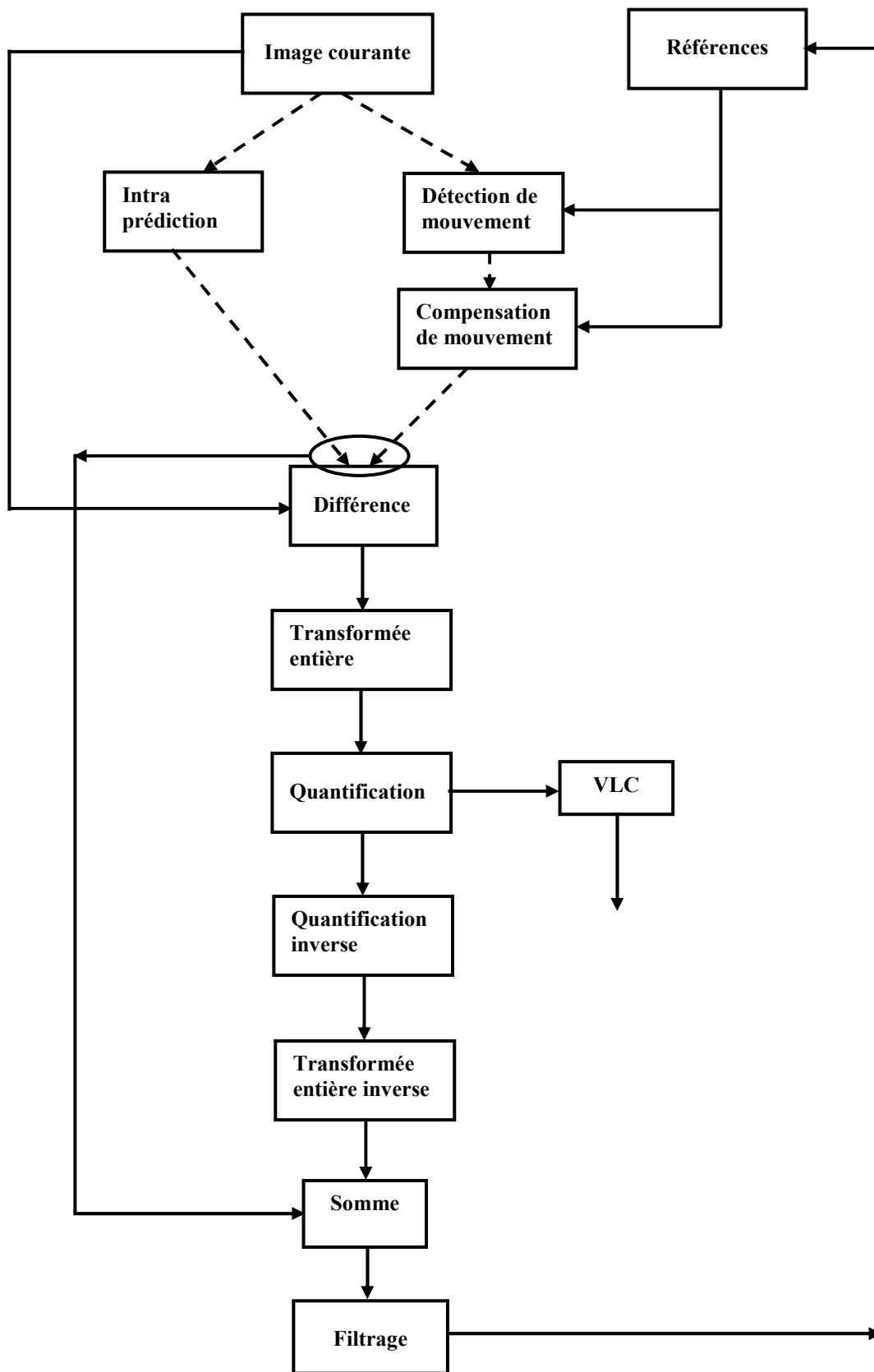


Figure II. 13 : Processus de codage H.264

II.4. Conclusion

Dans ce chapitre, nous avons donné un aperçu sur la norme de compression H.264, ses spécificités, ses applications. Nous avons, ensuite, décrit la solution implémentée sur la plateforme Panview d'ANALOG DEVICES tout en évoquant les détails de chaque bloc qui intervient dans la tâche de codage.

Dans le chapitre suivant nous allons proposer une nouvelle architecture pour l'implémentation d'un codeur vidéo H.264 sur VDK.

Chapitre III

Conception et mise en œuvre de la solution codeur H.264 sur VDK

III.1. Introduction

Dans tous les domaines de télécommunications la migration d'une génération à une autre consiste à supprimer des modules et d'en ajouter d'autres dans le but de trouver une manière de fonctionnement plus élégante.

Dans ce contexte s'intègre notre troisième chapitre qui décrit la mise en oeuvre et l'évaluation des performances d'un codeur H.264 sur VDK. La première partie sera consacrée pour présenter les architectures de partitionnement possibles en se referant sur les critères développés dans le premier chapitre. Nous nous serons intéressés, dans la deuxième partie à la description des détails d'implémentation et au choix des différents paramètres. Une dernière partie est dédiée à l'évaluation des performances de la nouvelle solution et l'étude comparative avec la solution déjà existante.

III.2. Configurations possibles du partitionnement

Les deux critères de partitionnement de l'application en threads développés dans le premier chapitre nous mènent à suggérer plusieurs solutions ;

- **Architecture 1:** Nous avons abouti à cette solution selon le premier critère " partitionner l'application en thread selon la fonctionnalité de chaque bloc" qui consiste à regrouper en un seul thread les blocs qui réalisent le même type de tâche. Dans cette solution chaque bloc forme un thread qui s'exécute indépendamment des autres et il est responsable à la réalisation de sa tâche dédiée, comme l'illustre la figure III.1.

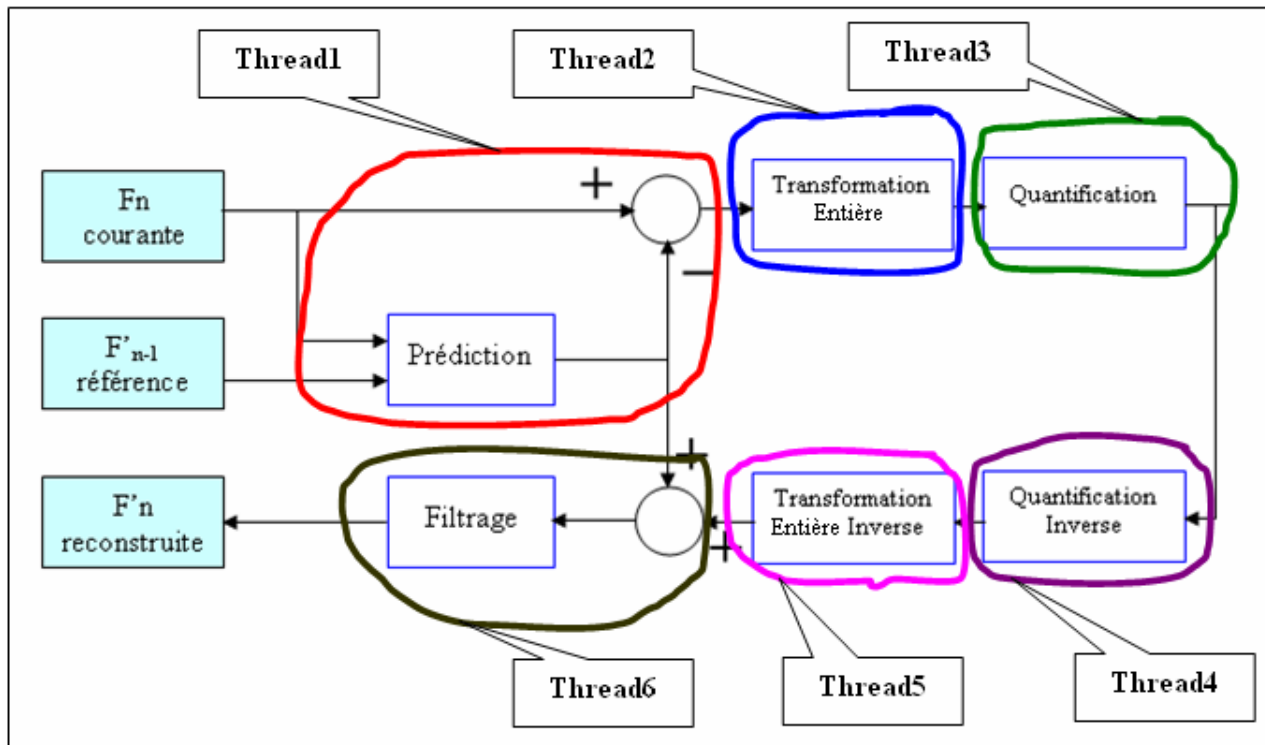


Figure III. 1 : Schéma du codeur selon l'architecture 1

Chaque thread demande des messages pour la communication avec les autres, une espace mémoire pour l'enregistrement de ses variables locales, une pile et des registres, etc. En effet cette architecture présente l'inconvénient d'utiliser plusieurs threads (sur-partitionnement), ceci introduit un temps perdu pour le changement de contexte entre les threads et une utilisation abusive des ressources.

- **Architecture 2:** Selon le premier critère, on peut partitionner l'application en deux threads tel que s'est illustré dans la figure III.2. Le premier consiste à effectuer le codage dans le sens direct et l'autre a pour but de faire la reconstruction.

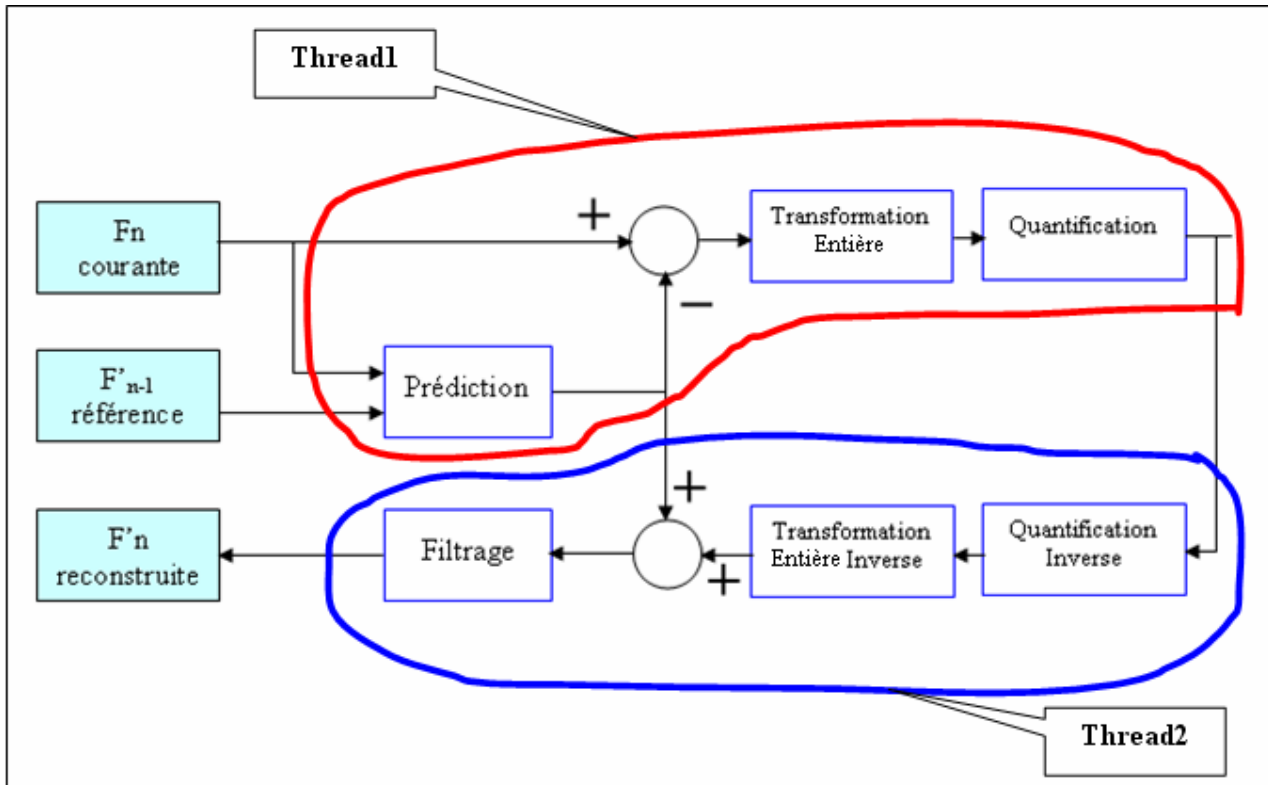


Figure III. 2 : Schéma du codeur selon l'architecture 2

- **Architecture 3:** Selon le deuxième critère " partitionner l'application selon la nature des données" nous avons abouti à cette solution, elle regroupe les blocs de transformation entière, Quantification, Quantification Inverse et Transformation entière Inverse en un seul thread et considère les deux autres blocs comme deux threads (Prédiction et Filtrage), tel que s'est illustré dans la figure III.3.

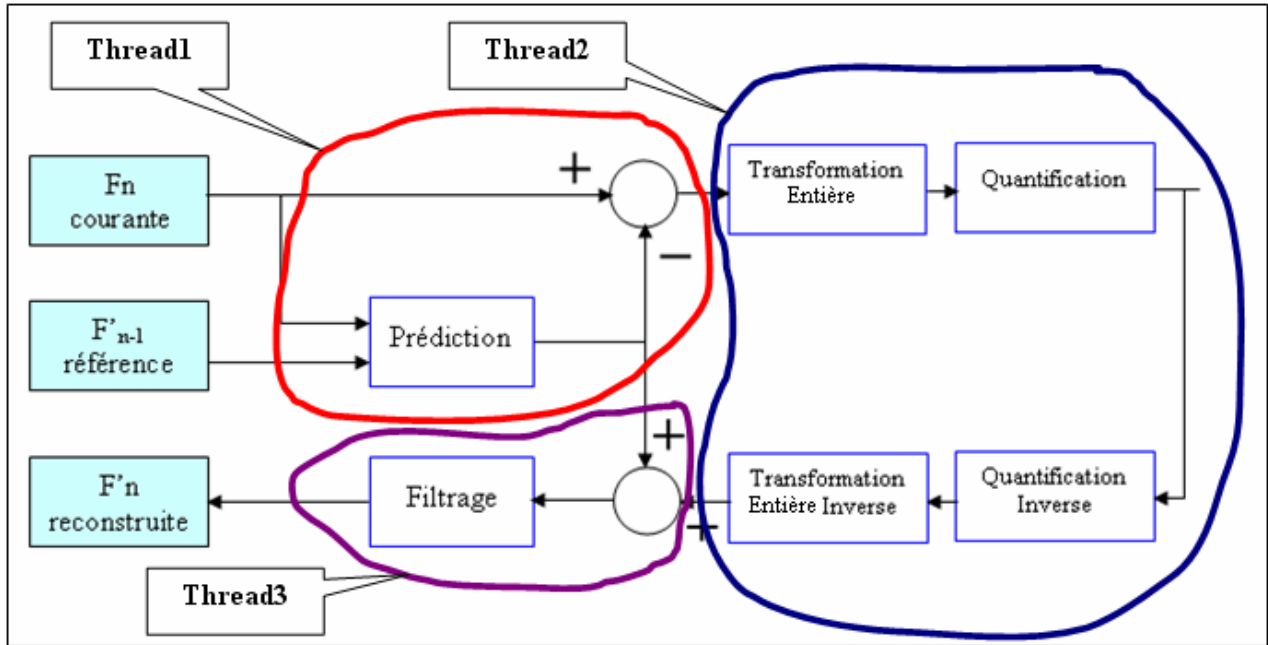


Figure III. 3 : Schéma du codeur selon l'architecture 3

- **Architecture 4:** Selon le deuxième critère on peut regrouper les tâches de filtrage, transformation, quantification, quantification inverse et transformée inverse en un seul thread tout en occupant le thread de prédiction, comme l'illustre la figure III.4.

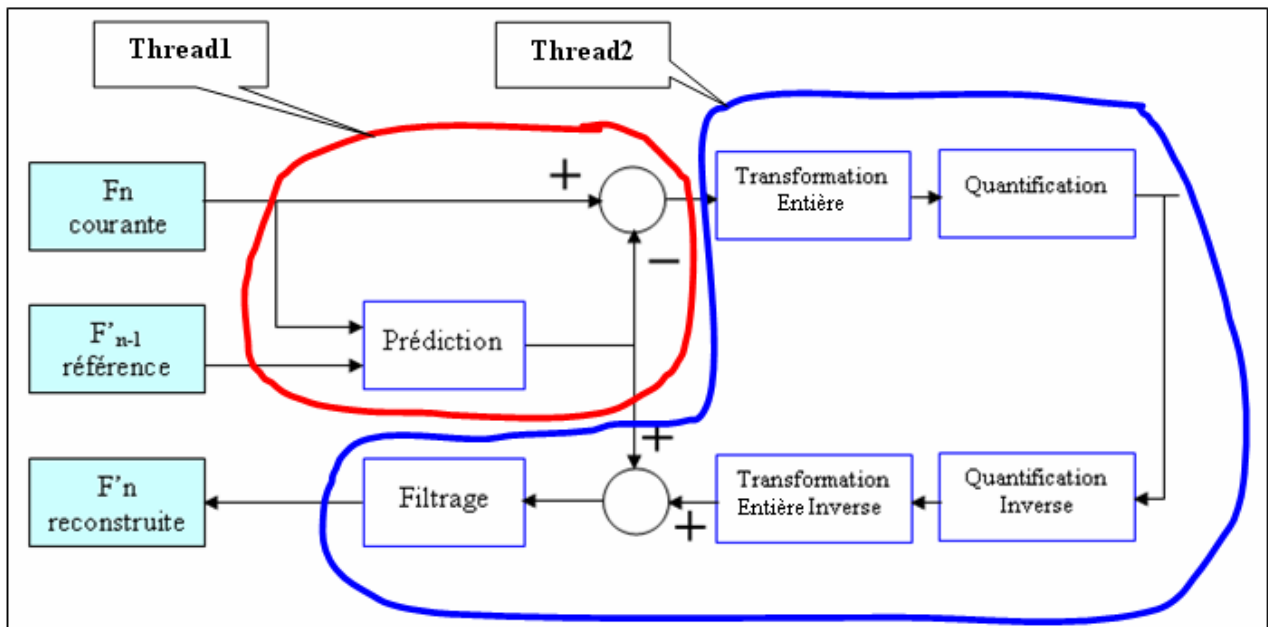


Figure III. 4 : Schéma du codeur selon l'architecture 4

III.3. Choix de la configuration optimale

Une configuration optimale doit vérifier les paramètres suivants:

- Augmenter la rapidité d'exécution et le taux d'utilisation du processeur.
- Optimiser l'utilisation des ressources.
- Rendre la maintenance et le test du code plus facile.
- Réduire le coût d'implémentation.

Dans notre étude nous nous sommes intéressés seulement aux deux premiers paramètres d'évaluation de performance puisque nous avons intérêt à augmenter les performances du codeur en terme de rapidité d'exécution et optimiser l'utilisation des ressources sans introduire les autres paramètres qui consistent à réduire le coût d'implémentation et de rendre la mise à jour plus aisée.

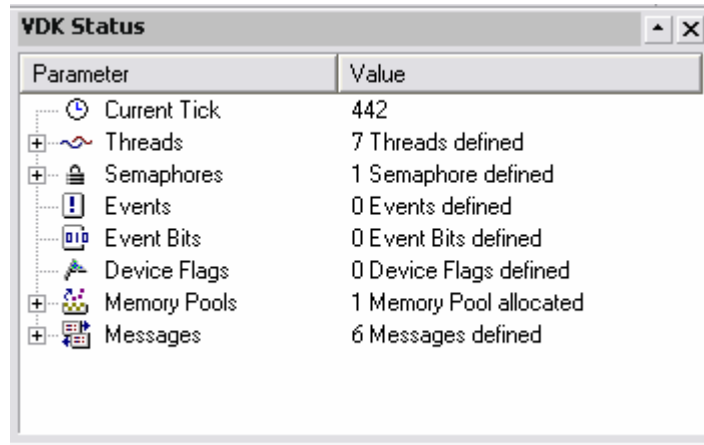
III.3.1. Conditions de test

L'étude comparative consiste à implémenter les quatre configurations en respectant la durée d'exécution de chaque bloc, ensuite nous allons exécuter chaque modèle pendant des durées égales sous les conditions suivantes ;

- La cadence d'arrivée des nouveaux MBs égale à 100 ms, en effet le choix de ce valeur n'est pas aléatoire car la fréquence réel d'arrivée des MBs est 0.1 ms (cadence de la camera), mais en VDK on ne peut pas faire une simulation avec cette valeur ce pour cela que nous avons multiplié toutes les valeurs par 1000.
- Le bloc de prédiction est constitué de 120 MIPS.
- Les blocs de Transformation entière, Quantification, Quantification inverse et Transformation inverse est constitué chacune de 30 MIPS.
- La tâche de filtrage occupe 60 MIPS.
- La communication entre les threads se fait par les messages.
- Le Boot thread permet la création des autres threads ensuite il va être détruit.
- L'arrivée de nouveaux MBs est modélisé par un sémaphore qui s'incrémente par le Kernel tous les 100 ms (100 ticks).
- Un tick est égale à 1 ms, c'est un intervalle de temps dans lequel le thread s'exécute sans préemption.

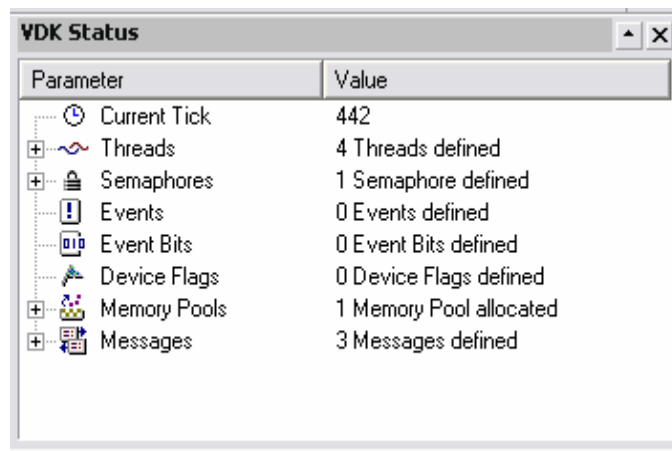
III.3.2. Optimisation des ressources

Dans chaque configuration, nous allons présenter le nombre des threads, le nombre des sémaphores et le nombre des messages utilisés, comme l'illustrent les figures III.5, III.6, et III.7.



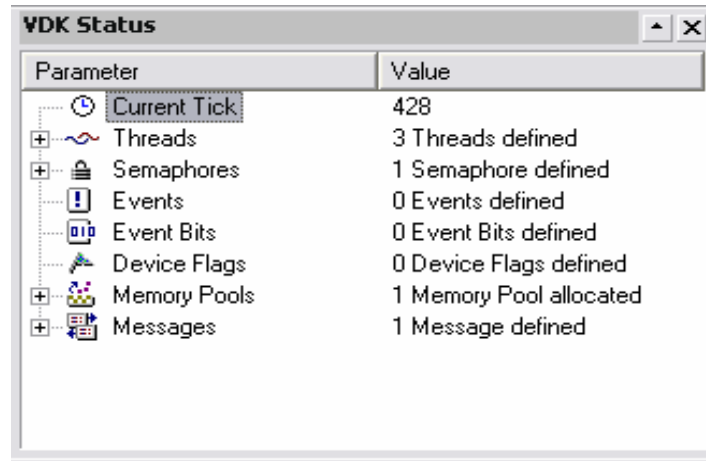
Parameter	Value
Current Tick	442
Threads	7 Threads defined
Semaphores	1 Semaphore defined
Events	0 Events defined
Event Bits	0 Event Bits defined
Device Flags	0 Device Flags defined
Memory Pools	1 Memory Pool allocated
Messages	6 Messages defined

Figure III. 5 : Les ressources utilisées dans la première architecture



Parameter	Value
Current Tick	442
Threads	4 Threads defined
Semaphores	1 Semaphore defined
Events	0 Events defined
Event Bits	0 Event Bits defined
Device Flags	0 Device Flags defined
Memory Pools	1 Memory Pool allocated
Messages	3 Messages defined

Figure III. 6 : Les ressources utilisées dans la troisième architecture



Parameter	Value
Current Tick	428
Threads	3 Threads defined
Semaphores	1 Semaphore defined
Events	0 Events defined
Event Bits	0 Event Bits defined
Device Flags	0 Device Flags defined
Memory Pools	1 Memory Pool allocated
Messages	1 Message defined

Figure III. 7 : Les ressources utilisées dans les architectures deux et trois

La simulation sur VDK a montré que la première architecture a utilisé :

- Sept threads
- Un sémaphore
- Six messages pour la communication inter-threads.

La troisième configuration est constituée de quatre threads qui ont besoin de trois messages pour l'échange des données et un sémaphore pour la synchronisation. L'architecture deux et quatre sont constituées chacune de trois threads, un sémaphore, et un message. Ces dernières utilisent moins des ressources que les autres configurations, donc d'après ce premier critère on peut choisir l'une des deux configurations.

III.3.3. Taux d'utilisation du processeur

L'environnement ADSP++ offre un mode d'historique permettant de visualiser les événements subis par les threads et aussi il permet de représenter la charge du processeur durant la période d'exécution (running), comme l'illustre la figure III.8. Avec cette option on peut analyser la charge du processeur pour les quatre configurations, comme l'illustrent les figures III.9, III.10, III.11, III.12.

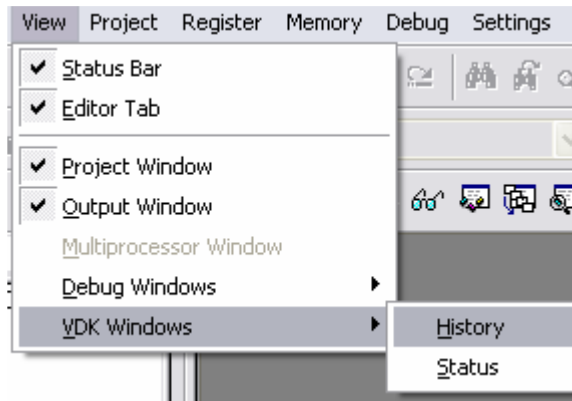


Figure III. 8 : Visualisation de l'historique sous ADSP

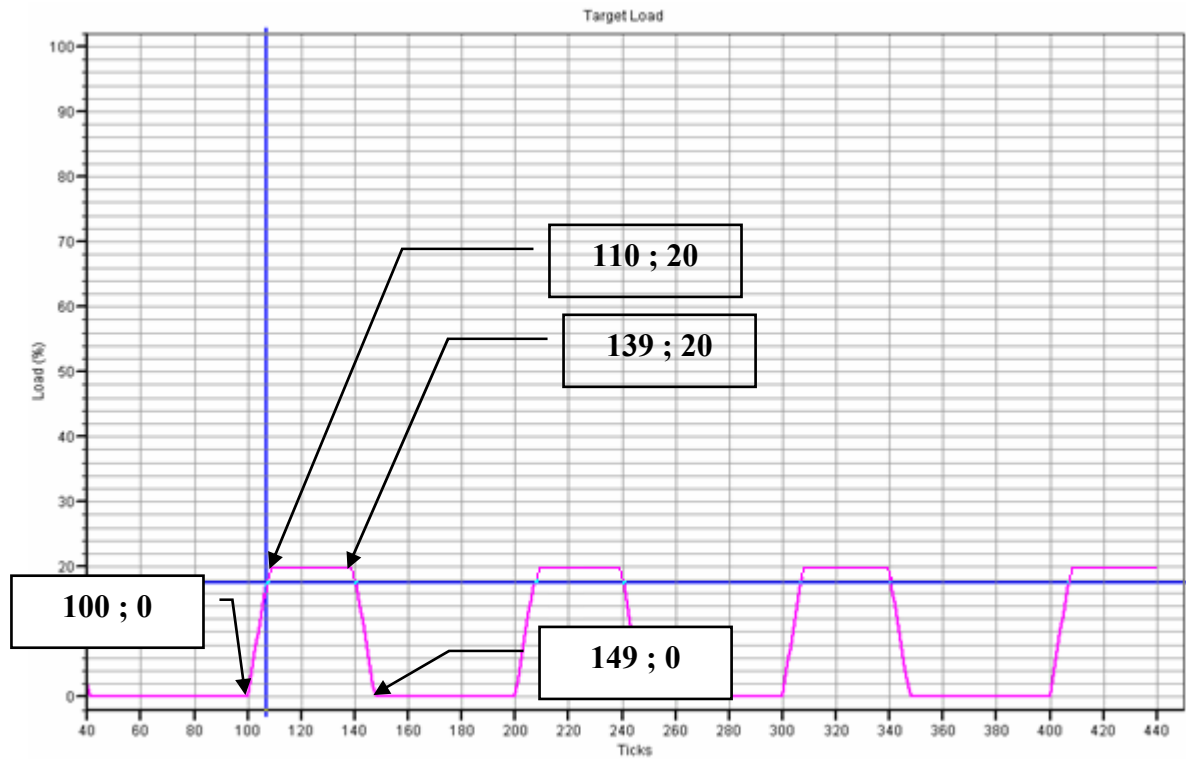


Figure III. 9 : Charge du processeur dans la configuration 1

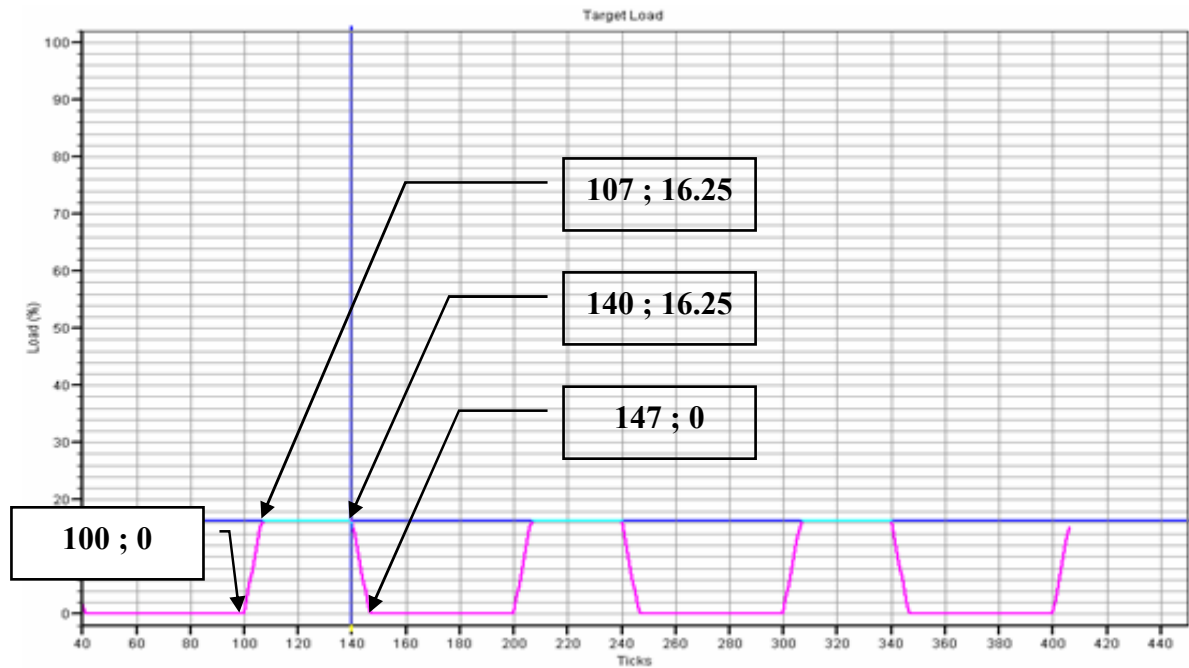


Figure III. 10 : Charge du processeur dans la configuration 2

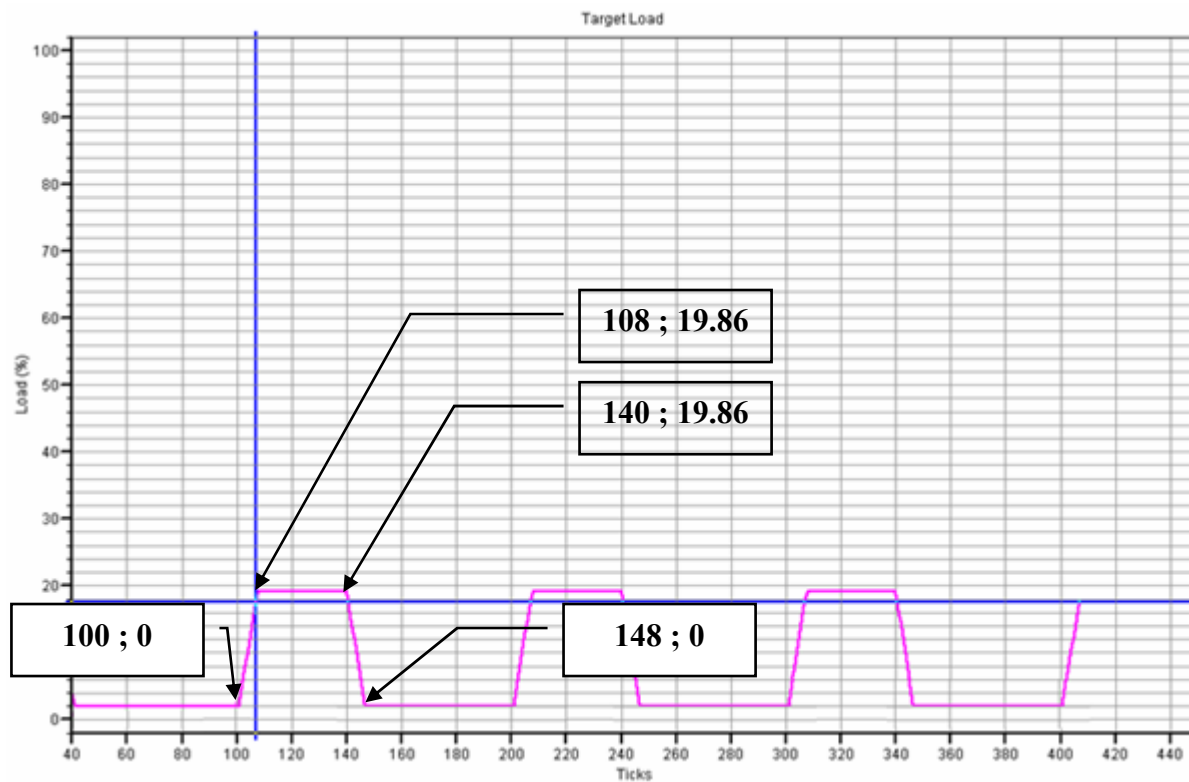


Figure III. 11 : Charge du processeur dans la configuration 3

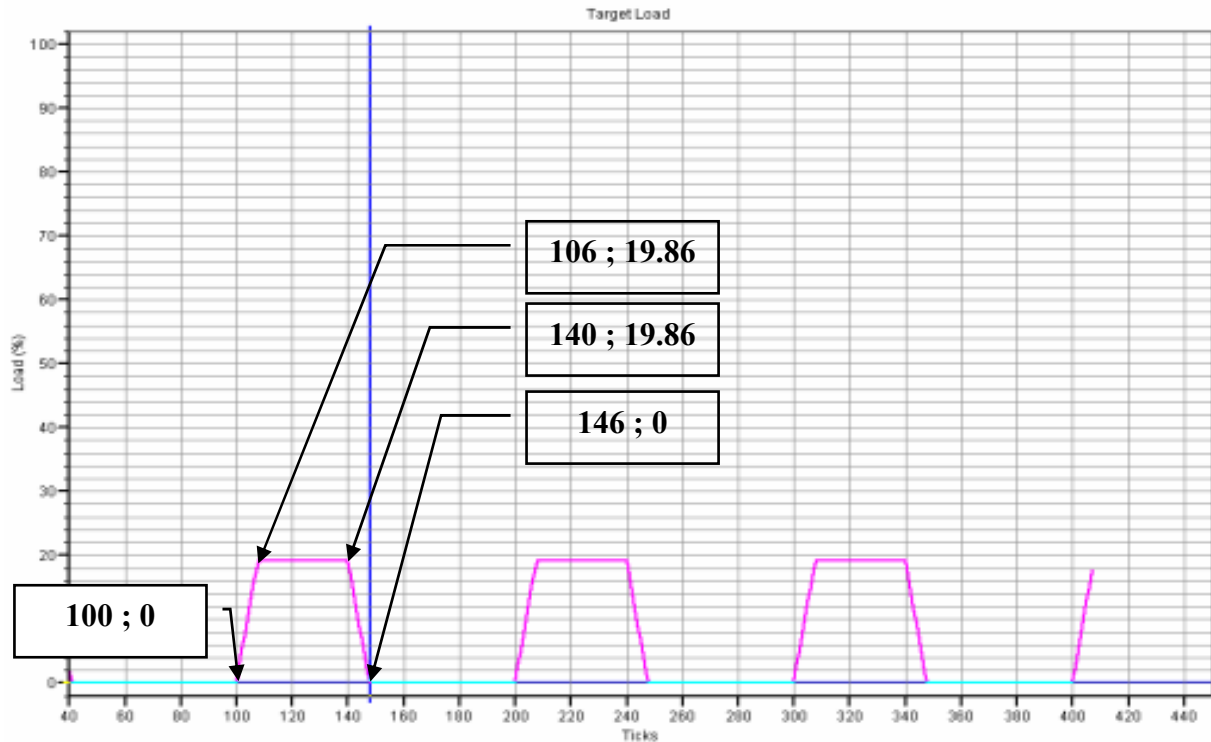


Figure III. 12 : Charge du processeur dans la configuration 4

Dans chaque figure, on représente la charge du processeur en pourcentage en fonction de nombre de ticks. Puisqu'il s'agit de la même tâche à exécuter dans chaque expérience, la quantité de travail effectué par le processeur est supposée la même dans les quatre configurations. Mais les durées d'exécutions et le taux d'occupation du processeur sont différentes d'où l'idée de choisir la configuration qui présente un temps d'exécution minimal et un pourcentage de charge maximal.

Nous pouvons considérer que la quatrième configuration est une architecture optimale puisqu'elle présente un temps d'exécution de 46 ticks, un taux max de 19.86 % et elle est moins gourmande en ressources. En effet dès le début nous avons conscience que cette architecture est la meilleure car elle sépare le bloc de prédiction aux autres blocs et elle le considère comme un thread indépendant, à la différence de la deuxième architecture qui le fusionne avec les blocs de transformée entière et de quantification. On remarque aussi que ce bloc communique avec l'environnement extérieur à travers l'interface PPI et si la cadence d'arrivée des nouveaux MBs devient moins important que le temps de codage on aura une famine du processeur car les autres tâches restent aussi en attente avec le bloc de prédiction (cas de la deuxième configuration). Mais nous voulons que les autres tâches ne soient pas bloquées par un événement extérieur (panne de camera, etc.). En effet nous avons pris le pire de cas pour rendre le système plus tolérant aux fautes et s'assurer de son fonctionnement.

III.4. Mise en œuvre de la nouvelle solution

III.4.1. Type et ordonnancement des tâches

Les tâches qui constituent un codeur vidéo H.264 sont fortement liées entre eux, par exemple la tâche de prédiction inter ne peut pas commencer le traitement du MB suivant si le bloc de filtrage est en train de filtrer le MB courant. Le bloc de transformée entière reste en attente de la fin de la tâche prédiction pour qu'il commence l'opération de transformation, etc. Donc ses tâches sont dépendantes entre elles avec une relation de précédence qu'on peut modéliser par un graphe orienté comme l'illustre la figure III.12.

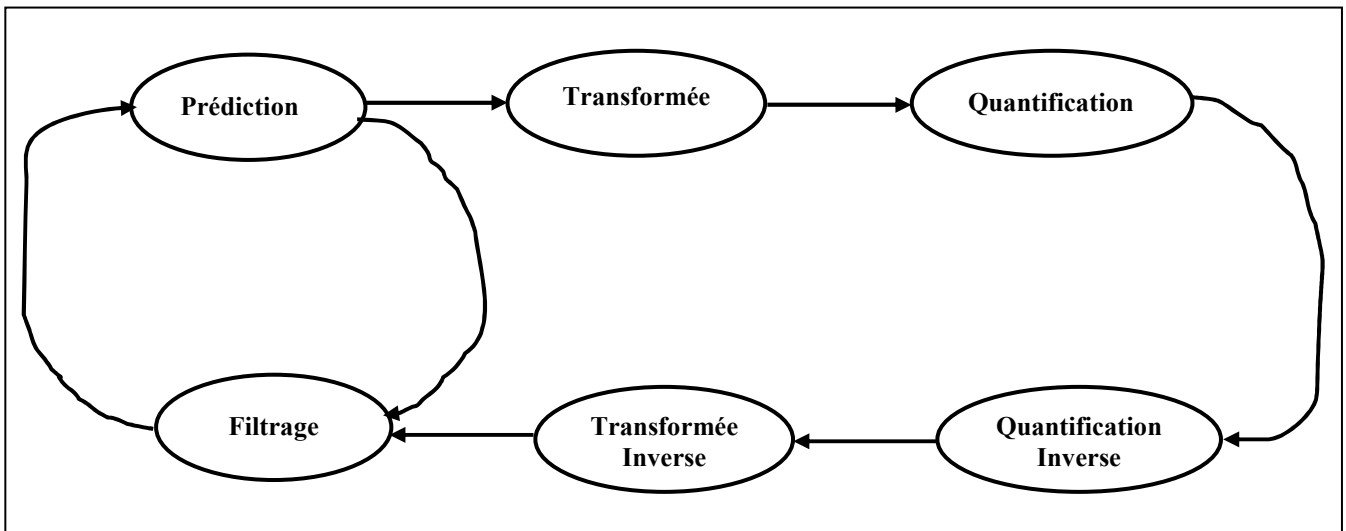


Figure III. 13 : Graphe de dépendance entre les tâches dans un codeur H.264

On remarque aussi que les tâches sont périodiques de période non constante. En effet le temps de codage n'est pas le même pour tous les MBs, par exemple la prédiction inter d'un MB qui ne subit aucun mouvement n'est pas la même d'un autre en mouvement. Donc on peut choisir les priorités fixes pour tous les threads, pour faciliter l'implémentation et puisque les tâches ne peuvent pas être préemptées au cours d'exécution.

On doit gérer les priorités et les mécanismes de synchronisation pour que l'ordonnanceur ne puisse pas allouer le processeur à une nouvelle tâche avant la fin d'exécution de la tâche en cours. On parle donc d'un système d'exploitation coopératif et l'ordonnancement se base sur les priorités.

III.4.2. Description fonctionnelle

La figure III.13 montre la synoptique de fonctionnement d'un codeur vidéo H.264. Les tâches peuvent être soit des transferts de données entre SDRAM et mémoire interne du BF561 effectués par le contrôleur DMA, soit du traitement effectué par l'ADSP BF 561.

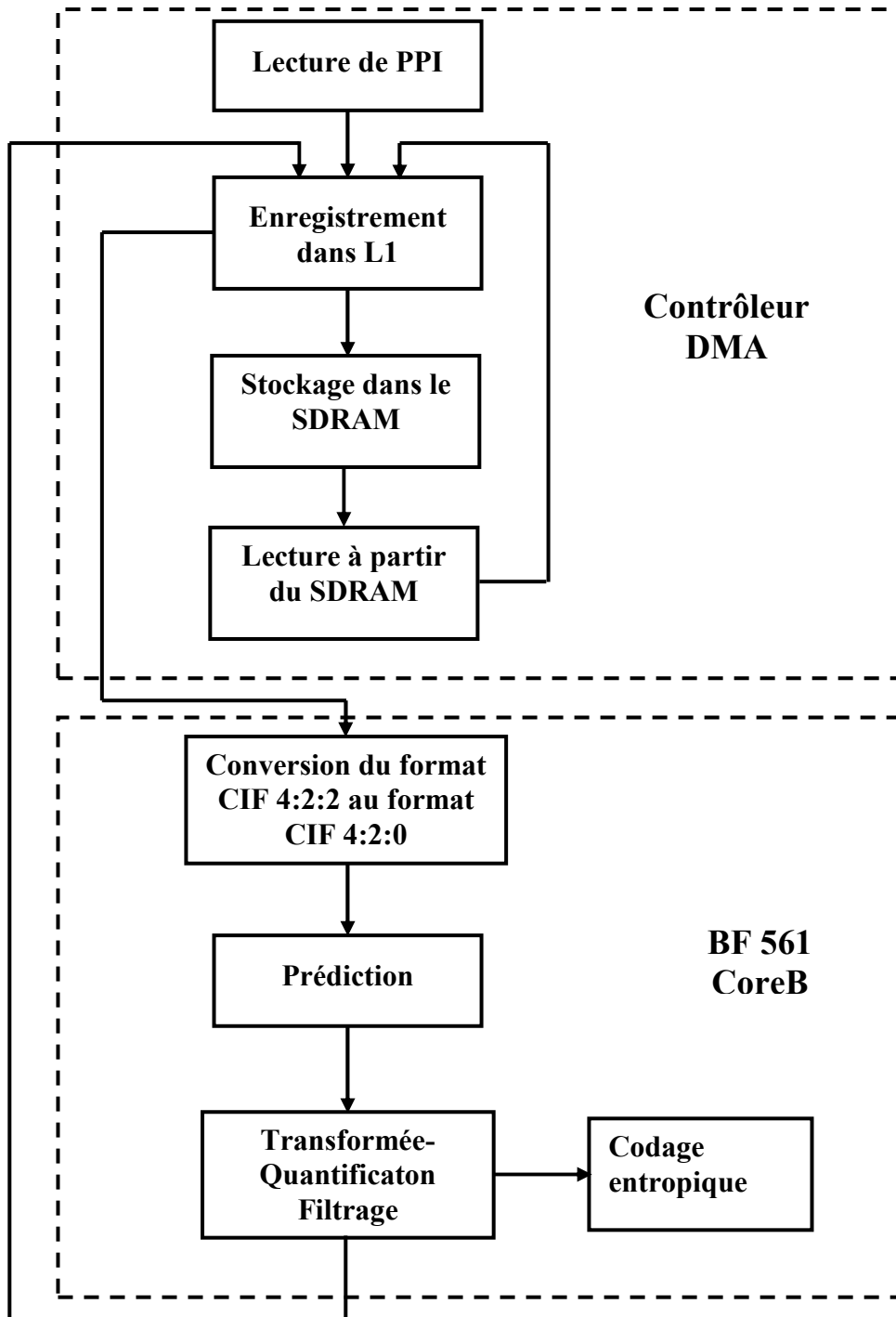


Figure III. 14 : Schéma synoptique d'un codeur vidéo H.264

III.4.2.1. Tâches effectuées par le contrôleur DMA

- Lecture à partir de PPI : PPI_Reader est le thread responsable pour le transfert des données à partir de la camera à la mémoire cache dite L1 à travers le port PPI. Selon le même principe que la version existante, le contrôleur DMA permet de transférer une ligne de 720 pixels au format CIF 4:2:2 s'il s'agit d'une capture à partir d'une camera NTSC, ou bien une ligne de 720 pixels aussi au format CIF 4:2:2 quand la carte est connectée à une camera PAL. Pour faire fonctionner le processus de transfert, il faut configurer le contrôleur DMA en donnant la taille des données à transférer et l'adresse du buffer destination dans le cache L1. La numérisation ou la conversion de format NTSC ou PAL au format CIF 4 :2 :2 se fait au niveau du port PPI et le module ADV7181 (chapitre 2).
- Stockage dans le SDRAM : Le SDRAM est partitionné en trois parties, la première partition est dédiée pour l'enregistrement des images brutes au format CIF 4 :2 :2, la deuxième zone est destinée pour le stockage des images références au format CIF 4 :2 :0 et la troisième partition utilisée pour le stockage des paquets qu'elles vont être envoyées au décodeur distant à travers l'interface Ethernet, comme le montre la figure III.14. Chaque partition est sous-partitionnées en des tampons dont la taille correspond à une image. Le thread SDRAM_Write permet de transférer les données à partir des tampons internes et les stocker dans la mémoire SDRAM. La configuration du contrôleur DMA consiste à spécifier la taille des données et les adresses de deux tampons (interne et externe).

Pour organiser l'accès en lecture et en écriture aux tampons internes et externes, on utilise un Mutex (verrouiller ou déverrouiller) et deux sémaphores qui indiquent l'état du tampon.

- Lecture à partir du SDRAM : La lecture des MBs références, des nouveaux MBs et les trames est effectuée par le thread SDRAM_Reader, qui consiste à chaque cycle de transfert d'apporter les données à partir des tampons externes aux tampons internes. Les trames seront enregistrées dans le cache L2, ensuite elles sont envoyées à travers l'interface Ethernet. La lecture des nouveaux MBs au format CIF 4:2:2 se fait MB par MB ou bien nous pouvons transférer à chaque fois un ensemble des MBs pour éviter un manque de la ressource. Pour déterminer le nombre N des MBs à transférer ensemble il faut vérifier l'inéquation $N.TC \geq TTN$ (1)
avec $TTN = f(N, TT, etc.) \neq N*TT$;

et TC : Durée de codage minimale

TT : Temps de transfert d'un MB.

N : Nombre des MBs à transférer.

TTN : Temps de transfert de N MBs.

Dans le cas contraire, si on suppose que le temps de transfert de N MBs est égal au produit de temps de transfert d'un seul MB et le nombre N et que le temps de codage d'un MB est inférieur au temps de transfert, on aura une famine du processeur dans tous les cas.

Par exemple si $TC = 1$ s et les temps de transfert de N MBs sont donnés par le tableau suivant :

N	TTN (s)
1	1.5
2	1.9
3	2.5
4	2.9
5	3.5
6	4

Tableau III. 1: Les temps de transfert des MBs

D'après l'inégalité (1) on peut choisir $N = 3$ pour s'assurer du bon fonctionnement du codeur.

Pour les MBs de référence, le contrôleur DMA fait un seul transfert, sa taille dépend de la zone de recherche pour l'estimation du mouvement.

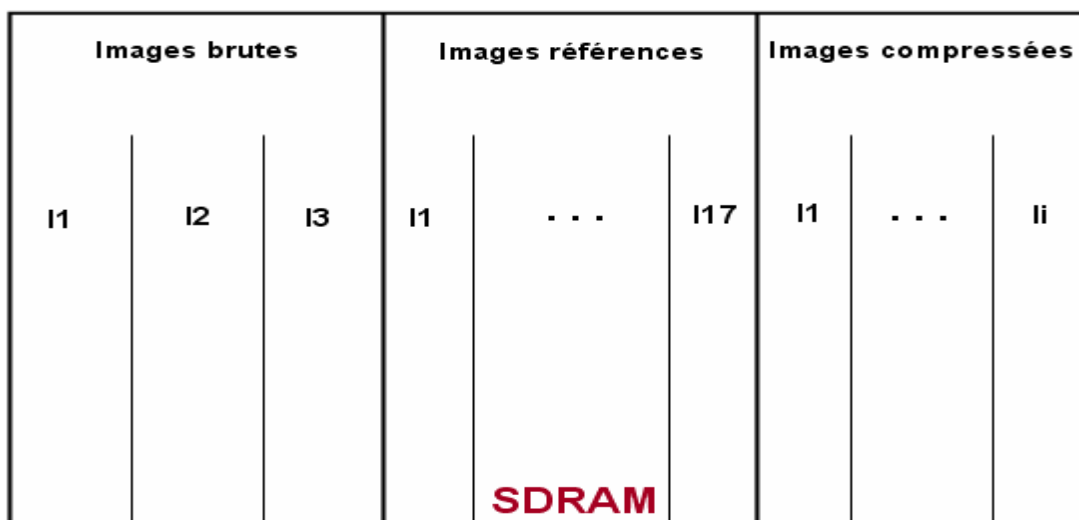


Figure III. 15 : Organisation de SDRAM

III.4.2.2. Tâches effectuées par le processeur

- Reformatage de l'image : La norme H.264 exige que les nouveaux MBs soient au format 4:2:0, donc il faut faire un reformatage des images du format CIF 4:2:2 au format CIF 4:2:0 pour pouvoir les traiter. Cette tâche peut être intégrée avec le thread de prédiction.
- Prédiction : Le thread de prédiction permet d'accomplir la tâche de prédiction inter ou intra pour les blocs de luminance et des chrominances. Le codage intra consiste à prédire les valeurs des échantillons qui forment le même bloc selon tous les modes (neuf modes pour un bloc 4x4, quatre modes pour un bloc 16x16 en luma et aussi quatre modes pour un bloc 8x8 en chroma) à partir des échantillons qui se trouvent dans la même image.

Le codage inter consiste à estimer et compenser le mouvement subit par le bloc en se référant à des images codées antérieurement. Un critère fréquemment utilisé pour estimer le mouvement consiste à déterminer la somme des différences entre les échantillons prédits et des échantillons courants (SAD). La première étape dans le cadre de la prédiction inter consiste à vérifier si le MB est fixe en comparant le SAD centre à un seuil T. Si le SAD inférieur à T on peut considérer qu'il est fixe, sinon on doit passer à la deuxième étape pour vérifier les partitions 16x16, 8x16, 16x8, 8x8, si le mode 8x8 est choisie on doit vérifier avec les sous-partitions 4x8, 8x4, 4x4. En H.264 on peut atteindre une précision de $\frac{1}{4}$ pixel pour l'estimation et la compensation du mouvement, mais ceci demande trop de calculs.

L'inter-prédiction utilise l'algorithme d'estimation de mouvement ARPS (Adaptive Rood Pattern Search). L'algorithme de recherche ARPS utilise un modèle adaptatif, celui-ci se base sur deux modèles de recherche.

- Le premier appelé ARP (Adaptive Rood Pattern) a une taille variable qui sera déterminée pour chaque bloc en s'accordant à son mouvement estimé. Ce modèle n'est pas exploité seulement comme un premier pas de l'algorithme, mais il permet d'orienter la recherche dans le sens de mouvement et de déterminer le bon point de départ.
 - Le deuxième, appelé URP (Unit-size Rood Pattern), est un modèle compact de taille fixe utilisé pour la deuxième étape nommée étape de raffinement.
- Transformation, quantification, filtrage: Ce thread permet d'effectuer la transformée entière qui consiste à décorréler les informations, de concentrer l'énergie aux fréquences basses et d'éliminer les redondances spatiales. La quantification consiste à diviser le bloc transformé

Les devices flags ne sont pas indiqués dans le tableau III.1, mais ils sont utilisés pour la synchronisation entre le programme et l'interface PPI. Aussi les événements et les événements bits sont utilisés pour signaler qu'un certain élément de système est dans un état indiqué.

III.5. Evaluation des performances de la solution adoptée

Pour évaluer les performances de la solution codeur H.264 sur VDK, on peut procéder de deux manières, soit une émulation sur la carte Blackfin 533 EZ_KIT Lite, soit on fait une simulation sur l'ordinateur avec les mêmes conditions (Vitesse du processeur, tâches à effectuées etc.) que l'émulation.

Dans cette deuxième approche s'intègre cette dernière partie de notre PFE qui consiste à simuler un modèle de compression vidéo selon la norme H.264.

III.5.1. Conditions et hypothèses de la simulation

La simulation consiste à implémenter les tâches effectuées par le processeur (Core B) tout en ignorant celles réalisées par le contrôleur DMA. Les conditions de test sont les suivantes :

- Vitesse de processeur 500 MHz.
- Les nouveaux MBs sont toujours disponibles pour le thread de Prediction.
- Le temps de codage est le même pour tous les MBs.
- La tâche de reformatage de l'image est intégrée avec le thread de Prediction.
- Les images références sont disponibles pour l'inter prédiction.
- Nombre des cycles pour la prédiction d'un MB sont 1212.212 cycles.
- Nombre des cycles pour effectuer la transformée, quantification, quantification inverse, transformée inverse et filtrage d'un MB sont 18181.818 cycles.
- Nombre des cycles pour le codage entropique d'un MB sont 6060.606 cycles.
- La communication entre le thread de prédiction et le thread de TQF (transformée, quantification, quantification inverse, transformée inverse et filtrage) se fait par un message et il utilise deux canaux logiques de communication.
- La communication et la synchronisation entre le thread de TQF et le thread de CE (codage entropique) se fait par un tampon partagé et un sémaphore (Mutex).

Le codage se fait d'une façon cyclique et dans chaque cycle le codeur réalise la prédiction, la transformée entière, la quantification, la quantification inverse, la transformée inverse, la filtrage

de boucle et le codage entropique. Le thread de Prediction s'exécute le premier, ensuite celui de transformée, quantification, filtrage (TQF) et enfin le thread de codage entropique (CE) comme l'illustre la figure III.15.

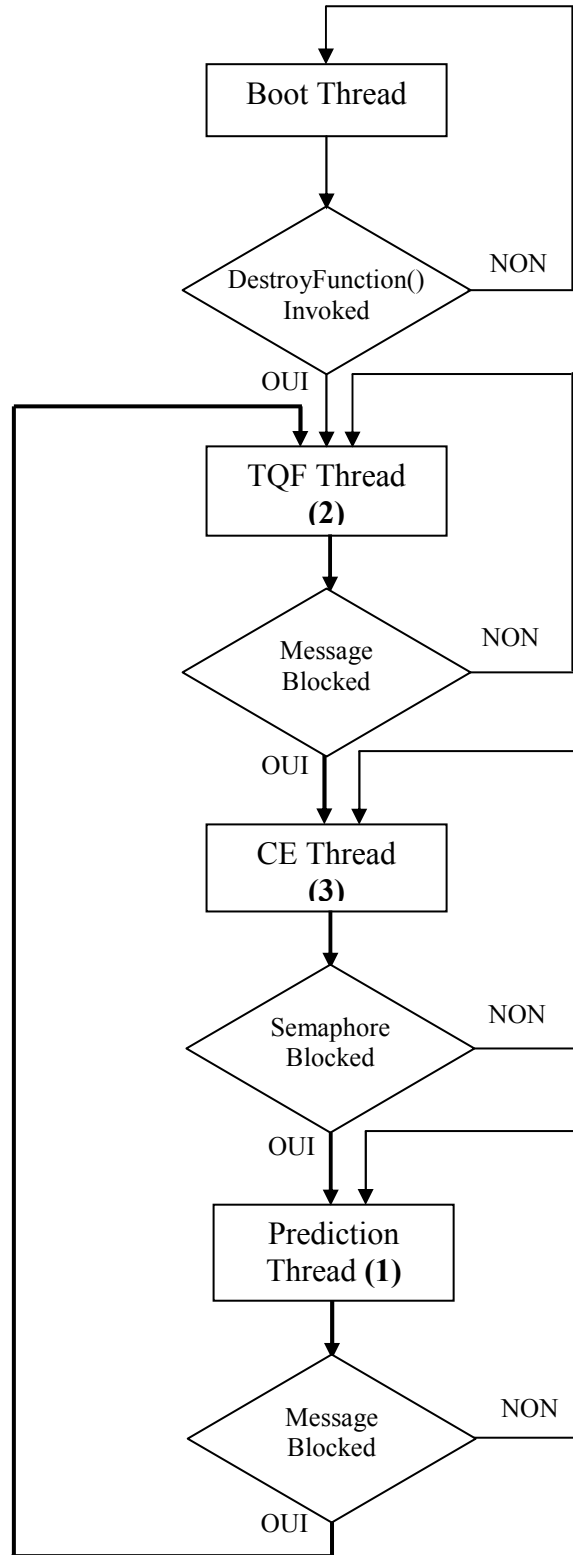


Figure III. 16 : Ordre d'exécution des threads

III.5.2. Simulation et analyse des résultats

Nous avons exécuté ce modèle pendant 118800 cycles, c'est équivalent au codage de 300 images, sachant qu'une image contient 396 MBs. L'étude comparative consiste à déterminer le temps de codage de cette séquence dans la solution existante et celle implémentée sur VDK.

Les figures III.15 et III.16 montrent un aperçu sur l'ordre d'exécution des threads. Après la destruction de boot thread, les autres s'exécutent d'une façon consécutive dans l'ordre indiqué au section précédente.

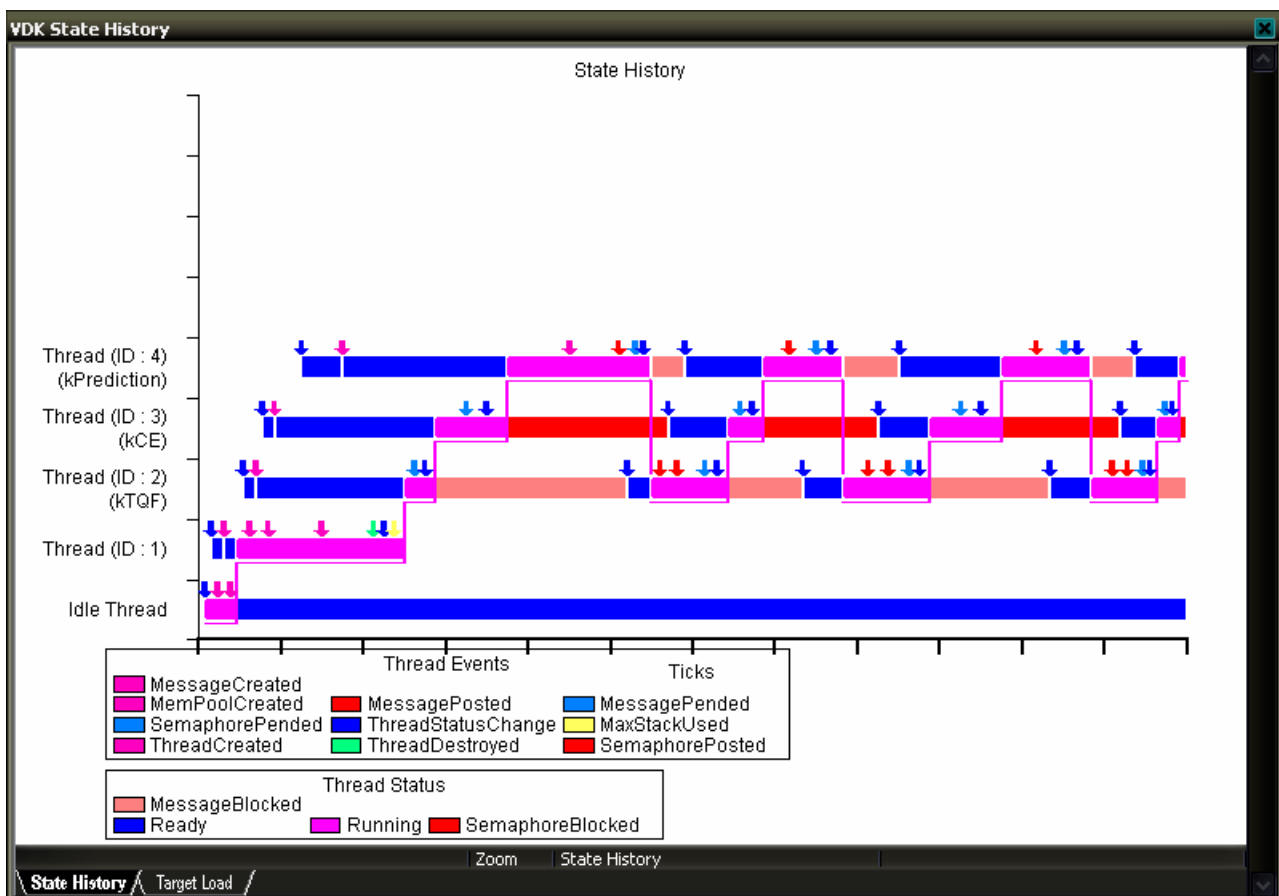


Figure III. 17 : Ordre d'exécution des threads sous VDK

On remarque que le temps du codage dans la solution proposée est plus important que celui offert par la solution existante, malgré qu'elles possèdent le même nombre des cycles pour la prédiction, la transformée, la quantification, la quantification inverse, la transformée inverse, filtrage et le codage entropique. Donc cette solution présente à la fois des avantages et des inconvénients.

a- Avantages :

- Développement plus facile grâce à l'utilisation des sémaphores, des messages, des threads. En effet, l'utilisation des sémaphores au lieu de la gestion des adresses pour accéder aux tampons partagés évite le problème d'inter-blocage entre les threads et rend la mise en œuvre du code plus aisée. Par exemple avec la fonction `PendSemaphore()` le kernel exclut l'accès des autres threads à cette ressource demandée par un thread. VDK, offre également un temps (`TimeOut`) pour l'attente de disponibilité de la ressource, si par exemple un thread veut attendre infiniment on doit choisir un `TimeOut` null. L'utilisation des messages rend plusieurs procédures qui sont complexes transparentes pour le développeur.
- En VDK on peut visualiser l'historique d'exécution des threads ainsi que la charge du processeur, ceci nous permet de vérifier et de s'assurer du fonctionnement de notre codeur.
- Les threads sont moins demandeuses de ressources (mémoire, registre, pile, etc.) que les processus. En peut aussi détruire chaque thread à la fin de son exécution par l'API `VDK_DestroyFunction()`, pour libérer les ressources.
- Avec l'utilisation des threads, on peut introduire un parallélisme dans l'exécution et rendre la mise à jour plus pratique et plus simple.
- Cette solution a un grand succès si les threads sont indépendants ou bien il n'existe pas des relations de précedence entre eux.

b- Inconvénients :

- La communication par message est une routine très complexe, elle demande trop du calcul par le processeur.
- Temps perdus lors de changement de contexte, en effet le changement du contexte est effectué par l'ordonnanceur, c'est un logiciel permettant d'assurer que le thread en cours est actuellement le thread prêt et le plus prioritaire. Cette intervention demande un temps d'exécution très important et par conséquent augmenter la charge du processeur et le temps total du traitement, comme le cas que nous avons simulé.

III.6. Conclusion

Pour développer une application sur VDK il faut tout d'abord la partitionner en threads en respectant plusieurs critères tel que le partitionnement selon la fonctionnalité ou selon la nature des données. Dans la première partie de ce chapitre nous avons proposé une méthodologie de partitionnement, c'est une étape critique pour la mise en œuvre de la nouvelle solution parce qu'elle a un impact sur les performances du processeur. Ensuite nous avons abordé le problème d'ordonnement, de communication et de synchronisation entre les threads pour s'assurer de l'échange des données entre eux et l'exécution sans préemption par un autre thread. Enfin nous avons proposé un modèle de simulation qui nous a permis d'évaluer les performances de la nouvelle solution et de faire une étude comparative entre les deux versions.

Conclusion Générale

Au cours de ce projet de fin d'études, nous nous sommes intéressé à l'étude d'un codeur vidéo H.264 implémenté sur la plateforme Panview d'ANALOG DEVICES et de comprendre ses différents modules afin de proposer une autre solution sur VDK.

Pour ce faire, nous avons commencé par une étude générale sur les OSs embarqués, tout en évoquant l'histoire de développement des processeurs et les trois lois empiriques pour l'évolution de la technologie. Ensuite nous avons dénombré les contraintes imposées par l'environnement extérieur sur les systèmes embarqués ainsi que les étapes de son co-design et son architecture générale. Puis nous avons introduit quelques critères pour choisir un bon OS embarqué pour qu'il puisse effectuer toutes les tâches avant leurs échéances, remplir les contraintes temporelles imposées par le type d'application et optimiser l'utilisation des ressources et de l'énergie. A la fin du premier chapitre, on a fait un survol sur l'outil de développement VDK, c'est un système d'exploitation multi-threads offrant l'exécution parallèle des tâches et les mécanismes de communication et de synchronisation entre eux, comme la plupart des OSs.

La deuxième section est consacrée à présenter un codeur vidéo H.264 implémenté sur la plateforme Panview d'ANALOG DEVICES. Cette étude nous a permis de spécifier les différents modules et fonctions dans cette carte.

Enfin, la dernière partie nous a permis de proposer et mettre en oeuvre un codeur vidéo sur VDK et évaluer ses performances qui sont moins importantes en terme de rapidité d'exécution mais elle est plus facile à implémenter grâce à l'utilisation des sémaphores et des messages pour la communication et la synchronisation entre les threads. Cette nouvelle solution se présente comme une migration à partir d'une version existante avec laquelle on peut s'assurer du bon fonctionnement du codeur même sous des contraintes dures.

Annexe A : Terminologies

DOC: DiskOnChip

Le DOC est une mémoire FLASH intelligente (contenant un BIOS) développée par la société MSystems.

LCD : Liquid Crystal Display

Ecrans à Affichage par cristaux liquides. Utilisés principalement pour les ordinateurs portables.

PDA : Personal Digital Assistant

Assistant numérique personnel. Equipement bureautique de poche entre l'ordinateur portable et l'agenda électronique.

SD : Single Side

Désigne un type de disquette ne permettant le stockage que sur une seule face du support magnétique. Cas des anciennes disquettes 5.25" de 160Ko de capacité

POSIX : Portable Operating System Interface for uniX

Système d'exploitation portable relevant d'une norme définie à l'initiative de groupes d'utilisateurs américains et reprise par l'IEEE en 1984, tournant autour d'un Unix de type System V.

Annexe B : Les concepts de numérisation d'une séquence vidéo

B.1 La notion de pixel

La première étape lors de la numérisation d'une image est son découpage en pixel. Cela signifie que chaque point de l'image est codé par des valeurs numériques. Ces valeurs sont constituées de trois nombres qui représentent l'intensité de rouge R , de vert V et de bleu B dans le point.

B.2 La décomposition en luminance et chrominance

La décomposition du flux vidéo en luminance (intensité lumineuse) et chrominance (couleur) correspond à un changement de base par rapport au codage en RGB (Red, Green, Blue).

En effet, les différences d'intensité lumineuse sont celles les mieux perçues par l'oeil humain. Les équations de changement de bases sont les suivantes :

$$\begin{pmatrix} Y \\ Cb \\ Cr \end{pmatrix} = \begin{pmatrix} 0.299 & 0.587 & 0.114 \\ -0.169 & -0.331 & 0.500 \\ 0.500 & -0.419 & -0.081 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix} \text{ et aussi, } \begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0.402 \\ 1 & -0.334 & -0.714 \\ 1 & 1.772 & 0 \end{pmatrix} \begin{pmatrix} Y \\ Cb \\ Cr \end{pmatrix}$$

Le symbole Y représente la luminance, alors que Cr et Cb correspondent à la chrominance.

B.3 L'échantillonnage de la chrominance

L'échantillonnage est un élément important de numérisation, il existe différentes résolutions et plus l'échantillonnage est grand plus la qualité de l'image est grande.

Dans le format 4 : 4 : 4 aucun échantillonnage n'a été appliqué pour l'image et il y a autant des points échantillonnés pour la couleur que pour la luminosité, comme illustré dans la figure B.1.

Dans l'échantillonnage 4 : 2 : 2 (resp. 4 : 1 : 1), il y a deux (resp. quatre) fois plus de points pour la lumière que pour les couleurs, comme indiqué dans la figure B.2 (a) (resp. figure B.2 (b)).

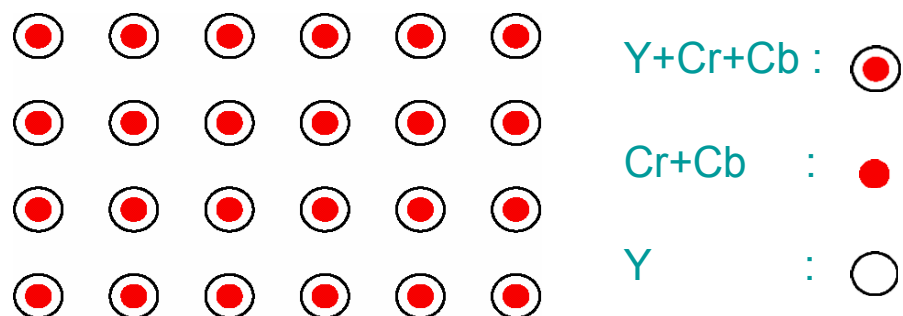


Figure B.1 : Echantillonnage selon la format 4 : 4 : 4

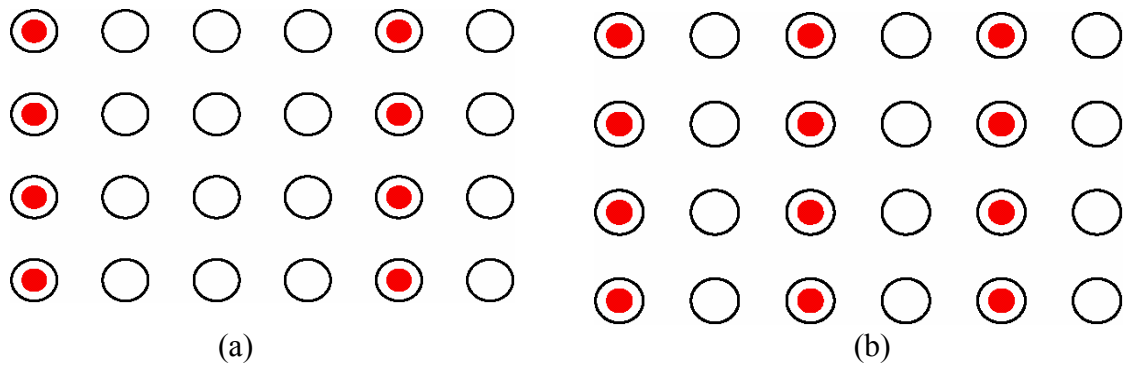


Figure B.2 : Echantillonnage selon les formats 4 : 2 : 2, 4 : 1 : 1

L'échantillonnage 4: 2:0 signifie qu'il y a un échantillonnage horizontal et un échantillonnage vertical d'un facteur de 4 avec positionnement du pixel de chrominance.

La plupart des techniques de codage réalisent leurs opérations de détection ou de calcul sur des échantillons de luminance, car c'est la que l'on trouve les plus forts contrastes. C'est pourquoi on garde un nombre important de valeurs de l'intensité lumineuse.

La structure d'échantillonnage de format 4: 2:0 est donnée par la figure B.3.

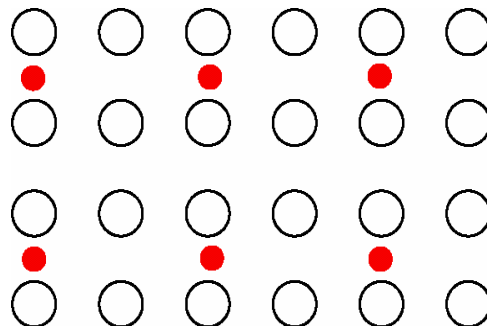


Figure B.3 : Structure d'échantillonnage de format 4 : 2 : 0

B.4 Les formats d'images standardisés

Il existe différents formats d'images standardisés. Les caractéristiques de ceux les plus connus sont données dans les le tableau qui suit.

		SIF (525)	SIF (625)	CIF	QCIF
Nombre de lignes actives	Luminance (y)	240	288	288	144
	Chrominance (Cr et Cb)	120	144	144	72
Nombre de points actifs par ligne	Luminance (y)	360	360	352	176
	Chrominance (Cr et Cb)	180	180	176	88

Tableau B.1 : Les caractéristiques des formats SIF, CIF et QCIF

Bibliographie

- [1] Michel Banâtre and Al, Projet aces "Informatique diffuse et systèmes embarqués", Rapport d'activité INRIA, 2002.
- [2] Adel Ghazel, "Systèmes Embarqués, Partie 1 : Conception & Méthodologie", cours de systèmes embarqués pour les élèves de 3^{ème} année option AST, SUP'COM, 2005-2006.
- [3] Patrice KADIONIK, " Les Systèmes Embarqués. Linux pour l'embarqué", projet Jessica Midi-Pyrénées, LAAS-CNRS TOULOUSE, 2002.
- [4] F. Touchard, "Noyaux et Exécutifs Temps Réel", Cours Temps Réel ESIL Département d'Informatique 3ème année 2005-2006.
- [5] Tarik KHOUTAIF, Fabrice PEYRARD, Thierry VAL et Guy JUANOLE, "Vers l'utilisation des RdPTS pour l'évaluation de performances d'un système de communication sans fil embarqué et temps réel dédié à la robotique", CNRIUT Tarbes, Université de Toulouse le Mirail (Toulouse), 2003.
- [6] Leila Baccouche, "Un Mécanisme d'Ordonnancement Distribué de Tâches Temps Réel", thèse pour l'obtention du grade docteur dans l'institut national polytechnique de GRENOBLE, 22 Novembre 1995.
- [7] Krithi RAMAMRITHAM, "Allocation and Scheduling of Precedence-Related Periodic Tasks" IEEE Transactions on Parallel and Distributed System, Vol.6, No. 4, April 1995.
- [8] Nicolas Navet – Jean-Pierre Thomesse, "L'ordonnancement, la clé d'une gestion efficace des ressources", Equipe TRIO-Laboratoire LORIA à Nancy 2003.
- [9] Analog Devices, Inc,"ADSP-BF533 Blackfin Processor Hardware ReferencePart Number 82-002005-01", Revision 3.1, May 2005.
- [10] Khaled Grati, "Systèmes Embarqués, Partie 4 : Langages et systèmes d'exploitation embarqué", cours de systèmes embarqués pour les élèves de 3^{ème} année option AST, SUP'COM 2005-2006.
- [11] H.261: Video codec for audiovisual services at p x 64 kbit/s. Recommandation H.261 à l'UIT-T. Mars 1993.
- [12] H.263: Video coding for low bit rate communication. Première Recommandation H.263 à l'UIT-T, Mars 1996.

- [13] H.263+: Video coding for low bit rate communication. Deuxième recommandation H.263 à l'UIT-T, Février 1998.
- [14] H.263++ : H.263 Annex U, V, W and X. Compléments de la recommandation H.263 à l'UIT-T, Janvier 2000.
- [15] Standard MPEG-1: ISO/IEC 11172-2, Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s, Avril 1993.
- [16] Standard MPEG-4: ISO/IEC 14496-2, Information Technology – Coding of Audio-Visual Objects, Janvier 2000.
- [17] Standard MPEG-2: ISO/IEC 13818-2, Information Technology – Generic coding of moving pictures and associated audio information, Novembre 1994.
- [18] White paper, Emerging H.264 Standard: Overview and TMS320DM642- Based Solutions for Real-Time Video Applications, Copyright © 2002 UB Video Inc.
http://www.ubvideo.com/public/h.261_white-paper.pdf.
- [19] F. Loras, J. Fourier, "H.264/MPEG-4 AVC, un nouveau standard de compression vidéo" France Télécom R&D/DMI/SGV (Laboratoire de service de visiophonie de Groupe et de visioconférence), « Standards de l'ISO », Mars 2003..
- [20] Iain EG Richardson, "H.264/MPEG-4 Part 10 Intra Prediction," H.264/MPEG-4 Part 10 White Paper 06 Janvier 2003.
www.vcodex.com.
- [21] Iain EG Richardson, "H.264/MPEG-4 Part 10 Inter Prediction," H.264/MPEG-4 Part 10 White Paper, 31 Décembre 2002.
www.vcodex.com.
- [22] "H.264/MPEG-4 Part 10 Transform & Quantification," H.264/MPEG-4 Part 10 White Paper, 20 Décembre 2002.
www.vcodex.com.
- [23] Ralf Schafer, Thomas Wiegand et Heiko Schwarz, "H.264/AVC La norme qui monte", Institut Heinrich Hertz, Berlin (Allemagne), Janvier 2003.

Résumé

Les systèmes embarqués multimédia sont devenus dans ces dernières années un produit de grand public, ils existent par tout ; dans les terminaux de communication sans fil, dans les voitures, dans les usines, dans les systèmes de vidéoconférences, dans les TV numériques, etc. Cette diversité d'utilisation oblige les constructeurs et les concepteurs de ces systèmes de prendre conscience lors de la réalisation software pour qu'ils soient capables d'interagir avec ses environnements, d'effectuer toutes les tâches avant leurs échéances, de servir toutes les interruptions, et d'optimiser l'utilisation des ressources et la consommation d'énergie, etc.

Le but de ce travail est de réaliser un codeur vidéo H.264 sur VDK (Kernel intégré dans l'environnement ADSP++ d'ANALOG DEVICES) avec les concepts des systèmes d'exploitation tout en respectant la fonctionnalité globale de la carte. Nous avons commencé tout d'abord par la proposition d'une méthodologie de partitionnement de l'application en threads selon deux critères, l'un consiste à partitionner selon la fonctionnalité de chaque élément dans l'application et l'autre tend à regrouper en un seul thread les blocs ayant le même type des données afin d'optimiser les ressources puisque dans la plupart des systèmes embarqués ces ressources sont limitées (faible taille de la mémoire, capacité du processeur limitée , etc.).

Mots clés

Système embarqué, OS, RTOS, thread, VDK, ordonnancement, partitionnement, compression vidéo, H.264, processeur, contrôleur DMA, SDRAM.