MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE FERHAT ABBAS – SETIF
UFAS (ALGERIE)

MEMOIRE

Présenté à la Faculté des Sciences de l'Ingénieur
Département de l'informatique
Pour l'Obtention du Diplôme de

MAGISTER

ECOLE DOCTORALE D'INFORMATIQUE (STIC)

Option : ingénierie des systèmes informatiques

Par
Mr : Kara Abdelaziz

THEME

Le développement d'un système embarqué implémentant le Peer-to-Peer pour la TVoIP.

Soutenu le : devant un Jury composé de :

ALIOUAT Makhlouf	M.C. à l'université de Sétif	Président
BOUKERRAM Abdallah	M.C. à l'université de Sétif	Rapporteur
REFOUFI Allaoua	M.C. à l'université de Sétif	Examinateur
SALEM Yacine	Dr. à l'université de Sétif	Invité

Table des matières

Liste des figures

Liste des tableaux

Introduction générale

Au cours de la dernière décennie, la banalisation de la diffusion satellite, du câble numérique, et la naissance de la télévision haute définition ont tous apporté un grand pas d'évolution dans le domaine de la télévision. Cette dernière qui n'a pas connu de réel changement depuis les années soixante-dix avec l'apparition de la télévision en couleurs, risque encore de connaître une authentique métamorphose avec ce qu'on appelle la *Télévision IP* (ou IPTV). La technologie IPTV va surement changer la perspective du grand public vis-à-vis de la télévision usuelle, car cette dernière va apporter aux spectateurs en plus des chaines terrestres et satellites, de la vidéo, des DVD a louer, la VoD, la presse écrite, du contenu web, le PVR (Personnal Video Recorder), la météo locale, du contenu ludique, l'interactivité avec d'autres téléspectateurs...etc. Et tout ça sur un simple téléviseur standard ou haute définition.

Le grand déploiement des connexions résidentielles haut débit comme l'ADSL et l'ADSL2 a rendu l'IPTV une opportunité alléchante pour la diffusion de la télévision sur Internet. Le plus usuel dans cette technologie est d'utiliser une architecture client/serveur, dans laquelle un client établit la connexion avec le serveur et le flux vidéo est directement irrigué du serveur vers le client. Cependant cette architecture présente plusieurs inconvénients, dont le manque de flexibilité, l'appétence à la bande passante, une installation conséquente…etc. D'où l'émergence de l'architecture Peer-to-peer TV. Dans cette architecture et contrairement à l'architecture client/serveur qui s'appuie sur des serveurs et routeurs puissants pour irriguer le flux, c'est le client qui prend cette responsabilité, il prend le rôle du client et du serveur en même temps. De cette manière la structure de l'architecture devient beaucoup plus flexible et moins gourmande en ressources.

Notre solution consiste à développer un système embarqué (le récepteurs-TV) qui implémente l'architecture Peer-to-Peer au lieu de l'architecture client/serveur pour la télévision par Internet, de ce fait le récepteurs-TV ne serait pas obligé de se procuré la diffusion vidéo du serveur (au cas où celui-ci serait saturé) mais d'un autre récepteur qui s'ait déjà procuré la diffusion. Ainsi le FAI n'aurais pas à investir sur un nombre important de serveurs, et éviter les craches de surcharges si le taux d'utilisation est trop élevé. Notre système embarqué est constitué d'une partie hardware ou plutôt d'un SBC (carte mère pour systèmes embarqués) nommée APF9328, cette dernière est une carte mère munie d'un processeur ARM9 tournant à 200 MHz, d'une RAM et d'une ROM de 16 Mégaoctets et d'un FPGA de 200 k portes, fonctionnant sous un système d'exploitation Linux embarqué. L'APF9382 selon ses constructeurs vise principalement les applications de consommation électrique minime, comme les dispositifs de communication, les dispositifs de contrôle industriel, et ce qui concerne notre cas, les applications multimédias. Pour la partie soft du système, l'APF9328 accueil l'algorithme Peer-to-Peer TV DONet développé pour assouvir les contraintes temps réel souples requises par la Télévision IP. Le protocole DONet est un protocole Peer-to-Peer TV très populaire dans le domaine de la recherche, cette popularité vient principalement de ses nombreuses qualités, comme sa simplicité d'implémentation, son

efficacité de transfert, et sa nature totalement distribué. Mais en se qui nous concerne, il a été choisi essentiellement pour son caractère en tant qu'algorithme quasi temps réel adéquat pour une application multimédia temps réel souple comme la notre.

Ce mémoire peut être perçu globalement comme être scindé en trois grandes parties, la première partie traite des systèmes embarqués, et ce compose principalement des quatre premiers chapitres. Le premier chapitre donne une introduction et un premier aperçu sur les systèmes embarqués. Le deuxième présente l'un des plus importants aspects en relation avec les systèmes embarqués, qui est le temps réel. Le troisième chapitre est une étude sur les systèmes d'exploitation propres aux systèmes embarqués, tandis que le dernier chapitre de cette partie se consacre au hardware des systèmes embarqués, plus précisément aux cartes mères pour systèmes embarqué appelées SBC. La deuxième partie qui englobe le cinquième et le sixième chapitre, traite globalement de la Télévision IP. Le chapitre cinq donnes une revue générale sur l'IPTV, tandis que le sixième chapitre se spécifie au Peer-To-Peer TV. La troisième et la dernière partie traite plus précisément de notre implantation de l'algorithme Peer-To-Peer TV sur l'SBC APF9328. Dans cette partie, le septième chapitre se consacre entièrement à la description de l'APF9328 et de sa carte hôte l'APF9328DevFull. Le huitième chapitre démontre le processus d'installation du système d'exploitation et des outils logiciels vitaux pour le fonctionnement correct du système embarqué. Le dernier chapitre de cette partie et de ce mémoire se consacre à proprement dit a l'implémentation de l'algorithme Peer-To-Peer TV DONet sur l'SBC APF9328, ainsi que la description d'un simulateur conçu spécialement pour simuler un environnement Peer-To-Peer TV qui permettra à notre système embarqué de fonctionner comme si il était dans un vrai environnement de diffusion vidéo par IP.

Généralités sur les systèmes embarqués 1

1.1. Introduction

Au cours des deux dernières décennies, nous avons assisté à une forte croissance des systèmes embarqués temps réel. Chaque jour, ces systèmes nous fournissent d'importants services, comme lors de la conduite d'une voiture, ils contrôlent son moteur, ses freins, et ses airbags. Ils réglementent la circulation routière en contrôlant les feux tricolores. Lorsque on prend un vol, ils contrôlent l'horaire, le décollage et l'atterrissage des avions, et maintiennent la trajectoire du vol pour assurer notre sécurité. Lorsque nous sommes malades, ils contrôlent notre pression sanguine et la cadence de nos pulsations cardiaques. Quand nous sommes joyeux, ils nous divertissent avec des jeux électroniques et du contenu multimédia. Ils sont souvent cachés de notre regard. Quand un système embarqué fonctionne correctement, son existence est généralement transparente aux utilisateurs.

Malgré leurs caractères enfuis et furtifs, les systèmes embarqués sont la convoitise de nombreux de chercheurs. L'association entre plusieurs disciplines comme l'électronique, l'informatique, les réseaux de télécommunication, la sécurité informatique constituent le nouveau secteur pluridisciplinaire des systèmes embarqués. Dans ce chapitre nous allons faire un détour non exhaustif sur les principaux axes selon lesquels s'articulent la recherche et l'industrie dans ce domaine. On va d'abord aborder quelques définitions théoriques sur le sujet, quelques illustrations de systèmes embarqués, pour ensuite considérer une des classifications les plus prédominantes dans le domaine, pour terminer avec un aperçu des principaux caractéristiques qui régissent les systèmes embarqués.

1.2. Définition

Michæl Barr définit un système embarqué [1] comme "une combinaison de matériel et de logiciel, et avec peut-être des additifs mécaniques ou autres composants. Qui ont pour but de réaliser une fonction dédiée". Dans certains cas, les systèmes embarqués font partie d'un plus large système ou d'un produit, comme dans le cas d'un système de freinage antiblocage (ABS : Antilock Braking System) d'une voiture.

De nombreux chercheurs s'accordent sur les propos qu'un système embarqué peut être un système électronique et informatique autonome, qui est dédié à une tâche bien précise. Il ne possède généralement pas des entrées/sorties standards et classiques comme un clavier ou un écran d'ordinateur. Le système matériel et l'application sont intimement liés, étant donné que le logiciel embarqué est généralement enfoui, noyé dans le matériel, le matériel et le logiciel ne sont pas aussi facilement discernables comme dans un environnement de travail classique de type ordinateur PC. Le terme s*ystème embarqué* peut bien être utilisé pour désigner le matériel embarqué ou bien le logiciel embarqué, ou les deux réunis.

La *figure*1-1 montre le schéma typique d'un système embarqué, on retrouve en entrée des capteurs généralement analogiques couplés à des convertisseurs Analogiques/Numériques, et en sortie des actionneurs généralement analogiques couplés à des convertisseurs Numériques/Analogiques. Au milieu, on retrouve le calculateur sous forme d'un processeur embarqué raccordé à des périphériques d'Entrées/Sorties. Il est à noter qu'il est complété généralement par une mémoire RAM ou/et ROM et d'un circuit FPGA jouant le rôle de coprocesseur afin de proposer une unité de calcul supplémentaire au processeur. Ou parfois il joue le rôle d'une unité d'entrée/sortie personnalisée pour le support d'une connectivité additionnelle pour le système embarqué.

Sur le schéma on peut remarquer la boucle d'un système d'asservissement entre les entrées et les sorties, c'est ce qui caractérise généralement les interactions des systèmes embarqués avec le monde extérieur. Il est à noter que l'expression qui découle de ce schéma peut inclure la majorité des classes des systèmes embarqués. Pour illustrer ces propos, on peut donner l'exemple d'un système embarqué minimaliste avec comme capteurs des interrupteurs, et comme actionneurs des LED.

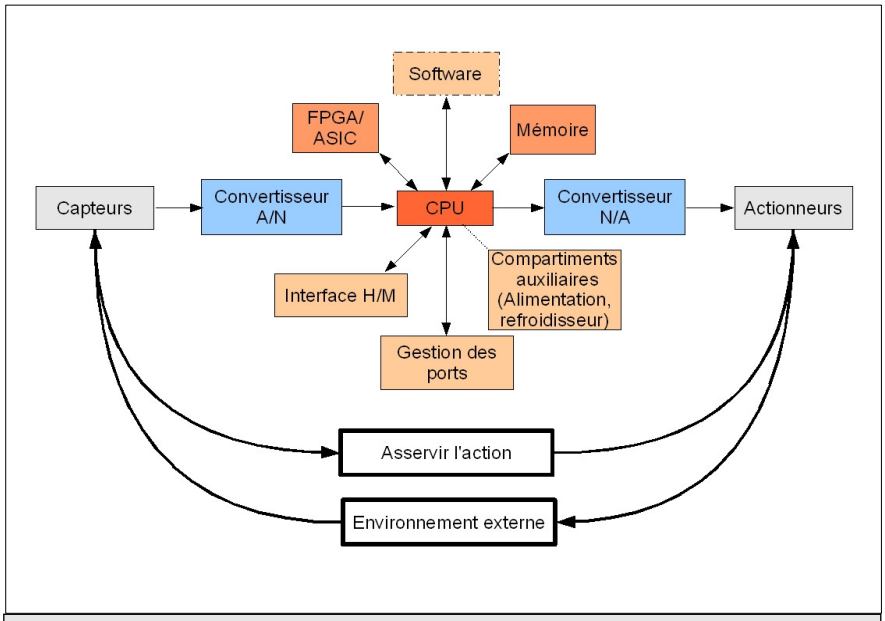

Figure 1-1 : *Système embarqué typique.*

1.3. Quelques exemples de systèmes embarqués

Les systèmes embarqués couvrent tous les aspects de la vie moderne et il existe de nombreux exemples de leur utilisation.

- La consommable électronique : les assistants numériques personnels (PDA), lecteurs mp3, appareils photo numériques, lecteurs DVD, récepteurs GPS.
- La domotique : contrôle de l'éclairage, la climatisation, la sécurité, l'audiovisuel…etc.
- Transport : Automobile, Aéronautique (avionique) …etc.
- Spatiale : fusée, satellite artificiel, sonde spatiale…etc.
- Militaire : missile, radar, vision nocturne…etc.
- Télécommunication : Set-top box, téléphonie, routeur, pare-feu, serveur, passerelles, téléphone portable…etc.
- Électroménager : télévision, four à micro-onde, lave-linge, lave-vaisselle.
- Impression : imprimante multifonction, photocopieur…etc.
- Informatique : disque dur, Lecteur CD…etc.
- Multimédia : console de jeux vidéo, TVHD, PVR (Personal Video Recorder).
- Guichet Automatique Bancaire (GAB).
- Équipement médical.
- Automate programmable industriel.
- Métrologie.

La *figure*1-*2* présente quelques images d'appareils équipés de système embarqué :

Figure 1-2 *: Exemples de système embarqué.*

1.4. Classification des systèmes embarqués

Il existe plusieurs critères de classification pour les systèmes embarqués, comme par exemple le gabarit, le type de calcul, ou le segment du marché économique dans le quel l'appareil figure. Ce dernier critère est le plus utilisé dans la littérature des systèmes embarqués, c'est une classification où les dispositifs embarqués sont à peu près répartis dans des secteurs de marchés divers comme le montre le *Tableau*1 [2] :

Secteur du marché	Système embarqué
Transport (véhicule en générale)	Mis-en-marche du véhicule
	Contrôle du moteur
	Système de freinage
Consommables électroniques	Téléviseur numérique
	PDA
	Domotique
	Jeux vidéo
	Téléphone
	Caméra
	GPS
	Micro-onde
Contrôle industriel	Robotiques et Systèmes de contrôle industriel
	Automate programmable industriel
Médicale	Moniteur cardiaque
	IRM
	Endoscope
	Dialyseur
Réseaux et télécommunication	Routeur
	Hub
	Passerelle
Équipements de bureau	Fax
	Photocopieur
	Imprimante
	Scanner
	Moniteur

Tableau 1 : *Classification des systèmes embarqués.*

1.5. Principales caractéristiques d'un système embarqué

Les caractéristiques d'un système embarqué se révèlent suite à la comparaison entre le système embarqué, étant un système dédié, et le PC étant un système pour usage général. Selon [3] on peut énumérer quelques différences sur la liste qui suit :

- Les systèmes embarqués sont dédiés à des tâches spécifiques, alors que les PCs sont des plates-formes informatiques génériques.

- Les systèmes embarqués sont soutenus par une large gamme de processeurs et d'architectures pour processeur.

- Les systèmes embarqués sont généralement sensibles aux coûts.

- Les systèmes embarqués sont généralement temps réel.

- Si un système embarqué utilise un système d'exploitation. Il est fort probable d'en utiliser un système d'exploitation temps réel (RTOS), plutôt qu'un système d'exploitation pour PC comme GNU Linux ou Microsoft Windows.

- Les craches soft sont beaucoup plus graves dans les systèmes embarqués que dans les systèmes de bureau.

- Les systèmes embarqués ont souvent des contraintes d'énergie.

- Les systèmes embarqués doivent fonctionner parfois dans des conditions d'environnement extrêmes.

- Quelques systèmes embarqués développent une connectivité riche et parfois spécifique à un domaine précis.

- La conception d'un système embarqué est différente de la conception logiciel ou matériel habituel, elle prend en considération plusieurs critères liés à des domaines diversifiés.

Dans les sections qui suivent, chacun de ces points est expliqué.

1.5.1. Les systèmes embarqués sont des systèmes dédiés

Dans la littérature des systèmes embarqués un microprocesseur embarqué est souvent appelé microprocesseur *dédié*. Car il est habituellement programmé pour effectuer une seule, ou à la limite un nombre réduit de tâches spécifiques. La modification de la tâche est généralement associée au changement d'ensemble du système et de la réorganisation de ces composants. Par exemple un processeur qui gère un moniteur cardiaque n'est pas prévu pour faire du décodage/encodage vidéo.

1.5.2. Gamme de processeurs pour systèmes embarqués

De nos jours on apprend que plus de 900 différents microprocesseurs/ microcontrôleurs sont disponibles auprès de plus de 60 fournisseurs de semi-conducteurs [4]. Ces fournisseurs sont continuellement confrontés à des contraintes économiques les obligeant à se concurrencer les uns les autres pour obtenir le meilleur processeur visant des marchés divers et bien spécifiques, comme par exemple les processeurs pour les prochaines

générations de PDA, ou encore des processeurs pour les puces graphiques destinées aux différentes consoles de jeux.

1.5.3. Le coût des systèmes embarqués

Le coût que devrait prendre en général un système embarqué est la plupart du temps lié au coût de l'architecture globale du système (microprocesseur, mémoires et bus). Étant donné que le processeur est en général le composant le plus cher d'un système embarqué, certains concepteurs approuvent que ce dernier soit le facteur principal dans l'évaluation du coût d'un système embarqué. Quoi que cette approbation perde peu à peu sa légitimité en raison de l'apparition des puces SoC (System On Chip). Un SoC typique contient : un processeur, un DSP, de la mémoire (RAM, ROM, flash..), des convertisseurs Analogiques/Numériques Numériques/Analogiques et plusieurs unités d'E/S. En d'autres termes le système tout entier sur une même puce, ce qui réduit grandement l'encombrement, l'alimentation et le coût du système (du point de vue matériel, les systèmes embarqués sont détaillés dans le chapitre 4 : *SBC pour systèmes embarqués*). Le coût d'un système est primordial dans le domaine des systèmes embarqués, car c'est très fréquent que ces petits appareils électroniques soient vendus sur une grande échelle de production. Ce qui fait qu'une petite variation du prix du produit se répercute sur l'échelle de leurs grand nombre par un grand bénéfice ou une grosse perte. Il est souvent dit que les systèmes embarqués sont sensibles au coût.

1.5.4. Les systèmes embarqués et les contraintes temps réel

Généralement, un système embarqué doit respecter des contraintes temporelles, où l'on y trouve enfoui un système d'exploitation ou un noyau Temps Réel (RTOS : Real Time Operating System). Le Temps Réel est un concept un peu vague, on pourrait le définir comme un système ou lors de l'acquisition et le traitement de l'information cette dernière reste encore pertinente. Cela veut dire que dans le cas d'une information arrivant de façon aléatoire (généralement sous forme d'une interruption), les temps d'acquisition et de traitement doivent rester inférieurs à la période de rafraîchissement de cette information. Pour cela, il faut que le noyau ou le système Temps Réel soit déterministe et préemptif pour toujours donner la main durant la prochaine interruption à la tâche de plus forte priorité. (Le temps réel sera plus approfondi dans le chapitre suivant : *Les systèmes embarqués et le temps réel*).

Une confusion classique est de mélanger Temps Réel et rapidité de calcul du système donc puissance du processeur (microprocesseur, microcontrôleur, DSP). On entend souvent : "Être temps réel, c'est avoir beaucoup de puissance processeur (nombre élevé de MIPS, MFLOPS…)", ce qui n'est pas vrai. Par exemple, on pourrait prendre un grand supercalculateur exécutant une simulation scientifique qui offre des performances impressionnantes, mais il ne s'agit pas d'un calcul en temps réel, puisqu'il n y a aucune contrainte temporelle sur ce calcul, le retard ou l'avance des résultats n'a plus ou moins aucune importance. En revanche, le système anti-blocage du freinage d'une voiture avec un microcontrôleur 16 bits de 20 Mhz [5] de puissance a été conçu pour répondre à des délais bien précis, qu'en aucun cas on ne doit dépasser, lui implanter un processeur plus puissant serait du gaspillage et ne serait d'aucune utilité.

1.5.5. Système d'exploitation temps réel (RTOS)

Un système d'exploitation temps réel (RTOS) est un système d'exploitation multi-tâches destiné aux applications temps réel. Ces applications sont généralement des applications pour systèmes embarqués. En réalité, la relation temps réel systèmes embarqués est très forte. Les systèmes embarqués sont souvent soumis à des contraintes temporel temps réel, et un RTOS est rarement conçu pour un usage général, car l'installation de nouveaux softs sur un RTOS ne doit pas être une opération prise à la légère comme dans les OS d'usage générale. Au fait un RTOS facilite la création d'un système temps réel, mais ne garantit pas que le résultat final sera temps réel, une grande partie pour cette garantie réside dans le logiciel installé sur l'RTOS, ce qui exige un développement correct et rigoureux de la part des développeurs. Il faut aussi prendre en considération ce qu'on appelle les paramètres non fonctionnels qui sont généralement liés aux caractéristiques techniques du système, comme par exemple sa fiabilité, sa consommation d'énergie, ou les temps de latence des supports de communications.

Un RTOS n'est pas nécessairement plus rapide qu'un système d'exploitation généraliste, mais un OS temps réel fournit des outils que si ces derniers sont correctement utilisés, des garanties peuvent être posées sur les délais d'exécutions des tâches dans le système. Si ces garanties sont respectées à l'absolu sur tous les délais des tâches, dans ce cas le système est dit temps réel strict (hard real-time) ou si une tolérance vis-à-vis le non-respect strict de tous les délais est permise alors le système est dit temps réel souple (soft real-time). Un RTOS utilise généralement un algorithme d'ordonnancement (scheduler) spécifique aux systèmes temps réel, dans le but de fournir aux développeurs la capacité de produire des applications avec un comportement déterministe et prévisible dans le système final. Contrairement à un OS généraliste, un RTOS est évalué beaucoup plus sur sa fiabilité, son déterminisme et sa réactivité, que par sa rapidité ou sa richesse fonctionnelle. (Les RTOS sont plus approfondis dans le chapitre 3 : *Systèmes d'exploitation temps réel*).

1.5.6. Les craches logiciel dans les systèmes embarqués

La question qui revient le plus souvent aux concepteurs et aux développeurs de systèmes embarqués est de comment savoir si le code est sans bug ? Comment faire pour tester les logiciels complexes qui doivent fonctionner correctement dans toutes les conditions ? Ainsi. Le point le plus important à tirer de ces interrogations, c'est que l'échec d'un logiciel est beaucoup moins tolérable dans un système embarqué que dans la majorité des PCs de bureau.

C'est inadmissible pour les concepteurs et les développeurs d'essayer de produire un code avec 0 % de bug. Dans la majorité des cas la phase de test dans le cycle de développement d'un système embarqué est la phase la plus longue et la plus coûteuse, et elle est généralement relative à la nature critique du produit. Il est évident que cela n'est pas suffisant pour prévenir les craches logiciel pour systèmes embarqués, c'est pour cette raison que ces derniers contiennent en général un mécanisme de réinitialisation ou de secours. On peut citer par exemple les chiens de garde (Watchdog timer), qui permettent de réinitialiser le système si le logiciel perd le contrôle.

1.5.7. Les systèmes embarqués et les contraintes d'énergie

Pour le PC généraliste. Les CPU x86 comme le Pentium ou l'AMD Athlon ont besoin d'un énorme dissipateur thermique pour garder le processeur hors d'atteinte d'une surchauffe. À vrai dire les PCs n'ont pas des contraintes particulièrement dramatiques pour ce qui concerne l'alimentation puisqu'ils sont alimentés par des secteurs de courant électrique standard de 110-220 Volt de maison. On ne peut pas dire autant pour les systèmes embarqués. Ils sont généralement alimentés par des batteries avec une puissance et durée limitée. Ça se vois clairement sur les téléphones portables et les PDA, lorsque ont est obligés de les recharger continuellement. Ce problème se discerne clairement pour les sondes d'explorations autonomes comme par exemple les sondes spatiales. Mise dans un enivrement extrême, elles n'ont que les panneaux solaires comme source d'énergie.

Habituellement, la tâche du respect du cahier des charges sur les contraintes de l'alimentation est laissée aux ingénieurs du matériel. Cependant, la répartition des responsabilités n'est pas bien clairement évidente. Le concepteur du logiciel pourrait ou ne pourrait pas avoir une idée des contraintes architecturales du matériel. En général si le projet du système est soumis à des contraintes de puissance, il est probable que la conception du logiciel doit être construite de telle sorte que le processeur soit en mode veille la plupart du temps et ne se réveiller que quand surviennent les interruptions. En d'autres termes, le système est entièrement axé sur la mis-en-veille et le réveil du processeur.

On remarque dans les lignes qui précèdent que le processeur est souvent mis en avant lors de la conception d'un système soumis à des contraintes d'énergies. C'est tout à fait légitime en sachant que dans la pratique c'est l'élément qui consomme le plus d'énergie dans les systèmes embarqués. Dans le marché des processeurs pour systèmes embarqués et contrairement au marché des PCs, la fréquence d'exécution n'est pas le seul facteur déterminant de qualité pour processeurs, il y en a plusieurs d'autres comme on vient de dire, la consommation, la résistance aux éléments physiques comme la chaleur, les interférences radios, les radiations solaires... etc.

1.5.8. Environnement pour systèmes embarqués

Contrairement aux PCs, un système embarqué doit faire face à des environnements plus hostiles. Il doit faire face à un ensemble de paramètres environnementaux agressifs :

- Variations dans la température.
- Vibrations et chocs physiques.
- Variations dans alimentations.
- Interférences radios.
- Corrosion.
- Eau, feu, radiation.

L'environnement dans lequel opère le système embarqué n'est pas contrôlé ou contrôlable. Cela suppose donc de prendre en compte ce paramètre lors de sa conception. On doit par exemple prendre en compte les évolutions des caractéristiques électriques des

composants en fonction de la température, des radiations, des interférences…etc.

1.5.9. La connectivité des systèmes embarqués

Les systèmes embarqués sont aujourd'hui fortement communicants. Cela est devenu possible grâce à la monté en puissance du calcul offert par les récents processeurs pour systèmes embarqués (32 bits en particulier) et grâce aussi à l'explosion de l'usage de la connectivité Internet ou connectivité IP. La connectivité IP permet fondamentalement de contrôler à distance un système embarqué par Internet. Ce n'est en fait que l'aboutissement du contrôle à distance d'un système électronique par des liaisons de tout type : liaisons RS.232, RS.485, bus de terrain...

Cela permet l'emploi des technologies modernes du web pour ce contrôle à distance par l'utilisateur, il suffit d'embarquer un serveur web dans son équipement électronique pour pouvoir le contrôler ensuite à distance, de n'importe où, à l'aide d'un simple navigateur web. L'IHM à concevoir devient ainsi plus aisé, puisque ce rôle étant en grande partie rempli par le navigateur web.

Il faut aussi noter la montée en puissance des communications sans fil dans l'embarqué au détriment des communications filaires pour limiter le câblage et faciliter la mise en place du système embarqué. Le wifi et toutes les normes de réseaux sans fil IEEE 802.15 comme Zigbee en sont généralement de bons candidats pour être utilisés dans l'embarqué et surtout en domotique (réseaux de capteurs sans fil par exemple). Mais mis à part ces facilités et commodités, cela a bien sûr un revers, la sécurité du système embarqué, en sachant qu'il est connecté sur Internet ou accessible d'une manière qui ne nécessite pas un raccord physique qui est le sans fil. La sécurité des systèmes embarqués est donc cruciale et doit être prise en compte dès leurs phases de conception.

1.5.10. La conception d'un système embarqué

Du point de vue technique, la conception d'un système embarqué demande à son concepteur d'être pluridisciplinaire : électronique, informatique, réseaux, sécurité. Mais le concepteur se doit aussi d'être un bon gestionnaire car concevoir un système embarqué revient finalement à un exercice d'optimisation : minimiser les coûts de production pour des fonctionnalités optimales.

Le système embarqué se doit d'être :

- Robuste.
- Simple. La simplicité est gage de robustesse.
- Fiable.
- Fonctionnel. Le système doit toujours fonctionner correctement. Surtout si la sécurité des personnes est en jeu.

- Tolérant aux fautes.

D'autres contraintes sont aussi à prendre en compte :

- L'encombrement.
- Le poids.
- Le packaging : difficulté de faire cohabiter dans un faible volume l'électronique analogique, l'électronique numérique, et la radio fréquence, sans générer des interférences entre ces compartiments.
- L'environnement extérieur.
- La consommation électrique. Le système embarqué nomade doit être de faible consommation, car il est alimenté par des batteries. Une consommation excessive augmente le prix de revient du système embarqué car il faut alors des batteries de plus forte capacité.
- Le temps de développement. Dans un marché concurrentiel et de niches, il convient d'avoir un système opérationnel le plus rapidement possible pour être le premier sur le marché.

1.6. Conclusion

Ce chapitre sur les généralités d'un système embarqué a pour but de faire un tour absolument non exhaustif sur les différentes facettes que peut avoir ce domaine. Ce qu'on peut retenir de ce chapitre c'est que le domaine des systèmes embarqués est un domaine vaste et pluridisciplinaire, qui a réussi à faire converger plusieurs disciplines a première vue totalement disjoints pour former une technologie des plus utilisée de nos jours. Les principaux domaines qui ont une relation directe ou indirecte avec les systèmes embarqués sont : les systèmes d'exploitation et la programmation système, le génie logiciel, la conception et/ou exploitation d'architecture matériel, l'électronique, buses de communication, réseaux de télécommunication, sécurité informatique, et le temps réel. Au fait le temps réel est omniprésent dans les systèmes embarqués et le fait de dire système temps réel çà inclus plusieurs caractéristiques pour le système embarqué. Le chapitre suivant traite le temps réel et l'influence de ses caractéristiques sur un système embarqué.

Les systèmes embarqués et le temps réel

2

2.1. Introduction

Les systèmes embarqués temps réel sont couramment utilisés dans notre vie quotidienne. Leurs applications vont du simple appareil domestique, tel que les machines à laver, le système du chauffage central ou les barrières automatiques, jusqu'aux systèmes hautement critiques, tels que les systèmes de contrôle dans un avion, le système d'airbag dans une voiture ou les systèmes de monitoring pour les patients dans un hôpital. Ces systèmes se diffèrent par bien des aspects, comme l'utilisation, la taille, et la criticité, mais restent néanmoins tous caractérisés par des contraintes temporelles plus ou moins strictes. Leur développement nécessite des techniques spécialisées. L'analyse des besoins doit prendre en compte l'aspect temporel qui doit être précisé et vérifié. La conception doit faire face à l'architecture du système, la répartition des tâches et des processeurs, l'ordonnancement de tâches…etc. Le tout est soumis à des contraintes en relation avec le temps et les ressources.

Si en prend par exemple le système d'airbag à l'intérieur du volant d'une voiture. Il doit, après la détection d'une collision (et seulement après) se gonfler juste à temps pour attraper doucement la tête du passager pour l'empêcher de heurter le volant. Une précision temporelle dans le système doit être respectée, car il ne doit pas se gonfler trop tôt, ce qui ferait que le coussin se dégonfle avant de pouvoir attraper la tête du passager, ni trop tard, pour que l'explosion de l'airbag ne blesse pas le passager en lui explosant sur le visage ou de le prendre trop tard après avoir pris le choc sur volant. Le système du coussin gonflable est un exemple typique d'un système temps réel, et il montre clairement l'impotence de prendre en compte le temps comme facteur déterminant dans la conception de ce genre de système. Dans ce chapitre on va aborder tout d'abord une perception introductive aux systèmes temps réel, avec quelques définitions comme le comportement temporel des systèmes temps réel, les événements, ainsi que les notions du soft et du hard real-time. Ensuite on se focalisera principalement sur les aspects fondamentaux à en prendre en considération dans la conception des systèmes temps réel, comme l'architecture physique, les modèles d'interaction, et la stratégie d'exécution. Au finale, le paradigme de conception Component-Based Design sera évoqué comme un modèle de conception prometteur pour la recherche et l'industrie.

2.2. Définition

Il existe une définition sur le temps réel [6] selon laquelle "un système temps réel, n'est pas tel, si des données sont présentées au système, lui permettant de calculer pour une période de temps indéterminée, puis de fournir les résultats en sortie. Les résultats en sortie doivent absolument être disponibles en temps opportun, et le processeur (ou les processeurs) doit être capable de choisir avec soin les tâches spécialement ordonnancées pour que les délais en temps soit respectés." Ou la définition [7] "un système temps réel est un système dans lequel les performances ne dépendent pas seulement de l'exactitude des résultats, mais aussi du temps dans lesquels ils sont fournis.". Il n'existe pas de définition commune pour tous les chercheurs pour un système temps réel, à cause de la diversité des domaines dans lesquels le temps réel est impliqué, quoique les qualifications suivantes soient communément acceptées :

- Un système temps réel est un système informatique qui interagit physiquement avec le monde réel.
- Ces interactions avec le monde réel doivent obéir à des contraintes temporelles.

En règle générale, les interactions d'un système temps réel sur le monde réel s'effectuent via des capteurs et des actionneurs, plutôt qu'un clavier et un écran d'ordinateur standard. Et les contraintes temporelles généralement exigent que les interactions s'accomplissent sur des intervalles temporels prédéfinis. Il convient de noter que cela est tout à fait différent quand l'interaction s'exécute le plus rapidement possible.

2.3. Le hard real-time et le soft real-time

Un comportement temporel strict typique peut être considéré comme une réponse à une interaction qui doit toujours se produire à l'intérieur d'une certaine limite temporelle prédéfinie. Par exemple, la charge d'air dans l'airbag de la voiture doit exploser entre 10 à 20 ms après la détection d'une collision, violer ces limites doit être évité à tout prix, car ça aboutirait à la mort ou à une blessure du passager, ce qui est totalement inacceptable. Un système qui est conçu pour répondre aux strictes exigences en temps est souvent évoqué comme un système temps réel dur (hard real-time).

En revanche, les systèmes dits temps réel souple (soft real-time) ont une spécificité pour laquelle une violation occasionnelle des délais temporels est acceptable. Comme par exemple les systèmes d'affichage vidéo. Si quelques images ne sont pas affichées, cela ne met pas en péril le fonctionnement correct de l'ensemble du système, et cela ne va en aucun cas mettre en danger les usagers. La *figure*2-1 illustre le comportement dans le temps des deux types de système. On peut apercevoir sur cette figure l'arrivée des événements aux systèmes ainsi que leurs délais de repense respectifs. Tandis que le *Tableau*2 donne quelques caractéristiques communément trouvées dans les deux systèmes.

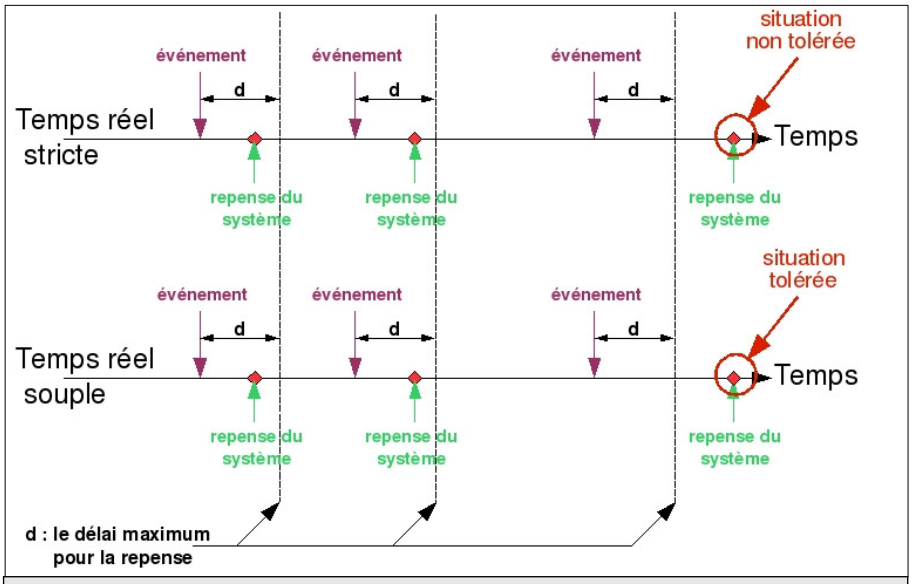

Figure 2-1 *: Comparaison temps réel strict et temps réel souple.*

Caractéristiques	Temps réel strict	Temps réel souple
Contraintes temporelles	Strict	Souple
Environnement	Monde externe	Ordinateurs
Performance étant surchargé	Prévisible	Dégradée
Criticité	Critique	Non critique
Redondance matérielle	Répondue	Rare
Granularité temporelle	micro/milli-seconde	milli-seconde/seconde
Granularité des données	Petite	Grande
Taille des fichiers	Petite	Grande

Tableau 2 : *Caractéristiques de systèmes temps réel strict et temps réel souple.*

2.4. Types d'événement dans les systèmes temps réel

Les événements qui sont en mesure d'être traités par un système temps réel peuvent être classés en trois catégories : les événements *périodiques*, les événements *apériodiques* et les événements *sporadiques*. Les événements périodiques s'affluent selon un laps de temps constant (appelé période). Tandis que les événements apériodiques ne possèdent pas de période, ainsi ils ne suivent aucune règle dans le temps et arrivent d'une façon totalement aléatoire. De même, les événements sporadiques ne suivent aucune période dans le temps, mais contrairement aux événements apériodiques, lors de l'avènement d'un événement, un laps de temps minimal doit s'écouler pour que l'événement suivant puisse se manifester. La

*figure*2-2 donne une représentation de l'afflux des événements dans les trois classes.

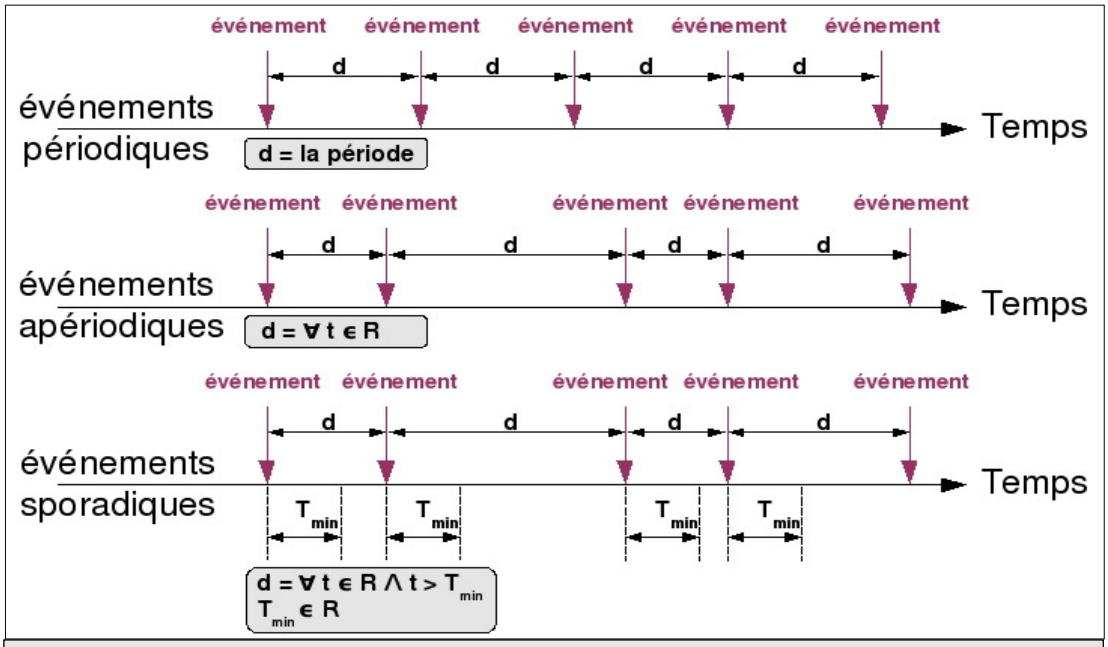

Figure 2-2 : *Illustration du comportement temporel des trois classes d'événement.*

Dans le cas général, les événements en relation avec les mesures dans le monde externe sont périodiques, car ils ont pour rôle de collecter des mesures à chaque unité de temps (ou période). Le reste est apériodique ou sporadique. En sachant que les événements sporadiques se distinguent des événements apériodiques quand un intervalle minimal dans le temps entre deux événements est bien connu lors de la phase d'analyse, et c'est souvent grâce aux connaissances sur les limitations physiques du système. L'exemple typique pour ce genre d'événements sont les événements déclenchés par des capteurs étalés sur des rails transportant un chariot pour connaître sa position, étant donné que le chariot ne peut pas excéder une vitesse maximum, un temps minimum est garanti entre le déclenchement de deux événements pour deux capteurs qui se succèdent. Si le temps minimal entre deux événements est a priori inconnu, l'événement est attribué à la classe des événements apériodiques.

2.5. Conception des systèmes temps réel

Le principal problème dans la conception des systèmes temps réel est l'aspect du *temps,* En sachant que le système doit toujours effectuer ses opérations aux bons moments. Ne pas tenir compte de cet aspect à la phase de conception, rend cette dernière pratiquement impossible à analyser et à prédire le comportement du système dans le temps. Cette section présente quelques-unes des considérations importantes à prendre en compte pour la conception d'une architecture embarquée temps réel, avec à la fin un survol sur quelques outils commerciaux repandus dans les milieux de la recherche et de l'industrie.

2.5.1. Architecture physique

L'architecture physique comme son nom l'indique définie l'architecture matérielle du système, prise en charge généralement par un groupe de concepteurs matériel. Contrairement à la conception logicielle qui est elle prise par un tout autre groupe de concepteurs logiciel. Pour une bonne conception du système, les deux groupes doivent être en relation constante l'un avec l'autre tout au long du cycle de développement, en raison de prendre des décisions communes sur les différents points qui impliquent les deux parties. L'architecture physique a pour rôle de définir les composants matériels, comme les unités de calcul, les mémoires, les supports de communication, les capteurs... et leurs distributions spatiales.

Une architecture générique pour un système temps réel est décrite dans la *figure*2-3. Cette architecture est un modèle de tout système fondé sur l'interaction avec un environnement extérieur par l'intermédiaire de capteurs et d'actionneurs.

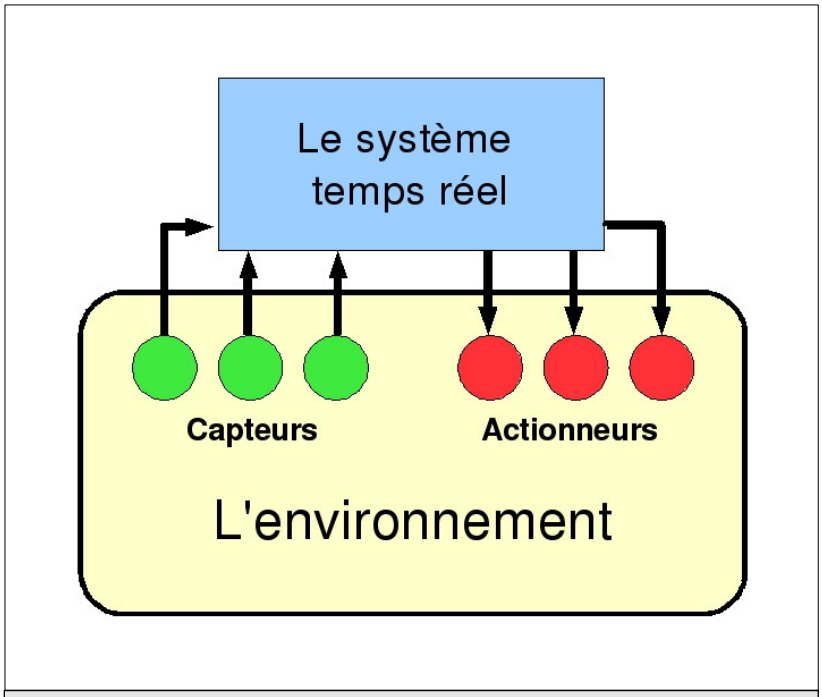

Figure 2-3 *: Architecture générique d'un système temps réel.*

Le système temps réel le plus simpliste est un système avec un processeur unique, mais dans de nombreux cas, les systèmes temps réel sont des systèmes informatiques distribués composés d'un ensemble de processeurs interconnectés par un réseau de communication. Plusieurs raisons peuvent pousser les concepteurs à construire un système temps réel distribué, parmi ceux-ci :

- La distribution physique de l'application.
- Les exigences de calcul qui peuvent ne pas être facilement assuré par un seul processeur.
- Le besoin de redondance pour répondre à la disponibilité, la fiabilité, ou d'autres exigences en matière de sécurité.

- Afin de réduire le câblage du système.

Le dernier point ne paraît pas aussi explicite que ça, mais si on considère l'architecture générale du système vue précédemment, les systèmes temps réel sont habituellement connectés aux capteurs et aux actionneurs qui sont souvent éparpillés physiquement à différents endroits dans l'environnement. Un moyen efficace de réduire la quantité du câblage est de regrouper les nœuds les plus proches sur un nœud de ralliement qui fera office d'un nœud de communication avec le processeur central.

La *figure*2-4 [8] montre un exemple d'un système temps réel distribué dans une voiture moderne. Dans une voiture il existe environ 20 à 100 nœuds de ce genre. Ce qui dans l'industrie automobile est appelé *unité de contrôle électronique* (ECU : Electronic Control Unit) interconnectés avec un ou plusieurs réseaux de communication. La première motivation pour une architecture distribuée dans les voitures était la nécessité de réduire la quantité de câblages comme signalé auparavant. Toutefois, cette architecture a également conduit à d'autres améliorations importantes, y compris une réduction de la pollution avec l'abaissement du nombre des compartiments mécano-hydrauliques et l'ajout de nouveaux mécanismes de sécurité dans la voiture, comme les ESP (Electronic Stabilization Programs : ce sont des mécanismes anti-dérapage dans une voiture). La recherche actuelle dans le domaine tente de rendre les compartiments les plus sensibles en termes de sécurité dans la voiture, comme la direction et le freinage totalement contrôlable par ordinateurs. Ceci en enlevant les connexions mécaniques (par exemple, entre le volant et les roues avant, et entre la pédale et les plaques de freinage). Et les remplacer par des systèmes informatiques distribués.

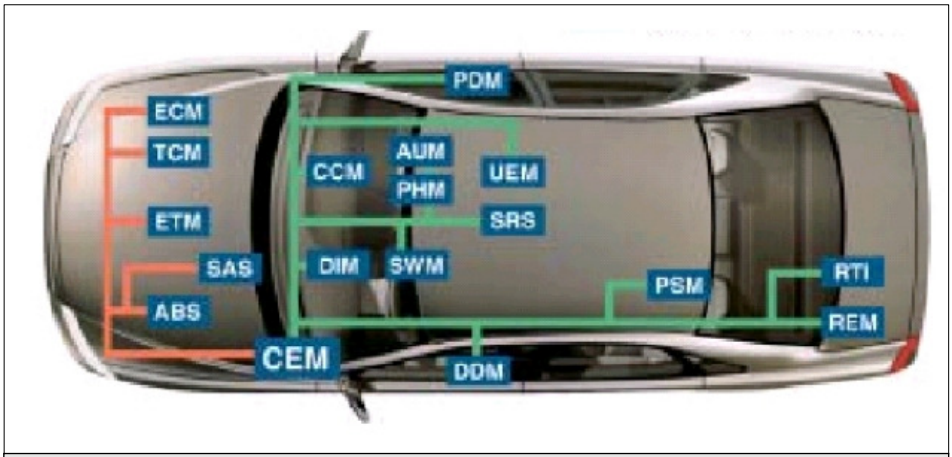

Figure 2-4 *: Architecture d'un système temps réel distribué dans une voiture moderne.*

2.5.2. Modèles d'interaction

Dans la section précédente, nous avons présenté l'architecture physique d'un système temps réel, mais pour le groupe des concepteurs logiciel, ce n'est pas vraiment l'aspect le plus important. Au fait, du point de vue du concepteur de l'application une vue de l'architecture du système est donné par son *paradigme d'exécution* (ou *stratégie d'exécution*) et du *modèle d'interaction* utilisé dans le système. Dans cette section, nous allons décrire ce qu'est un modèle d'interaction et de la façon dont il affecte les propriétés en temps réel du système. Après, nous discuterons les stratégies d'exécution dans les systèmes temps réel.

Un modèle d'interaction décrit les règles par lesquelles les composants interagissent les uns avec les autres (dans cette section, nous allons utiliser le terme *composant* pour désigner une entité logicielle cohérente, comme par exemple une tâche ou un module). Le modèle d'interaction régit bien les flux de contrôles que les flux de données entre les composants du système. Pour la conception logicielle, le modèle d'interaction compte parmi les plus importantes décisions que les concepteurs logiciels doivent prendre. Malheureusement cette décision est souvent prise implicitement, du fait que le modèle d'interaction est souvent implémenté par le système d'exploitation ou le middleware choisi pour le système.

Lors de la conception d'un système temps réel, l'attention devrait être accordée à propriétés temporelles du modèle d'interaction choisie. Certains modèles ont un comportement temporel plus prévisible et plus robuste que d'autres. On peut citer par exemple quelques modèles des plus prévisibles et les plus utilisés dans la conception des systèmes temps réel comme le modèle *tuyaux et filtres*, le modèle *éditeur-abonné*, et le modèle *tableau noir*.

D'un autre côté, il existe des modèles d'interaction qui augmentent l'imprécision temporelle du système. En connaissance de cause, ces modèles doivent être pris avec précaution, et si possible, être évités lors de la conception d'un système temps réel. Les deux plus importants, et communément utilisés sont le modèle *clients-serveur* et le modèle *boîte aux lettres.*

Le modèle tuyaux et filtres : Dans ce modèle, les flux de donnée et les flux de contrôle sont spécifiés en utilisant les ports d'entrée et de sortie des composants. Un composant devient apte pour l'exécution que lorsque les données sont arrivées sur ses ports d'entrées et il termine son exécution par produire ces flux sur ses ports de sortie.

Ce modèle convient très bien pour de nombreux types de systèmes de contrôle, car les règles de contrôle sont facilement implémentable sur ce type modèle. Ainsi, il a largement été utilisé dans la conception temps réel. Le modèle présente de bons caractéristiques temps réel, en sachant que les deux flux de donnée et de contrôle d'une façon unidirectionnelle traversent une série de composants, l'ordre d'exécution et les délais de bout en bout sont facilement calculables, ainsi le modèle devient facilement prévisible dans le temps.

Le modèle éditeur-abonné : Se base sur le principe qu'un éditeur produit des publications sous forme de valeurs de données ou de contrôles, que les abonnés sont disposés a les utiliser. Le modèle éditeur-abonné est plus ou moins similaire au modèle tuyaux et filtres, quoique généralement le modèle éditeur-abonné dissocie le flux de donnée du flux de contrôle. Aussi, un abonné a généralement l'opportunité de choisir délibérément le déclenchement de son exécution parmi les différentes publications de son éditeur. Si on prend le cas où l'abonné choisit de se déclencher à chaque nouvelle publication, le modèle prend exactement la forme du modèle tuyaux et filtres. En plus

de choisir la publication, un abonné a le droit d'ignorer la date de la publication et de prendre que la dernière valeur publiée. Aussi, pour le modèle éditeur-abonné, les éditeurs ne sont pas nécessairement conscients de l'identité, ni même de l'existence de leurs abonnés. Cela donne une plus grande abstraction dans la conception du système temps réel.

Comme pour le modèle tuyaux et filtres, le modèle éditeur-abonné offre de bonnes propriétés temporelles. Toutefois, une condition préalable pour pouvoir faire l'analyse des systèmes utilisant ce modèle, est que les composants abonnés doivent indiquer d'une façon explicite aux éditeurs les valeurs auxquelles ils sont abonnés (ce qui n'est pas imposé par le modèle lui-même), cette information sert par exemple dans un système embarqué de décider des valeurs qui doivent être publiées sur son réseau de communication, et les nœuds qui doivent réceptionner ces valeurs.

Le modèle tableau noir : Le modèle du tableau noir permet à des variables globales d'être publiées sur une zone mémoire accessible pour tous les composants (Le tableau noir fait référence à la métaphore d'un tableau d'une classe d'étude où tout élève peut écrire dessus). Ainsi, Le modèle permet à tout composant de lire ou d'écrire dans les valeurs des variables sur le tableau. De toute évidence l'utilisation de ce modèle dans la conception temps réel reste discutable, néanmoins c'est un modèle couramment utilisé, car dans certains cas, il fournit des solutions pragmatiques aux problèmes qui sont difficilement abordables avec les modèles d'interaction cités précédemment.

Le modèle client-serveur : Dans le modèle client-serveur, un client invoque les services d'un serveur d'une manière asynchrone. Lors de l'invocation du service par le client un flux de contrôle (en plus d'un flux de donnée) est passé au serveur, et le contrôle reste dans le serveur jusqu'à ce qu'il termine son exécution. Entre temps le client reste bloqué jusqu'à ce que le serveur termine en lui renvoyant le flux de contrôle (et le flux de donnée). Ainsi le client peut continuer son exécution.

Le modèle client-serveur est intrinsèquement imprévisible dans le temps, étant donné que les services sont invoqués de façon asynchrone, il est très difficile d'évaluer a priori la charge sur le serveur pour un service donné. Ainsi, il est difficile d'estimer le retard de l'invocation du service et, à son tour, il est difficile d'estimer le temps de réponse du client. Cette situation est encore plus compliquée par le fait que la plupart des composants se comportent souvent en tant que clients et serveurs (un serveur utilise souvent d'autres serveurs pour mettre en œuvre ses propres services). De ce fait le chemin du flux de contrôle et l'analyse temporelle du système sont très complexes à calculer.

Le modèle boîte aux lettres : Un composant peut avoir un ensemble de boîtes aux lettres, et les composants communiquent les uns avec les autres en envoyant des messages dans les boîtes aux lettres des autres composants. Les messages sont généralement traités selon la stratégie premier arrivé, premier servi (FIFO), ou par ordre de priorité (si l'expéditeur spécifie une priorité). Faire passer un message pour un expéditeur ne change pas son le flux de contrôle. Tandis qu'un composant qui tente de

recevoir un message qui n'est pas encore délivré par l'expéditeur peut se bloquer jusqu'à ce que le message arrive (souvent, le récepteur spécifie un délai pour éviter le blocage pour une durée indéterminée).

Du point de vue de l'expéditeur, le modèle boîte aux lettres a les mêmes problèmes que le modèle client-serveur. Les données envoyées par l'expéditeur (et l'action que l'expéditeur s'attend du récepteur à effectuer) peuvent être retardées de façon imprévisible lorsque le récepteur est trop chargé. Aussi, en ce qui concerne la nature asynchrone du passage des messages, il est difficile de prévoir la charge d'un récepteur à un moment donné.

De plus, à partir du point de vue récepteur, la lecture des boîtes de message est imprévisible dans le sens que le récepteur peut ou ne peut pas se bloquer sur la boîte de message (selon la présence ou non d'un message). Sans oublier que les boîtes aux lettres sont de taille limitée, il y a toujours un risque qu'un récepteur surchargé de messages perd des messages (donc une autre source d'imprévisibilité).

2.5.3. Stratégie d'exécution

Il existe deux principaux paradigmes d'exécution pour les systèmes temps réel : le *paradigme temporel* et le *paradigme événementiel*. Dans le premier, les activités se produisent à des périodes de temps prédéfini, ça se présente généralement comme un capteur qui recueille périodiquement des informations de l'environnement extérieur (toutes les 10 ms par exemple), qui sont lues par le système, et 2 ms plus tard, l'actionneur correspondant reçoit une mise à jour sur son paramètre d'activité, ce qui colle exactement à la classe des événements périodiques. Par contre dans le paradigme événementiel, comme son nom l'indique, les paramètres transmis du système aux actionneurs sont déclenchés par occurrence d'événement, et dans la plupart des cas, ce sont des événements apériodiques ou sporadiques. On peut l'illustrer par l'exemple suivant : lorsque la température d'un moteur atteint un certain seuil, le capteur produit un événement pour le signaler au système.

Il convient de noter que les mêmes fonctionnalités, en général, peuvent être mises en œuvre par les deux paradigmes. Par exemple, pour implémenter un paradigme temporel par un paradigme événementiel, il suffit d'acquérir les événements périodiques par le système événementiel. Ou le contraire, pour faire implémenter le paradigme événementiel par un paradigme temporel, le capteur des événements sera lu périodiquement pour voir si le niveau de mesure a dépassé le maximum autorisé. Quoique dans ce cas, si les alarmes se font rare, cette implémentation aura une sur-consommation des ressources de calcul. Malgré ce fait elle présente un avantage considérable pour les systèmes hautement fiables, puisque a chaque lecture du capteur, le système a l'opportunité de vérifier l'état fonctionnel de ce dernier, en d'autres mots, il vérifie si le capteur est toujours en bon état et qui ne présente aucun signe d'anomalie.

Les paradigmes d'exécution temporelle sont utilisés dans de nombreux systèmes embarqués critiques avec de hautes exigences de fiabilité (comme les systèmes de contrôle

avionique), alors que la majorité des autres systèmes utilisent le paradigme événementiel. La fiabilité peut être aussi garantie par les paradigmes événementiels, mais en raison de l'observabilité fournie par les paradigmes temporels (la vérification des capteurs expliquée auparavant), la plupart des experts plaident pour l'utilisation de ce dernier dans les systèmes a haute exigence de fiabilité [8]. Quoique son principal point faible soit son manque de souplesse et le surplus en consommation du processeur par rapport au paradigme événementiel.

Le paradigme temporel est souvent mis en œuvre la plupart du temps par de simples systèmes d'exploitation fait maison implémentant le scheduling dit *table-driven* [9]. Quoique des produits commerciaux complets, y compris des outils de conception sont également disponibles [10]. Pour le paradigme événementiel un grand nombre d'outils commerciaux et de systèmes d'exploitation sont disponibles (des exemples sont donnés dans le chapitre suivant : *Systèmes d'exploitation temps réel*). Il existe également des exemples de systèmes intégrant les deux paradigmes d'exécution, en vue d'obtenir le meilleur des deux mondes : le temporel pour sa fiabilité et l'événementiel pour sa flexibilité. Un exemple de ce genre de systèmes est le noyau temps réel Rubus [11].

2.5.4. Conception basée composant

La conception basée composant ou *Component-Based Design* (CDB) est une approche intéressante pour le génie logiciel en général, et pour l'ingénierie des systèmes temps réel en particulier [12]. Dans le Component-Based Design, un composant logiciel est utilisé pour encapsuler les fonctionnalités, et une fonctionnalité est uniquement accessible via l'interface du composant. Un système est composé par le regroupement d'un ensemble de composantes et des connexions de leurs interfaces.

La raison qui pourrait exposer l'utilité de la conception Component-Based Design pour les systèmes temps réel est la possibilité d'étendre les composants avec des interfaces d'introspection. Une interface d'introspection ne fournit pas de fonctionnalité en soi, mais cette interface peut être très utile pour la récupération d'informations sur les propriétés extra-fonctionnelles d'un composant. Extra-fonctionnelle veut dire les attributs caractéristiques du composant tels que les attributs qui fournissent la consommation de mémoire, le temps d'exécution, la période d'exécution d'une tâche…etc. Ces informations sont d'une extrême importance pour un système temps réel.

À la différence des interfaces fonctionnelles des composants, les interfaces d'introspection peuvent être disponibles hors ligne, ça veut dire au cours de la phase d'assemblage des composants. De cette façon, les caractéristiques temporelles des composants du système sont obtenues au moment de la conception et des outils pour analyser le comportement temporel du système pourraient être utilisé. Si les interfaces d'introspection sont également disponibles en ligne (état d'exécution), ils pourraient être utilisés par exemple avec des algorithmes de contrôle d'admission. Un algorithme de contrôle d'admission peut interroger un nouveau composant en se basant sur ces caractéristiques temporelles et son comportement vis-à-vis de sa consommation en ressources avant de décider de l'accepter ou non dans le système.

Malheureusement, de nombreux standards logiciels de l'industrie sont basés sur le modèle d'interaction client-serveur ou du modèle boîte aux lettres, que tout les deux présentent des lacunes en se qui concerne la conception pour systèmes temps réel. Cela est particulièrement vrai pour la plupart des modèles a base de composants existant de nos jours. On peut prendre comme exemple, le CORBA Component Model (CCM) [13], Microsoft COM [14], Microsoft DotNET [15], et Java Beans [16]. Tous sont à base de modèle d'interaction client-serveur. Aussi, aucune de ces technologies ne prennent en charge l'utilisation des propriétés extra-fonctionnelles a travers des interfaces introspectives. Par conséquent, ces technologies ont un intérêt minime en ce qui concerne la conception temps réel.

Toutefois, il existe de nombreux projets de recherche traitant de la conception basée composant temps réel et systèmes embarqués (comme par exemple, [17-19]). Ces projets portent sur les points délaissés par les technologies commerciales existantes, comme la prévisibilité dans le temps, et le support pour les systèmes aux ressources limitées. Souvent, ces projets visent à éliminer la grande flexibilité d'exécution prévue par les technologies existantes. Cette flexibilité tente de cacher les détails de l'implémentation au programmeur et de lui faciliter la création et la communication des composants. Toutefois, elle est jugée comme le premier contributeur à l'imprévisibilité. Cette flexibilité ajoute également un degré de complexité dans l'exécution qui n'est pas commode dans les systèmes aux ressources limitées.

2.5.5. Outils pour la conception temps réel

L'un des outils les plus utilisé pour la conception de logiciels de nos jours est l'UML. Toutefois, UML est centré principalement sur les solutions client-serveur (puisqu'il est à base d'objets, et que le principal moyen de communication des objets est l'invocation de méthodes). Ce qui le rend peu sollicité pour la conception des systèmes temps réel. Par conséquent, des outils basés sur l'UML étendu pour inclure la conception temps réel ont vu le jour. Les deux plus connus sont des produits d'IBM: Rational Rose RealTime [20], et Telelogic Rhapsody [21]. Ces outils fournissent le support d'UML, avec des extensions pour le temps réel, tout en donnant aux concepteurs des modèles d'abstraction et de calcule. Quoique ces outils ne fournissent pas les méthodes nécessaires de décrire les propriétés et les besoins temporels d'une manière formelle. Un manque aussi ce fait sentir sur les outils de vérification automatique en ce qui concerne le comportement temporel.

Telelogic d'IBM offre aussi des outils de support à la conception et à la programmation temps réel sous l'appellation de TAU [22], ces outils sont principalement basés sur le langage SDL. SDL a été développé à l'origine comme un langage de spécification pour l'industrie de la télécommunication. Ainsi en tant que tel, il est très approprié pour décrire les systèmes réactifs complexes. Toutefois, ses modèles de calcul sont centrés sur le modèle d'interaction boîte aux lettres, qui a, de nature, un comportement imprévisible dans le temps. Cependant, pour les logiciels temps réels embarqués, SDL peut fournir une mise en œuvre relativement efficace en matière de temps et d'espace mémoire.

Pour les systèmes embarqués temps réel aux ressources limitées, plusieurs outils de

conception sont fournis par des firmes comme Arcticus System [11], TTTech [10], et Vector [23]. Ces outils sont essentiellement utilisés au cours des deux phases : Conception et mise en œuvre, ils fournissent aussi des techniques d'analyse dans le temps qui permettent de faire une vérification temporelle du système (ou d'une partie du système). Toutefois, ces outils sont basés sur des formats propriétaires fermés, sans oublier qu'ils visent un nombre restreint de concepteurs (principalement ceux de l'industrie automobile). Pour le secteur de l'automobile il existe aussi The Architecture Description Language (EAST-ADL2) [24], c'est un langage de description qui couvre l'ensemble du cycle de développement de la production en prenant en compte la limitation des ressources et l'aspect de sécurité.

2.6. Conclusion

Dans ce chapitre nous avons présenté l'un des aspects les plus importants dans le domaine des systèmes embarqués. Au fait, le temps réel prend une grosse part dans les systèmes informatiques dédiés à un usage non généraliste, car il est souvent considéré comme un facteur garant de la fiabilité temporelle pour ces derniers. Dans ses premières parties, ce chapitre donne une perception introductive au système temps réel, avec quelques définitions sur les propos les plus importants, comme le comportement temporel des systèmes temps réel, les événements, ainsi que le soft et le hard real-time. Le reste se focalise principalement sur les aspects fondamentaux à en prendre en considération dans la conception des systèmes temps réel, comme l'architecture physique, les modèles d'interaction, et la stratégie d'exécution. Au final, le paradigme de conception Component-Based Design a été évoqué comme un modèle de conception promoteur, ainsi que quelques outils et supports appartenant à la recherche ou au monde de l'industrie dans le but de rendre la conception de systèmes temps réel plus robuste et plus rapide possible. La conception des systèmes temps réel ce caractérise par une complexité accrue, a cause des différents aspects qu'elle doit prendre en considération, comme par exemple le matériel, les médias de communication, le système d'exploitation, les applications...etc. Le chapitre suivant traitera l'un de ces aspects qui est le système d'exploitation temps réel.

Systèmes d'exploitation temps réel 3

3.1. Introduction

Pour qu'un système embarqué soit capable d'accomplir ses fonctionnalités, des programmes informatiques doivent être installés sur le matériel du système. L'un de ces programmes, qui est le système d'exploitation, a pour le rôle d'effectuer un certain nombre d'opérations logistiques afin d'assurer l'interactivité des autres programmes entre eux-mêmes et avec les ressources matérielles du système comme le processeur, la mémoire, et les périphériques d'entrées/sorties. La notion de temps réel est abondante dans le domaine des systèmes embarqués. De ce fait, le système d'exploitation dit système d'exploitation temps réel, doit être capable de gérer les programmes et le matériel de telle sorte que le système soit capable de respecter les contraintes temporelles du temps réel. Un système d'exploitation temps réel, en général, fournit au concepteur les outils temps réel nécessaire pour la gestion des processus, la gestion des ressources, la communication inter-processus, la gestion du système de fichier....etc.

Du fait de la diversité des systèmes embarqués, le nombre et les spécificités des systèmes d'exploitation temps réel sont énormes. Dans ce chapitre, nous allons faire un détour sur les principaux caractéristiques des systèmes d'exploitation temps réel, ainsi qu'une analyse sur les principaux outils qui y régissent, comme le Scheduler et les mécanismes de partage de ressources. À la fin du chapitre, nous aurons une perception sur quelques systèmes d'exploitation temps réel actuel, ainsi qu'une discussion sur les normes les plus influentes sur domaine.

3.2. Définition

Un système d'exploitation temps réel (RTOS : Real Time Operating System) est un système d'exploitation multi-tâche destiné aux applications temps réel. Il fournit le support de base pour la planification des tâches (scheduling), la gestion des ressources, la synchronisation et communication inter-processus, la gestion des entrées/sorties...etc. La différence avec un système d'exploitation non temps réel, c'est qu'il fournit en plus des services similaires à ce dernier, des services adaptés pour le développement des application temps réel et un support pour le portage de l'ensemble système/applications sur le matériel du systèmes embarqués.

3.3. Propriétés typiques des systèmes d'exploitation temps réel

Un RTOS est un système d'exploitation multi-tâche et preemptible, il doit gérer la notion de priorité et avoir un support pour la synchronisation prévisible dans le temps, l'héritage de priorités doit être soutenu, et le comportement temporel du système d'exploitation doit être connu. Cela signifie que la latence des interruptions, le pire temps d'exécution (ou le WCET : Worst-Case Execution Time) des appels système, et le temps maximum pour lequel les interruptions sont masquées doivent être connue. Un RTOS est généralement caractérisé comme.

Convenable pour les environnements avec des ressources limitées. En généralement un RTOS fonctionnent dans ces genres d'environnements. La plupart des RTOS ont une faculté à être costomisé à volonté avant la mise en marche (ça se fait généralement au moment de la compilation) pour permettre au développeur d'inclure seulement les parties du système dont il a besoin. Il convient aussi de noter que les RTOS ont tendance à stocker leurs fichiers de configuration en ROM. Cela se fait principalement pour deux raisons : la première est pour minimiser l'utilisation de la mémoire vif et la deuxième pour minimiser le risque que ces fichiers critiques soient altérés par des anomalies provenant des applications installées.

Donner l'accès facile au programmeur pour la manipulation du matériel. Ça inclus aussi bien les interruptions que les périphériques. Le plus souvent, un RTOS donne les services nécessaires au programmeur pour installer des routines d'interruptions lors de la phase de compilation et/ou en cours d'exécution. Ce qui veut dire que le RTOS confie toute la gestion des interruptions au programmeur, permettant ainsi une gestion plus rapide, efficace et prévisible de ces derniers. En ce qui concerne la gestion des périphériques, dans un système d'exploitation non temps réel, l'accès mémoire aux périphériques est généralement protégé par l'MMU (Memory Management Unit) du processeur (en sachant que l'accès aux périphériques est similaire à l'accès mémoire, la différentiation ce présente seulement sur les plages d'adresses), obligeant ainsi l'application à passer par le système d'exploitation. Dans les RTOS, de telles protections sont inhibées laissant les mains libres à l'application de manipuler directement les périphériques. Cela donne plus d'efficacité et plus de rapidité à l'accès aux périphériques au détriment d'une augmentation du risque d'une mal utilisation de ces derniers.

Fournir des services permettant la gestion temporelle du code. Généralement, un RTOS fournit de nombreux mécanismes pour le contrôle temporel entre les différents processus dans le système. Plus précisément, un RTOS dispose d'un Scheduler de processus temps réel, dont la principale fonctionnalité est de faire en sorte que les processus s'exécutent dans la façon que le programmeur s'attend d'eux. Un RTOS prévoit également des mécanismes de contrôle d'accès aux ressources partagées prévisibles dans le temps. Ils utilisent généralement des stratégies, comme par exemple l'ordre de priorité au lieu de la stratégie FIFO généralement utilisée dans les systèmes d'exploitation non temps réel. Habituellement, un RTOS dispose d'un ou plusieurs protocoles de verrouillage de ressources temps réel, tels que le PCP ou IIP qui seront tous les deux détaillés dans les sections qui viennent.

Adéquat au processus de développement de systèmes embarqués. Les RTOS sont habituellement construits dans un environnement différent de celui où ils vont être implantés. Cet environnement est généralement appelé *hôte*, tandis que l'autre est appelé *cible*. Et tout le processus de développement est appelé *cross-développement*. C'est de coutume dans ce genre de développement que la totalité de l'image mémoire cible soit créée en incluant le RTOS et les applications sur la plate-forme hôte pour ensuite être uplaoder sur la plate-forme cible. Par conséquent, la plupart des RTOS sont livrés sous forme de code source ou de modules de bibliothèques précompilés qui sont statiquement liées aux applications dans la phase compilation.

L'une des fonctions les plus importantes d'un RTOS est d'arbitrer l'accès à des ressources partagées de telle sorte que le comportement du système dans le temps reste prévisible. Les deux ressources les plus évidentes qu'un RTOS peut gérer l'accès sont :

- Le processeur - le Schudeler du RTOS doit permettre l'exécution des processus sur le processeur d'une façon prévisible dans le temps.

- La mémoire et les périphériques E/S – le RTOS doit résoudre les conflits entre processus pour une zone mémoire ou un périphérique d'une manière qui laisse le comportement du système prévisible dans le temps.

Les deux sections suivantes vont s'étaler sur la gestion de ces deux ressources :

3.4. Le scheduling temps réel

Dans un système temps réel l'accès au processeur obéit à une politique d'ordonnancement (ou scheduling) temps réel. Le Scheduler peut baser sa politique d'ordonnancement sur quelques attributs relatifs aux tâches comme la priorité, les délais, ou le taux d'exécution. Certaines de ces politiques utilisent directement le temps comme attribut (comme l'attribut du délai) d'une tâche pour entreprendre les décisions de planification, tandis que d'autres utilisent des attributs, qui indirectement influent sur le timing des tâches (comme la priorité ou le taux d'exécution). Ainsi le temps reste l'aspect le plus important dans une politique de scheduling temps réel. C'est parce que c'est le plus sûr moyen de fournir une

analyse apriori sur le comportement temporel du système. Le plus souvent dans la planification des systèmes d'exploitation non temps réel, l'accent est mit sur des attributs tels que l'équité, le débit d'exécution, et la garantie du progrès. Cependant ces attitudes sont souvent en conflit avec les exigences de prévisibilité temporelle d'un RTOS.

3.4.1. Caractéristiques de tâches

Un système temps réel est composé d'un ensemble de programmes temps réel, chaque programme à son tour se compose en un ensemble de tâches. Ces tâches sont des séquences de morceaux de code qui s'exécutent sur une plate-forme avec des ressources limitées. Les tâches se différentient par leurs propriétés temporelles, comme par exemple le temps d'exécution, les délais d'exécution (par exemple une tâche doit être exécutée et terminée 10 ms après l'arrivée de son événement déclencheur) et les périodes d'exécution (par exemple une tâche doit être exécutée tous les 10 ms).

Un système temps réel peut être preemptif ou non preemptif. Dans un système preemptif les tâches peuvent s'interrompent les unes les autres, laissant la tâche avec la plus grande priorité s'exécuter. Dans un système non preemptif une tâche qui a été autorisée à s'exécuter ne peut plus être interrompue jusqu'à son achèvement. Les tâches peuvent aussi être classées selon le type d'événement déclencheur, que ce soit périodique, apériodique ou sporadique. Les tâches périodiques s'exécutent tous les laps de temps (période). Les tâches apériodiques s'exécutent d'une façon totalement aléatoire dans le temps, et les tâches sporadiques s'exécutent d'une façon similaire aux tâches apériodiques, quoiqu'un minimum de temps doit perdurer entre l'exécution de deux tâches.

Un Scheduler temps réel ordonnance les tâches pour se partager le processeur, avec le but de s'assurer que les exigences temporelles du système soient remplies. Le Scheduler décide sur la base des attribues temporels des tâches, laquelle doit être exécutée (ou utiliser le processeur). D'une manière générale il existe deux manières pour lesquelles le Scheduler peut prendre ses décisions, une *offline* (ou statique) dans laquelle les décisions sont établies avant le lancement du système, et l'autre *online* (ou dynamique) où les décisions sont prises au cours de l'exécution du système.

3.4.2. Les Schedulers offline

Le Scheduler offline (ou Table-driven Scheduler [25]) fonctionne comme suite : le Scheduler doit créer un calendrier (une table) avant la mise en marche du système (dans la phase de compilation). Ainsi, au moment de l'exécution, le dispatcher fait en sorte que les tâches ne soient exécutées qu'une fois entrées dans leurs intervalles de temps prédéterminés par le calendrier. Il est de coutume de voir les Schedulers offline utilisés pour la mise en œuvre de paradigme d'exécution temporel (décrits dans le chapitre précédent), étant donné que ce dernier se base sur l'utilisation de tâches périodiques. Un Schedeler online (plus flexible est plus adéquat aux tâches apériodiques et sporadiques) n'est nullement nécessaire. Au fait, il est décrit dans [25] qu'un Scheduler offline pour un système avec un paradigme d'exécution temporel se caractérisant d'une complexité accrue dans ses contraintes temporelles est plus efficace qu'un système avec un Scheduler online.

Avec un calendrier créé dans la phase de compilation et utilisé lors de la phase d'exécution, il est évidant que le comportement du système implémentant le Scheduler offline est très déterministe dans le temps. En raison de ce déterminisme, les systèmes temps réel avec un Scheduler offline sont plus communément utilisés dans les applications hautement critiques. Toutefois, étant donné que le calendrier est créé offline, la marge de manœuvre est très limitée, dans le cas où le système devrait être changé (en raison, par exemple, d'ajout de fonctionnalité ou de changement de matériel), une nouvelle table doit être créée et attribuée au dispatcheur. Dans la plupart des cas, la création d'une nouvelle table pour le Scheduler offline est un processus complexe et fastidieux qui peut parfois demander beaucoup de temps.

3.4.3. Les Schedulers online

Les politiques de planification qui prennent leurs décisions au cours de l'exécution sont classées dans la catégorie des Schedulers online. Ces Schedulers prennent leurs décisions de planification en se basant sur certaines propriétés des tâches, comme par exemple la priorité. Les Schedulers qui basent leurs décisions sur la propriété de la tâche sont généralement appelés *Schedulers basés priorité* (ou priority-based Schedulers).

Schedulers basés priorité : Dans l'utilisation des Schedulers basés priorité la flexibilité est grandement augmentée (par rapport au Table-driven Scheduler), puisque les décisions de planification sont crées online (en cours d'exécution) basées sur les attribues propres aux tâches actifs. Par conséquent, les Schedulers basés priorité ont de meilleurs attitudes pour faire face à des changements dans la charge de planification et dans l'ajout de nouvelles tâches, tant que l'ordonnancement de l'ensemble des tâches reste correct en rapport avec les exigences temps réel. Toutefois le comportement exact des Schedulers basés priorité reste difficile à prédire dans le temps. C'est pour cette raison que ces planificateurs sont en général rarement utilisés dans les applications critiques.

Les deux ordonnanceurs basés priorité les plus connus en ce qui concerne le temps réel sont le *Fixed-Priority Scheduling* (FPS) et le *Earliest Deadline First* (EDF). La différence entre ces deux Schedulers réside dans l'attribut priorité de la tâche. Pour le premier, la priorité des tâches est fixée avant son exécution, tandis que le deuxième peut les changer en cours d'exécution (c'est-à-dire, quand les tâches sont dynamiques).

Comme signalé auparavant dans les FPS, les priorités sont assignées aux tâches avant leurs exécutions. La tâche avec la plus haute priorité parmi toutes les tâches qui sont disponibles pour l'exécution est planifiée pour être exécutée. Un point important lors de la conception d'un FPS est le schéma d'assignation des priorités aux tâches. Il est connu que certaines assignations sont mieux que d'autres. Par exemple, pour un simple modèle de tâches avec des tâches périodiques sans interférence les uns sur les autres, et le délai de chaque tâche est égal à sa période, l'assignation de priorité *Rate Monotonic* (RM) a été démontré dans [26] qu'elle était optimale pour ce genres de modèle. Dans le Rate Monotonic la priorité est accordée en se basant sur la période de la tâche. Plus la période est courte, plus la priorité à assigner à la tâche est grande. Dans l'autre partie, l'affectation des priorités dans EDF est totalement différente, car dans celui-ci, la tâche avec le plus proche (le plus tôt) délai parmi toutes les tâches est sélectionnée pour l'exécution. Par conséquent, la priorité n'est pas fixe,

elle change dans le système en relation du temps écoulé.

Ordonnancement avec les tâches apériodiques et sporadiques : pour que les Schedulers basés priorité puissent faire face aux tâches apériodiques, différentes méthodes basées *service* ont été présentées. L'objectif de ces méthodes est de donner un acceptable temps de réponse pour les tâches apériodiques, tout en préservant le comportement temporel des tâches périodiques. Le principe de fonctionnement des méthodes basées service est de fournir un serveur de tâches. Ce serveur a la faculté de générer les tâches tout en étant capable de leurs affecter des priorités avant ou pendant leurs exécutions. Dans la littérature des Schedulers plusieurs types de serveurs existent. Comme par exemple pour le cas des FPS, le *Sporadic Server* (SS) [27] est un serveur qui affecte les priorités aux taches avant leurs exécutions en ce basant sur des politiques RM. Ou pour les EDF, le *Dynamic Sporadic Server* (DSS) [28,29] qui tente d'étendre le SS. Deux autres serveurs pour les EDF qui sont, le *Constant Bandwidth Server* (CBS) [30] et le *Total Bandwidth Server* (TBS) [28,31]. Chaque serveur est caractérisé en partie par son mécanisme unique d'attribution des délais, et en partie par un ensemble de variables utilisées pour configurer le serveur. Des exemples de ces variables sont le taux d'exécution, les délais, et les périodes.

3.5. Mécanismes de partage de ressources temps réel

Sans parler du processeur, les ressources partagées (comme la mémoire, les sémaphores, et les mutexes) sont également soumises à un arbitrage par le RTOS. Quand une tâche verrouille une ressource partagée elle bloque toutes les autres tâches dont elles ont besoin de cette ressource. Afin de réaliser un verrouillage prévisible dans le temps, des protocoles ont été proposés pour satisfaire ce besoin.

3.5.1. Priority Inheritance Protocol

Le Priority Inheritance Protocol (PIP) fonctionne sous le principe de faire hériter une tâche de basse priorité la priorité d'une autre tâche bloquée par cette dernière. La tâche bloquée doit avoir une plus grande priorité que la tâche qui bloque la ressource.

Il s'agit d'une méthode simple et directe pour réduire le temps de blocage. Toutefois, le calcule du pire cas de blocage est imprévisible. En sachant que le protocole n'inhibe pas l'interblocage, ainsi le calcul peut s'étendre pour l'infinie. Par conséquent, pour les systèmes temps réel dur ou lorsque le comportement temporel du système doit être calculé a priori, le PIP n'est pas le protocole le plus opportun.

3.5.2. Priority Ceiling Inheritance Protocol

Priority Ceiling Protocol (PCP) associés à chaque ressource, un plafond de valeur qui est égale à la plus haute priorité de toute tâche qui est susceptible de verrouiller la ressource. Par la manipulation judicieuse des valeurs de chaque plafond de ressource, l'ordonnanceur du RTOS va manipuler les priorités afin d'éviter les problèmes du PIP.

PCP garantit l'absence de l'interblocage entre processus, et le pire des cas de blocage est relativement facile à calculer. Cependant, la complexité du suivi des valeurs de plafond et des priorités des tâches rendent le PCP lourd en temps d'exécution, ainsi inapproprié pour les plates-formes minimes en termes de ressources.

3.5.3. Immediate Ceiling Priority Inheritance Protocol

Le Immediate Inheritance Protocol (IIP) associe également à chaque ressource, un plafond de valeur qui est égale à la plus haute priorité de toute tâche susceptible de verrouiller la ressource. Cependant, il se différentie du PCP dans le fait qu'il attribue immédiatement le plafond de la ressource à la priorité de la tâche qui la verrouille.

IIP a les mêmes caractéristiques en temps réel que le PCP (y compris le calcule du pire cas de blocage). Cependant, l'IIP est nettement plus facile à mettre en œuvre que le PCP. Il est a vrai dire, pour les systèmes avec un seul nœud le plus facile à mettre en œuvre que tous les autres protocoles de verrouillage de ressource. Pour la mise en œuvre du protocole IIP aucun verrouillage n'est nécessaire, au fait il suffit juste aux RTOS d'ajuster la priorité de la tâche qui verrouille ou libère la ressource. IIP a d'autres avantages opérationnels, en particulier il ouvre la voie à de multiples tâches d'utiliser une pile mémoire commune, ainsi il permet au RTOS implémentant le IIP de façonner des systèmes temps réel avec des empreintes mémoires extrêmement faible.

3.6. Les RTOS actuels

Il existe de nos jours une grande variété de système d'exploitation temps réel, en comparaison avec les systèmes d'exploitation pour usage généraliste. La plupart d'entre eux fournissent les outils nécessaires pour le développement de systèmes temps réel destinés aux systèmes embarqués. On peut citer quelques-uns des plus célèbre, comme Tornado/VxWorks [32], LYNX [33], OSE [34], QNX [35], RT-Linux [36], et ThreadX [37]. Toutefois, le principal problème de ces systèmes d'exploitation est leurs grandes richesses en termes de primitives. Ces systèmes fournissent à la fois des primitives qui sont fondamentalement convenables pour les systèmes temps réel. En plus, des primitives qui sont en discordance avec les principes du temps réel (ou qu'ils doivent être utilisées avec beaucoup de précaution pour ne pas porter atteinte à la rigueur temps réel). Cette situation est clairement visible par exemple dans la liste des protocoles de verrouillage des ressources partagés. Quelques-uns sont temps réel (comme le IIP), d'autres sont non temps réel, et d'autre sont temps réel mais qu'il faut les utiliser avec connaissance de cause (comme PIP). Cette richesse devient un problème quand ces systèmes d'exploitation sont utilisés par des concepteurs et/ou des chefs de projet inexpérimentés. Dans cette situation, il est très facile d'utiliser des primitives qui contribueront à l'imprévisibilité temporelle du système développé.

Toutefois, il existe un ensemble restreint de RTOS qui ont été conçus pour faire face à ce genre de problème, et aussi en même temps permettre la mise en œuvre de RTOS prévisible extrêmement léger. L'idée était de restreindre le nombre de primitives dans le système pour ainsi guider le concepteur vers une conception et une analyse correcte de son système. Des exemples de ce genre de RTOS sont, le RTOS CubTrix [38] issus du domaine de

la recherche, et le RTOS commercial Salvo [39]. Ces systèmes fournissent un modèle de tâches très simpliste, dont les tâches ne peuvent pas suspendre leurs propres exécutions (en enlevant la primitif `sleep()`). L'invocation de tâche, redémarre automatiquement l'exécution de celle-ci de son point d'entrée initiale. Le seul protocole de verrouillage de ressource pris en charge est le IIP, et la politique du Scheduler est du genre Table-driven. Ces limitations ont donné naissance à de RTOS qui sont en mesure de s'exécuter sur du matériel extrêmement léger, en assurant un comportement temporel correct du système.

Plusieurs RTOS, que ça soit commerciaux ou de la recherche fournissent des API standard (appartenant à une norme donnée). Les plus importantes normes d'API pour RTOS sont : RT-POSIX [40], OSEK [41], et l'APEX [42]. Ici nous nous limiterons sur la norme POSIX car elle est la plus largement adoptée comme standard pour les RTOS. Les deux normes OSEK et APEX sont principalement destinées respectivement à l'automobile et à l'avionique.

Le POSIX est un standard basé sur Unix. Son principal objectif est d'assurer la portabilité application au niveau code source. La norme donne des spécifications sur la gestion des tâches/threads, le système de gestion de fichier, les entrées/sorties, et la notification d'événements via les signaux. Pour le sous-ensemble POSIX temps réel, le standard définit des interfaces (API) permettant, premièrement la programmation concurrentielle, et deuxièmement de prédire le comportement du système dans le temps. La programmation concurrentielle est soutenue par des mécanismes de synchronisation et de communication qui prennent en compte le concept de prévisibilité. Tandis que le comportement prévisible dans le temps est soutenu par un Scheduler préemptif à priorités fixes, un gestionnaire de temps avec une importante résolution, et un gestionnaire de mémoire virtuelle. Plusieurs sous-ensembles du standard en vue le jour, à raison de combler les besoins de standardisation pour des systèmes plus spécifiques que ces derniers.

3.7. Conclusion

Dans ce chapitre, nous avons pu voir quelques caractéristiques des plus importantes dans le domaine des systèmes d'exploitation temps réel. L'accent a été mis sur le scheduling, dont on a distingué les deux principales familles, le offline et le online, dont la première ce caractérisant par un comportement temporel correct, tandis que la deuxième se caractérise d'une flexibilité plus accrue. Ensuite, le point du partage de ressources entre processus dans un système temps réel a été abordé. Les trois principaux protocoles en relation avec ce dernier qui sont le PIP, le PCP, et le IIP, ont été analysés. En terminant le chapitre avec une critique sur les différents systèmes d'exploitation temps réel actuellement disponible.

Il faut savoir aussi qu'un système d'exploitation temps réel est un élément nécessaire pour la construction d'un système temps réel, quoiqu'il ne soit nullement suffisant pour le faire. Car pour construire un système temps réel, tout un ensemble d'éléments comme le matériel, le système d'exploitation, et les applications, doivent être premièrement commodes pour le temps réel. Et deuxièmement, que l'adjonction de l'ensemble d'éléments préserve leurs qualités en ce qui concerne leurs comportements dans le temps. Dans le chapitre suivant nous allons aborder l'un de ces éléments non négligeable cité précédemment, qui est le matériel.

SBC pour systèmes embarqués 4

4.1. Introduction

L'univers du hardware pour systèmes embarqués est fort riche de différents composants électroniques, avec chacun ses propres caractéristiques et sont domaine d'utilisation. Il est aisé de se perdre dans ce nombre important de cartes mères, de processeurs, de mémoires, et de périphériques de communication... Ce chapitre a pour but de faire une introduction au monde des SBC (en sachant que "SBC" est le terme donné aux cartes mères pour systèmes embarqués). Pour les concepteurs du hardware, c'est l'élément de base pour leurs architectures. Il contient les éléments matériels qui vont composer le hardware du système, ce qui implique que tous les choix que devrait prendre l'équipe de conception vont se refléter sur l'SBC du système. Au final, les concepteurs du hardware auront le choix entre deux alternatives : De se procurer une carte SBC pré-construite d'un fournisseur de ces derniers, et de la personnaliser selon les besoin du projet. Ou d'en construire une en locale, en assemblant de toutes pièces les composants de la carte.

La premier partie du chapitre donne une introduction aux SBC par une définition et une classification, avec une vue sur l'historique de ces derniers. La deuxième partie traite les cartes disponibles sur le marché des SBC. Les plus importantes normes, avec un aperçu de leurs spécificités, seront abordées. En continuant avec la description d'autres SBC qui ne sont pas des normes, mais qui bénéficient d'une popularité considérable dans le domaine. Et en terminant avec un survole des SBC avec des gabarits à très petite taille.

4.2. Définition

Selon [43] "Un Single Bord Computer est un ordinateur complet construit sur une seule carte électronique (circuit imprimé). L'architecture est centrée sur un seul ou double microprocesseur, avec une mémoire centrale, des interfaces d'entrées/sorties, et de tous autres éléments nécessaires pour être un système informatique fonctionnel, le tout est réuni sur une unique carte". En règle générale le matériel pour systèmes embarqués se présente sous forme SBC. Ce sont des cartes mères à taille réduite, qui contiennent tous les composants fonctionnels qu'un système est susceptible d'avoir besoin.

4.3. Historique

Au début de la micro-informatique les micro-ordinateurs étaient constitués d'une demi-douzaine, ou plus, d'unités sous forme de circuits imprimés branchés sur un *backplane* (un backplane est un circuit imprimé faisant office de plate-forme de communication entre les différentes unités du système). Ces unités représentaient respectivement l'unité du processeur central (CPU), la mémoire, le contrôleur de disque, les ports série/parallèle...etc. En opposition au "Single Board computer" ces cartes formait ce qu'on appelle *Multi board Computer*. Ces Multi board Computer à base de backplane étaient utilisés pour l'acquisition de données, le contrôle de processus, et les projets de R&D, mais étaient en général trop volumineux pour être utilisé comme support matériel pour les systèmes embarqués.

Au début des années 80, la technologie des circuits intégrés (IC) était assez mature pour que les fonctions précédemment occupantes la totalité des circuits imprimé pourraient être disposées sur une seule puce (ou circuit imprimé). Le fait d'implanter sur un circuit imprimé une puce pour le CPU, une ou plusieurs puces pour la mémoire, une pour le stockage, autre pour le contrôle des ports série/parallèle, permettait la mise en œuvre complète d'un micro-ordinateur sur une seule carte sans l'utilisation d'un backplane. En ce qui concerne le monde industriel et les systèmes embarqués, le "Big Board" à base de processeur Zilog Z80 fut en 1980 probablement le premier Single Board Computer (SBC), avec la capacité de fonctionner sous un système d'exploitation commercial : Le CP/M.

Comme le "Big Board", le "Little Board" (produit par Ampro en 1983) a utilisé un processeur Z80 et était axé sur le système d'exploitation CP/M. Mais contrairement au "Big Board" il était beaucoup plus petit en taille, correspondant à l'empreinte d'un lecteur de disquette 5,25 pouces. Grâce à sa petite taille, sa fiabilité et son faible coût, le "Little Board" rendait pratique l'intégration un système d'exploitation commerciale dans des dispositifs qui ne sont pas elles-mêmes des ordinateurs. Au début, chaque produit SBC été complètement unique, à la fois architecturalement et physiquement, ceci était dû en grande partie à l'inhérente diversité des exigences des systèmes embarqués, combinée avec la vaste gamme de processeurs et contrôleurs de périphériques disponibles. En outre, il n'y avait pas de normes ou de spécifications pour influencer les développeurs de carte SBC sur les choix fonctionnels et mécaniques de leurs cartes.

Dès le milieu des années 80, il y avait un intérêt croissant dans le domaine de l'embarqué et d'autres applications non bureautique pour la compatibilité IBM PC, et ça pour deux principales raisons. Premièrement, le facteur matériel, car la compatibilité des périphériques et des chipsets PC pourrait produire des systèmes moins coûteux, plus simples et plus faciles à maintenir. Deuxièmement le Facteur logiciel, dès lors la compatibilité logicielle x86 pourrait permettre de tirer partie des systèmes d'exploitation PC (MS-DOS, Windows pour cette période), des langages de programmation, des outils et des logiciels d'application.

De nos jours, plusieurs facteurs importants influencent l'orientation architecturale des cartes SBC. Comme l'explosion des demandes pour l'intelligence embarquée, car même les produits et les appareils les plus petits et les moins cher exigent d'avoir un minimum d'intelligence intégrée. Sans oublier la connectivité, avec la nécessité croissante pour interconnecter tous les appareils électroniques, que ce soit par voie filaire ou sans fil. Ou encore l'évolution des périphériques et des interfaces de bus, bien que des normes populaires d'interconnexion puissent parfois sembler immortelles (comme le Centronics et RS232), de nouvelles interfaces arrivent à supplanter progressivement les anciens. Après presque deux décennies d'existence, le bus ISA a finalement été remplacé par le PCI. L'USB et le FireWire (IEEE-1394) sont maintenant entrains de remplacer les vénérables ports série, parallèle et PS/2. Sans oublier l'Ethernet qui domine actuellement les réseaux filaires.

4.4. Classification des SBC pour systèmes embarqués

Selon [43], les SBC peuvent être classifiés en rapport avec la distribution architecturale de leurs composants, en trois catégories : les SBC modulaires, les SBC tout-en-un, et les SBC macro-composant.

4.4.1. Les SBC modulaires

Ce sont des composants détachés sur des circuits imprimés, sous forme de modules, rassemblées autour d'un backplane pour construire des dispositifs consistants (comme technologie de bus pour backplane on trouve le PC/PCI, le VME, ou Compact Flash). Ou parfois mis en pile les uns sur les autres au travers de bus PCI ou ISA (comme les normes PC/104, PC/104-plus qui seront détaillés plus loin).

4.4.2. Les SBC tout-en-un

Les SBC tout-en-un contiennent la plupart des fonctionnalités informatiques embarquées sur un même circuit imprimé. Ils arrivent généralement à fournir plus de moyens de personnalisation à travers, soit un slot PC/104 (-Plus), soit des emplacements d'extension PCMCIA ou Compact Flash, pour ainsi accroître leurs capacités en termes de modularité.

4.4.3. Les modules macro-composants (Macrocomponent)

Les modules macro-composant contiennent l'essentiel des fonctionnalités embarquées génériques, comme le processeur, la mémoire vive, la ROM, et quelques contrôleurs de périphériques indispensables, condensé sur une petite carte électronique. Cette dernière est branchée sur un circuit imprimé lui servant de hôte. L'avantage de l'utilisation des macro-composants est que ces derniers sont des SBC génériques, par conséquent le circuit imprimé hôte peut être personnalisé pour une application spécifique. Ce qui donne une flexibilité et facilité d'utilisation accrue pour le concepteur.

4.5. Normes et tendances

En raison de l'énorme diversité des applications pour systèmes embarqués, le marché des produits SBC est divers autant. Toutefois quelques standards ont pu se hisser aux rangs de normes, en considération de quelques tailles pour SBC des plus usuels. Parmi ceux-ci se trouve les PC/104, les EBX et les EPIC. Quoique, les autres tailles d'SBC ont du mal à discerner une norme qui met tous les fabricants d'accord sur les différentes caractéristiques de leurs produits. Cependant quelques SBC ont un fort potentiel de devenir norme, en raison de leur acceptabilité par plusieurs fabricants. Parmi ceux-ci en trouvent les Half-Biscuit d'Advantech, Les ETX COM de Kontron, et les EnCore d'Ampro.

4.5.1. Les PC/104 et PC/104-*plus*

Le PC/104 (ou PC 104) est une norme de SBC créée par le Consortium PC/104 [44], qui définit la forme et l'architecture de la carte. La norme PC/104 impose la consommation, la taille, le bus standard, mais pas le processeur. Bien que le x86 soit très utilisé, il existe bien des architectures à base d'ARM, de PowerPC ou encore de SuperH. Initialement dérivé de la carte d'extension "MiniModules" utilisés pour étendre "Ampro's Little Board SBC", le PC/104 est maintenant devenue l'une des plus populaires normes pour les SBC pour systèmes embarqués.

Les PC/104 appartiennent à la classe des SBC modulaires. Ainsi les cartes PC/104 se présentent sous forme de modules qui s'empilent les uns sur les autres, formant une pile de cartes construisant le système embarqué (comme sur la *Figure*4-1). On peut prendre comme exemple d'une configuration comprenant le processeur plus la mémoire, un convertisseur analogique/numérique et un module d'entrées/sorties numériques.

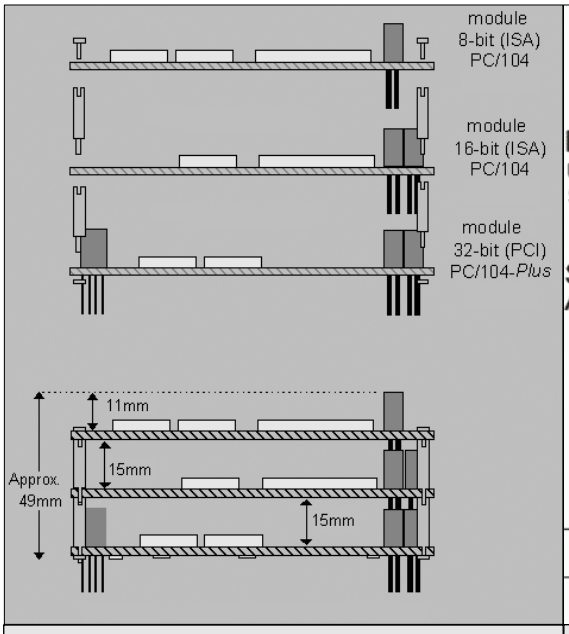

Figure 4-1 : *Une pile de cartes PC/104.*

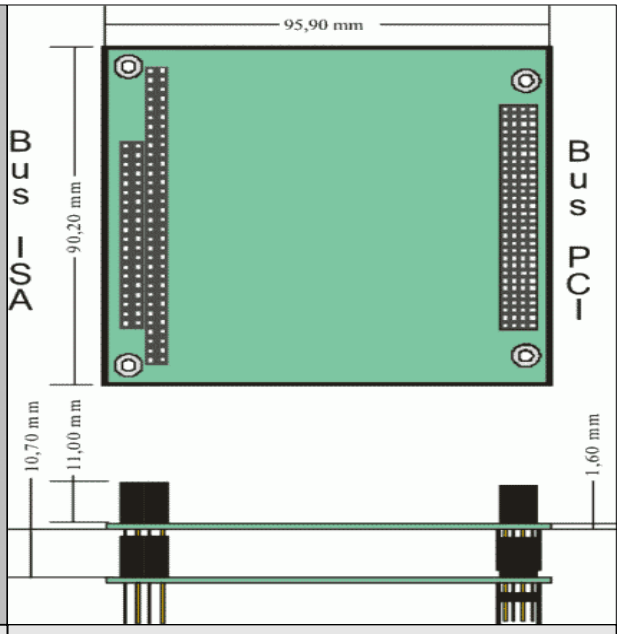

Figure 4-2 : *Les mesures d'un PC/104-Plus.*

Les cartes communiquent entre elles garce au bus ISA (le même utilisé pour les PCs au niveau signales), constitué de deux connecteurs, le premier de 64 pins et l'autre de 40 pins (comme sur la *Figure*4-2), le tout est de 104 pins, d'où l'origine du nom PC/104. La taille standard est de 3.55 x 3.775 pouces (soit 90.17 x 95.89 mm) et la hauteur dépend du nombre de modules empilés et de leur hauteur individuelle. Les cartes PC/104-plus sont identiques au PC/104 avec en plus du bus ISA pour le transfert des données inter-modules, s'ajoute un bus PCI à 120 pins identique au niveau signale à celui des PCs, offrant ainsi un plus grand taux de transfert. Il existe aussi des cartes nommées PC-104 qui sont identiques aux PC/104-plus mais sans le connecteur ISA.

4.5.2. Les EBX

L'EBX (ou l'Embedded Board eXpandable) est une norme dans l'industrie des SBC de plus en plus populaire [45]. Naissant d'une collaboration entre Ampro et Motorola Computer Group, elle est essentiellement inspirée de la carte "Ampro's Little Board". Ainsi elle appartient à la gamme des SBC moyennes tailles modulaires extensibles. Ça taille, comme montré sur la *Figure*4-3 est de 5.75 x 8 pouces (soit 146.05 x 203.2 mm).

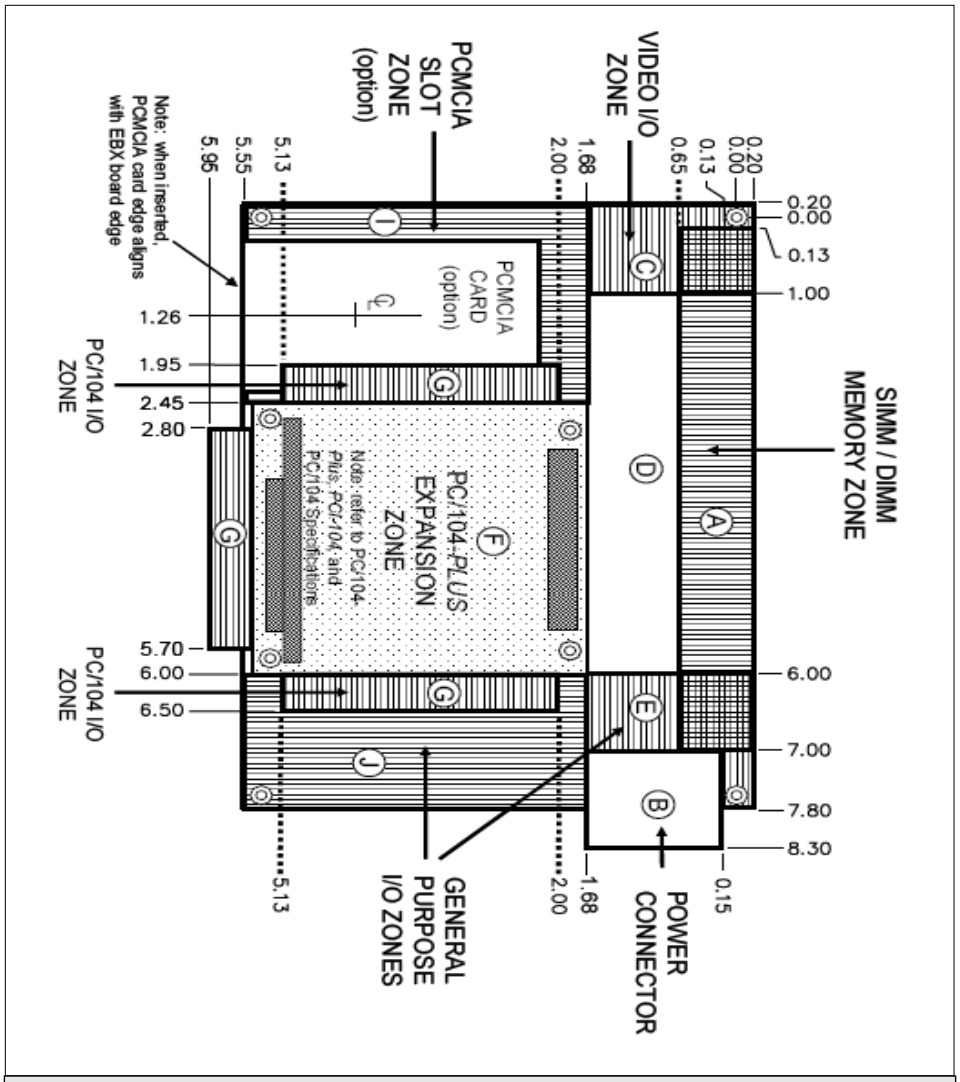

Figure 4-3 *: Le schéma d'une carte EBX.*

Par rapport à des modules PC/104, elle est plus grande (mais reste toujours raisonnable pour des applications embarquées). Les EBX ont tendance à intégrer tous les composants d'une carte mère PC standard, on trouve par exemple des interfaces audio, des interfaces analogiques/numériques pour les E/S…etc. Il est aussi bien plus facile pour ces SBC d'intégrer un CPU Pentium, alors qu'il est généralement trop encombrant ou trop cher de le faire sur un SBC PC/104. Typiquement, un EBX contient : un processeur, une RAM extensible par le biais des connecteurs DIMM ou SIMM, de la mémoire Flash faisant office de mémoire de masse, des ports USB, des ports série et parallèle, une interface réseau (généralement Ethernet) et de la vidéo (typiquement des connecteurs CRT, LCD ou TV). Les cartes EBX offrent une extensibilité accrue grâce aux connecteurs ISA et PCI qu'ils lui permettent une inter-opérabilité avec les SBC PC/104, ou pour l'ajout d'autres cartes d'extensions.

4.5.3. Les EPIC

Les EPIC [46], acronyme pour "Embedded Platform for Industrial Computing" est une norme pour SBC apparus pour satisfaire le besoin d'une taille intermédiaire mi-chemin entre PC/104 et l'EBX (comme on peut le voire sur la *Figure*4-4). Cinq fabricants de cartes pour l'embarqué : VersaLogic, WinSystems, Ampro, Micro/Sys, et Octagon se sont conjointement réunis pour développer la norme et leurs projets a était dévoiler en mars 2004 à la conférence "Embedded Systems Conference" à San Francisco. Ensuite l'EPIC a été adopté par le Consortium PC/104.

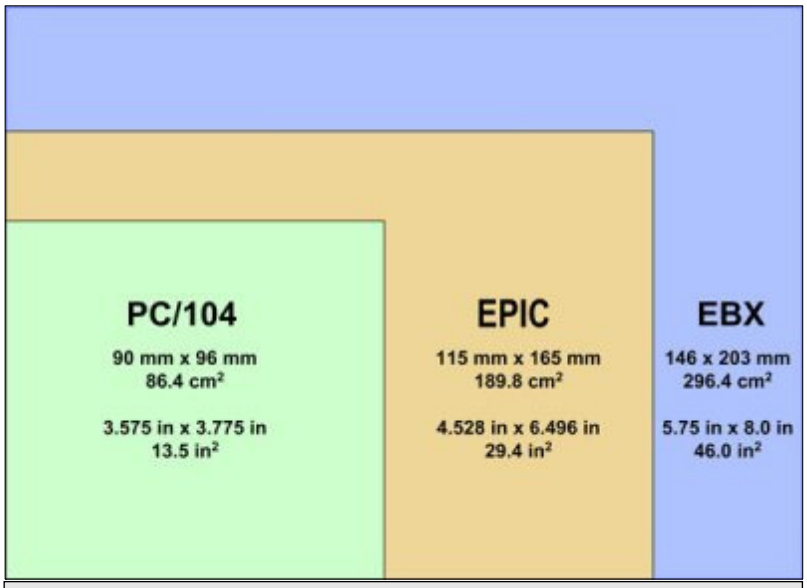

Figure 4-4 *: Comparaison entre les trois cartes PC/104, EPIC et EBX.*

Les spécifications définissent une carte de 4.528 x 6.496 pouces (soit 115 x 165 mm), et permettent des connections d'E/S à mettre en œuvre, soit sous forme de pins ou du style connecteurs PC standards. La norme fournit aussi des zones pour implanter des fonctions d'E/S telles que l'Ethernet, les ports séries, des interfaces numériques/analogiques, de la vidéo, et du sans fil, et diverses interfaces spécifiques pour des applications données. Le EPIC a aussi la capacité de supporter les bus à grande vitesse comme PCI Express. Sans oublier qu'il prend aussi en charge les formats PC/104 et PC/104-Plus grâce à ses connecteurs ISA et PCI.

4.5.4. Autres SBC

Mise à part la norme EPIC un nombre croissant de sociétés offrent des SBC qui s'inscrivent à l'intérieur de l'intervalle des tailles entre le PC/104 (soit 13 pouces carrés) et le EBX (46 pouces carrés). Bénéficient d'un espace de 1,5 à 2 fois la taille d'un module PC/104, ajouter a cela la disponibilité de nos jours de la technologie SoC (Systeme On Chip) ainsi que des contrôleurs hautement intégrés pour la gestion des périphériques, les SBC de cette catégorie ont suffisamment d'espace à bord pour pouvoir implanté à peu près tous les fonctions qu'un système embarqué serai susceptible d'avoir besoin. En outre, beaucoup de

ces SBC peuvent également être étendus en utilisant des modules PC/104-Plus ou PC/104, ou via des cartes PCMCIA ou CompactFlash. Un autre avantage de la grande taille de ces SBC, en comparaison avec les cartes PC/104, c'est qu'elles peuvent accueillir plus facilement un processeur de haute performance, car la taille de ce dernier et l'espace pour la dissipation de la chaleur n'est plus un handicape pour ce genre de carte. Parmi ces cartes en trouve :

Le Half-Biscuit d'Advantech

Comparé à un disque dur de 3,5 pouces par son fabricant Advantech, cette famille à une taille de 145 x 100 mm [47] qui contient généralement toutes les fonctions que peut comporter un PC complet (comme sur la *Figure*4-5). Plusieurs versions d'architectures basées sur 486 sont disponibles, comme par exemple les processeurs National Geode, et les processeurs Transmeta Crusoe. Cette carte aussi disponible par plusieurs autres producteurs taiwanais dont Aaeon, Axiom, ICP, et Lanner.

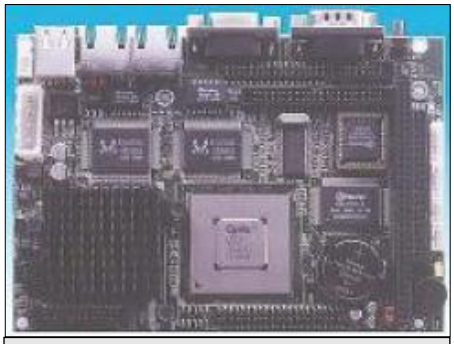

Figure 4-5 *: Le Half-Biscuit.*

Le EnCore d'Ampro

Il existe plusieurs variantes de ce SBC avec chacun une taille de 100 x 145 mm (comme sur la *Figure*4-6) comprenant un processeur, un système de stockage et de la mémoire Flash, ainsi qu'un ensemble d'interfaces périphériques standard (IDE, disquette, Ethernet, série, parallèle, USB, et sonores) [48], et certains modules fournissent également des contrôleurs graphiques pour des moniteurs CRT ou des écrans plats. Pour plus de flexibilité les modules EnCore ont la capacité de supporter des bus PCI pour les PC/104-plus, mais pas les bus ISA.

Figure 4-6 *: Le EnCore.*

Les EnCore sont des SBC de type macro-composant ce qui veut dire qu'ils sont conçus pour être enficher sur des cartes de base *baseboards* (ou carte hôte) développer par le client. L'interface se fait au moyen d'une combinaison de bus PCI et des connecteurs d'E/S normalisés. Les modules EnCore ont était conçus pour cohabiter à plusieurs sur une même baseboard, ce qui fait que plusieurs architectures peuvent être assemblées sur une même carte de base sans aucun problème, ce qui rend le EnCore un sérieux candidat pour être un standard pour les modules à architectures de processeur indépendants. Ampro produit actuellement des modules avec des architectures pour processeur 486, Pentium, Pentium III, MIPS, et PowerPC.

Le ETX COM de Kontron

Le ETX Computer-On-Modules [49] est un SBC du type macro-composant. C'est une carte hautement intégrée, puisqu'elle contient un CPU, une mémoire, la connectique d'E/S d'un PC/AT (série, parallèle, etc), USB, audio, graphiques, et l'Ethernet. Tous ça sur une carte de dimension 112 x 94 mm (comme sur la *Figure*4-7). Tous les signaux d'E/S ainsi que d'une mise en œuvre complète de l'ISA et PCI sont mappés sur quatre connecteurs à haute densité sur la partie inférieure du module. Une association JUMPtec-Adastra avait lancé la norme avant d'être acquise par la société Kontron. Kontron vend ses ETX COM avec des processeurs National Geode et processeurs Intel Pentium III. Au moins deux autres sociétés sont aujourd'hui favorables à cette norme ETX, notamment Advantech et de la TMC Technology. De plus, Kontron a récemment annoncé des plans pour une mise à niveau de ses ETX par le PCI Express, appelé ETXexpress.

Figure 4-7 : *Le ETX COM.*

4.5.5. Les SBC à taille réduite

Un nombre de plus en plus croissant de SBC de très petite taille, de moins d'une douzaine de pouces carrés, où ça peut même atteindre les moins de trois pouces carrés, avec des taux d'intégrations très élevés puisque sur seulement quelques pouces carrés, on peut avoir toutes les fonctions d'un système informatique complet, notamment Le CPU, la mémoire vif, la mémoire flash, les ports série et parallèle, les interfaces d'affichage et des interfaces réseaux. Leurs utilisations touchent un large éventail d'applications, depuis les appareils de poche aux périphériques informatique en passant par les caméscopes et appareils photo

numériques.

Une observation notable par rapport à cette classe de SBC, c'est le manque flagrant d'interopérabilité entre ces minuscules systèmes modulaires. Bien qu'il y ait une tendance à adopter la taille et le format des connecteurs de modules mémoire DIMM ou SIMM, il y a une absence de cohérence dans la manière dont leurs signaux sont affectés aux connecteurs du module. Également, avec le grand nombre de ces produits profitant de processeurs de la dernière technologie des SoC (System On Chip) comme StrongARM, Elan et Etrax, ils sont généralement non compatibles x86. Parmi ces cartes en trouve :

Le Zingu de Telematix

Cet SBC de 68 x 92 mm (comme sur la *Figure*4-8) est basé sur un processeur Intel Xscale i80200 qui monte en fréquence jusqu'à 850 Mips. Il inclut 128 MB de SDRAM et 32 MB de mémoire flash, en plus d'un contrôleur vidéo, un UART, les codecs audio AC97, bus PCMCIA et un bus I2C. La consommation est au-dessous de 2,5 Watt [50].

Figure 4-9 *: Le Bitsy.*

Le Bitsy de ADr

Cet SBC de 101,5 x 92 mm (comme sur la *Figure*4-9) est basé sur un processeur Intel StrongARM SA-1110 avec une fréquence de 206 MHz (plus du SA-1111 comme processeur annexe). Il ne consomme pas plus que 450 mW, comprend un port série, USB, audio, des E/S numérique et analogique, un emplacement PCMCIA de type II, en plus d'un contrôleur LCD couleur de résolution 1024 x 1024 [51].

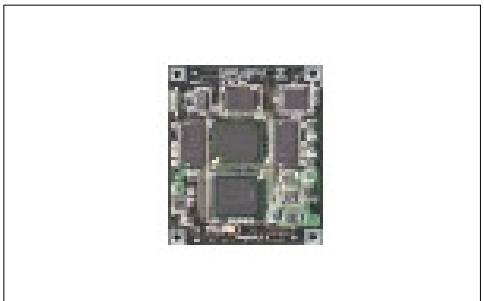

Figure 4-8 *: Le Zingu.*

Le Balloon board d'Aleph One

Aleph One [52] vent des Ballooon board de 33 gramme et 1 Wat tde consommation (comme sur la *Figure*4-10), basés sur des StrongARM de 206MHz de fréquence. pré-installés avec du soft open source (Linux embarqué) et du matériel open source (bien que c'est la conception qui est open source). Aleph One encourage les concepteurs à utiliser librement la conception, la mise en œuvre et de contribuer en retour avec des informations utiles à d'autres. Aleph One affirme que sa Balloon board est idéale pour l'utilisation dans les systèmes de Contrôle, les périphériques portables, les ordinateurs portables et la robotique.

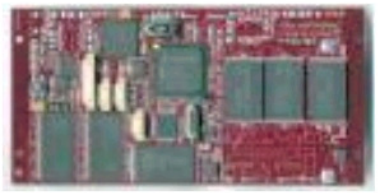

Figure 4-10 *: Le Balloon board.*

4.6. Conclusion

Le marché des SBC pour systèmes embarqués est beaucoup plus riche que celui de son homologue PC, ça s'explique par le caractère polyvalent des systèmes embarqués, puisque ils touchent pratiquement tous les secteurs de la technologie moderne, comme l'automobile, l'avionique, l'aérospatiale, les télécommunications, la domotique, le multimédia, la vidéo ludique...etc. Il faut savoir qu'un nombre élevé de constructeurs pour cartes SBC, avec pour chaque constructeur une panoplie de cartes à lui tous seul, sans oublier les SBC propriétaires développées par une firme et utilisées spécialement pour ces propres produits, et qui ne serons jamais vendus sur le marché des cartes SBC. Donc ce chapitre sur les SBC pour systèmes embarqués avait pour but de faire un tour absolument non exhaustive sur les SBC disponible actuellement. Au fait, il donne juste un aperçu sur les grandes lignes qui régissent les architectures SBC pour systèmes embarqués. Le chapitre commence avec les SBC qui forment des standards comme le PC/104, l'EBX, le EPIC. Plusieurs centaines d'SBC suivent ces standards et il est aisé de trouver des modules d'extensions compatibles avec ces normes pour pouvoir personnaliser une carte pour l'adapter à un projet bien précis. Ensuite, il liste d'autres cartes bien connues sans être pour autant des standards, après il s'étale sur une section pour les SBC extrêmement petites, couvrent des domaines bien divers, comme la téléphonie, les PDA, et la robotique petite taille.

L'IPTV 5

5.1. Introduction

Au cours de la dernière décennie, la banalisation de la diffusion satellite, du câble numérique, et la naissance de la télévision haute définition ont tous apporté un grand pas d'évolution dans le domaine de la télévision. Cette dernière qui n'a pas connu de réel changement depuis les années soixante-dix avec l'apparition de la télévision en couleurs, risque encore de connaître une authentique métamorphose avec ce qu'on appelle la *Télévision IP* (ou IPTV). La technologie IPTV va surement changer la perspective du grand public vis-à-vis de la télévision usuelle, car cette dernière va amener aux spectateurs en plus des chaînes terrestres et satellites, de la vidéo, des DVD a louer, la VoD, la presse écrite, du contenu web, le PVR, la météo locale, du contenu ludique, l'interactivité avec d'autres téléspectateurs...etc. Et tout sa sur un simple téléviseur standard ou haute définition.

Ce chapitre va dans sa globalité être scindé en deux parties. Dans la première partie nous nous focaliserons sur l'IPTV fournie par l'intermédiaire des fournisseurs d'accès à internet, étant donné qu'actuellement ces derniers sont les principaux fournisseurs de diffusion IPTV pour le grand publique. On débutera avec une définition et un survol sur la terminologie utilisée dans ce domaine. On continuera ensuite avec une étude plus ou moins détaillée sur l'architecture IPTV, sur le matériel et le logiciel habituellement choisi par les fournisseurs d'accès à internet pour implanter leurs installations IPTV. Pour la deuxième partie, on débutera avec une revue sur l'architecture IMS qui représente l'effort de normalisation émis par les grandes organisations de standardisation dans le domaine. Ça commencera avec une inspection sur les caractéristiques fondamentales de l'IMS, pour ensuite étudier l'architecture globale d'IMS avec un survol sur ses principaux composants. Pour terminer avec une exploration de quelques protocoles majoritairement utilisés dans l'IMS.

5.2. Définition

L'IPTV peut être définie comme un système dans lequel des services pour la télévision numérique sont fournis aux clients en utilisant le protocole internet IP, conçus principalement sur une infrastructure réseau pour Internet (ligne téléphonique, câble...) au lieu des réseaux de distribution usuelle, comme le satellite (DVB-S), la télévision terrestre (DVB-T) ou la télévision par câble (DVB-C). La définition officielle pour l'IPTV de ITU-T FG IPTV [53] (International Télécommunication Union focus group on IPTV) est comme suite : "l'IPTV est définie en tant qu'un ensemble de services multimédia, comme la télévision, la vidéo, l'audio, le texte, les graphiques, les données. Ces services sont délivrés sur des réseaux à base IP et doivent respecter un certain QoS, ainsi que des critères de sécurité, d'interactivité et de fiabilité".

5.3. IPTV et Internet TV

L'abondance de termes se relatant à la télévision et à l'IP peuvent facilement induire une personne à la confusion. On peut par exemple trouver les termes *IPTV*, *TV over IP* ou *Internet TV*. En ce qui concerne l'IPTV, ça implique généralement les services de télévision proposés par les fournisseurs de service internet avec l'offre qu'on appelle habituellement le *triple play* (internet, téléphonie par IP et télévision par IP), ou encore les opérateurs de téléphonie mobile avec l'offre *quadruple play* lorsque les trois services du triple play sont fournis en mobile. Dans le triple play ou le quadruple play une portion de la bande passante est entièrement dédiée à la télévision, se qui rend l'IPTV hautement fiable, avec une garantie sur le QoS et sur le débit nécessaire pour la télévision. L'usager doit avoir un STB (Set Top Box : tout appareil électronique pouvant être déposé au dessus/dessous d'un téléviseur, ça inclus les magnétoscopes, les récepteurs DVB... etc) pour pouvoir décoder et contrôler les chaines qui lui sont fournies.

Contrairement à l'IPTV, l'Internet TV (ou par fois appelé *WebTV*) se repose sur le réseau Internet pour pouvoir acheminer son flux vidéo aux usagers. La diffusion des chaines se fait principalement à partir de serveurs web pour ainsi permettre aux détenteurs d'une connexion internet la visualisation du flux vidéo sur un ordinateur personnel avec un simple navigateur web. Dans ce cas, l'usager sera amené à utiliser son PC au lieu d'un poste télé pour regarder la télévision, ce qui peut parâtre un peu contraignant pour certains. Cependant la principale contrainte de l'Internet TV par rapport à l'IPTV c'est bien la qualité de service. Étant principalement transmis sur Internet, son niveau de QoS ne peut pas devancer le niveau *best effort*, rendant ainsi le flux vidéo sujet à plusieurs défectuosité comme la démesure dans le délai ou le jitter (la variation dans les délais), l'insuffisance dans la bande passante, la fragmentation du flux...etc. Le principal avantage de l'Internet TV par rapport à l'IPTV est son aisance d'installation, puisque sont infrastructure réseau est l'Internet, nul besoin d'une nouvelle installation réseau lourde et couteuse comme pour l'IPTV. Pour mieux résumer les différences entre l'IPTV et l'Internet TV on s'appuie sur le *Tableau*3 ci-dessous :

	IPTV	Internet TV
Portée	Locale, ou nationale au maximum.	Internationale, l'ensemble du web.
Usagers	Abonnées avec leurs adresses IP et leurs régions géographiques bien connues.	N'importe quel utilisateur avec une connexion Internet, donc avec un minimum d'informations sur ce dernier.
Qualité vidéo	Avec un QoS bien contrôlé, la qualité est équivalente à celle du DVB.	Avec un QoS en Best Effort, la qualité est inférieure à celle d'un DVB.
Format de compression vidéo	MPEG-2 MPEG4-Part2 (ASP) MPEG4-Part10 (H.264)	Flash QuickTime RealNetworks
Bande passante	Entre 1 et 4Mb.	Généralement < 1Mb.
Réception	STB + téléviseur.	PC.
Fiabilité	Stable.	Relative à la connexion.
Sécurité	Les usagers sont authentifiés et protégés.	Peu sûr.
Services clientèle	Généralement présents.	Généralement absents.
Propriété intellectuelle	Présentes.	Généralement absentes.
Autres services	PVR (Personal Video Recorder) EPG (Electronic Program Guide).	Rare.

Tableau 3 : *Table de comparaison entre l'IPTV et l'internet TV.*

En se qui concerne le terme TV over IP, c'est un terme qui peut être employé pour désigner les deux technologies, l'IPTV comme l'Internet TV.

5.4. Fonctionnement des réseaux IPTV

Pour comprendre le fonctionnement général d'une installation IPTV, il faut suivre le parcours du flux vidéo de sa source qui se trouve dans les locaux du FAI (Fournisseur d'Accès Internet) jusqu'au STB de l'usager qui se trouve dans la demeure de ce dernier, en passant par l'opérateur téléphonique chargé de fournir la connexion xDSL. La *figure*5-1 donne une représentation graphique du parcours typique suivie par le flux vidéo dans une installation IPTV.

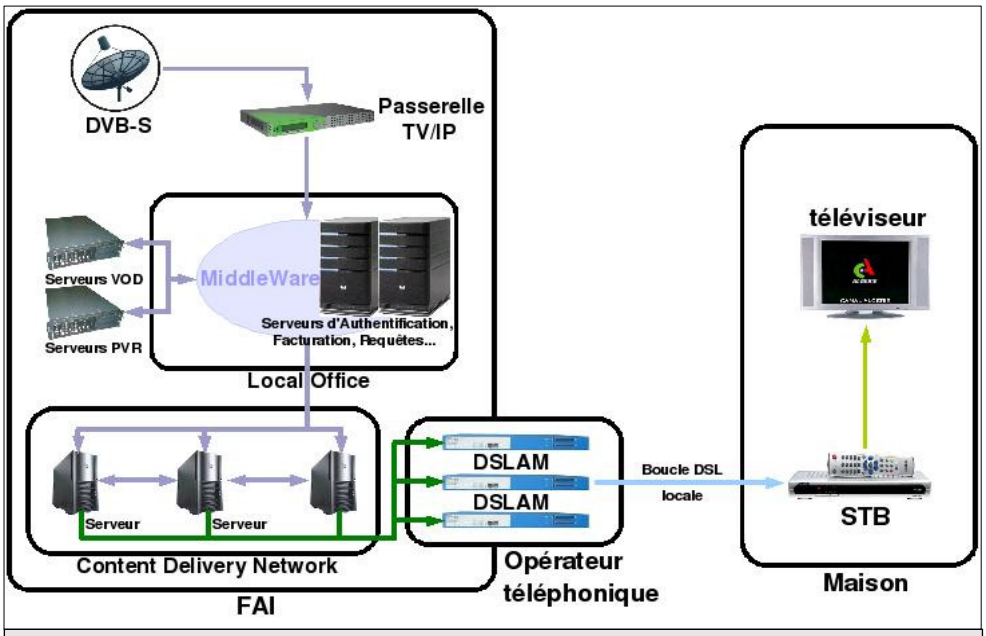

Figure 5-1 *: Système embarqué typique.*

Comme on le voit sur la *figure*5-1, le flux vidéo entre dans le système avec l'appui de la passerelle TV/IP, cette dernière est branchée d'un bout sur un récepteur satellite (DVB-S: Digital Video Broadcasting - Satellite). Le récepteur satellite n'est là qu'à titre d'exemple, ça peut être n'importe quelle source vidéo numérique (DVB-C, DVB-T...) ou même analogique. L'autre bout est directement lié à l'infrastructure réseau interne du FAI, ainsi le rôle de la passerelle TV/IP est de décomposer le flux vidéo en entrée en paquets IP et de les déversés dans le réseau de télécommunication du FAI (la passerelle TV/IP peut aussi faire office d'un encodeur vidéo, l'encodage ce fait le plus souvent en MPEG2, MPEG4-ASP, H.264). Le réseau de télécommunication interne du FAI est un immense réseau IP qui gère en plus de la vidéo, toutes sortes d'autres trafics comme les données, la voix... etc. L'avantage d'une architecture de la sorte est que le FAI détient le contrôle totale de son réseau de télécommunication de la source (la passerelle TV/IP, appelé aussi *Headend*) jusqu'à la cible (DSLAM). Dès lors qu'intervient le rôle du QoS, contrairement à l'Internet qui à un QoS qui n'excède pas le *best effort,* le niveau du QoS dans le réseau interne du FAI pour la vidéo ce fait habituellement ajusté aux niveaux *Controlled Load* ou *Voice and Video*, assurant ainsi des délais et une bande passante correcte pour la diffusion vidéo avec la fluidité que l'usager s'attend de son fournisseur.

Le flux vidéo venant du headend est directement déversé sur ce qu'on appelle dans la littérature de l'IPTV le *local office.* Le local office est un ensemble de serveurs inter-connectés constituant l'hôte hébergeant l'*IPTV middleware.* L'IPTV middleware dans l'architecture IPTV représente l'élément le plus important de celle-ci, étant donné que ce dernier est responsable de la gestion des usagers comme l'authentification, le changement de chaines (habituellement en servant le protocole IGMP responsable de la gestion multicast), la facturation, le contrôle parental, ...etc. l'IPTV middleware est aussi responsable de la gestion du contenu vidéo de l'infrastructure. Comme on peut le voir sur la *figure*5-1 le local office est directement lié aux serveurs VOD (Video On Demande : vidéo à la demande, comme les services de locations multimédia à distance proposés par les FAI) ou les serveurs PVR (Personal Video Recorder : pour le visionnement en différé des émissions manquées par l'usager), ou encore les serveurs pour le contenu publicitaire. Plus exactement, l'IPTV middleware a comme rôle la facturation des vidéos à la demande, la gestion et

le suivi du contenu sur les différents serveurs ou NAS (Network Attached Storage : système de fichier distribué sur plusieurs serveurs de stockage), le swaping des vidéo les plus fréquemment demandés on les plaçant sur les zone de stockage les plus rapides. Un autre rôle de l'IPTV middleware et la gestion et le monitoring de l'ensemble de l'infrastructure réseau, pour la gestion, ça consiste à fournir des outils automatiques ou/et manuels pour l'équilibrage de charge sur le réseau, ou pour le monitoring en fournissant les outils de détection de défaillances des serveurs ou les outils de signalisation des goulots d'étranglement. Plusieurs produits commerciaux faisant office d'IPTV middleware existent comme la suite **Video Server Appliance**, l'**Interactive Media Manager** et le **Multimedia Content Manager** de la firme **Alcatel-Lucent** [54] ou les suites de **Thomson** [55] ou de **Cascade** [56]. Pour l'open source il existe quelques projets qui devront aboutir à des produits corrects dans les années qui viennent, on peut donner comme titre d'exemple le projet **Toroid** [57]. Une liste de plus d'une centaine d'auteurs de solution IPTV est rapportée par le Multimedia Research Group (MRG) [58], ce dernier est l'un des principaux cabinets d'analystes du marché de l'IPTV, son rapport biannuel intitulé *IPTV Market Leaders Report*, indique le classent des IPTV middleware les plus influant dominent les différents segments du marché de l'IPTV.

L'IPTV middleware ensuite distribue le flux vidéo aux différents serveurs du CDN (Content Delivery Network). Le CDN est constitué de serveurs reliés en réseau, qui coopèrent afin de mettre à disposition le contenu vidéo aux usagers. Ce réseau est constitué de nœuds répartis géographiquement, et généralement connectés entre eux par des liaisons *backbone* leurs assurant une interconnexion haut débit. L'idée derrière l'utilisation des CDN est de fournir une architecture décentralisée, inhibant ainsi les inconvénients d'un seul serveur pour la diffusion vidéo à tous les usagers. Dans l'architecture CDN, Les serveurs coopèrent afin de satisfaire les requêtes émises par les usagers en leurs envoyant le flux et le contenu à travers le serveur le plus proche d'un usager donné, et cela d'une façon totalement transparente à ce dernier. Les CDN contribuent fortement dans l'optimisation du mécanisme de transmission, cette contribution se traduit par la réduction des coûts de bande passante et l'amélioration des délais d'attentes du flux.

L'étape finale est relativement simple a implémenté, puisque chaque serveur du CDN est relié directement à un DSLAM (Digital Subscriber Line Access Multiplexer) appartenant à l'opérateur téléphonique. Ce dernier comme son nom l'indique est un multiplexeur qui permet de récupérer le trafic (Internet, IPTV, VoIP) issu des serveurs CDN et de l'injecter dans les lignes téléphoniques (le circuit qui lie le DSLAM et le modem/STP est appelé *boucle DSL locale*). La technologie impliquée dans le transfert de données entre le DSLAM et le modem/STP et bien entendu l'ADSL, ADSL2, ou ADSL2+...etc. Après que le STB (dans le cas de l'IPTV, le terme *IP-STB* est plus approprié) commence à recevoir les paquets IP, il les assemble pour former le flux vidéo, ce flux est ensuite décodé avec l'un des codecs MPEG2, MPEG4, ou H.264 installé sur l'STB. Et finalement après le décodage, le STP transmet le flux vers le téléviseur de l'usager.

En ce qui concerne les protocoles utilisés pour l'IPTV, l'ensemble de ces derniers est plus ou moins géré par L'IPTV middleware du FAI. L'un de ces protocoles est le protocole IGMP, c'est le protocole le plus usuel pour la gestion de la diffusion multicast. Contrairement au DVB-S et DVB-C, l'IPTV utilise le multicast au lieu du broudcast, ainsi lorsque l'utilisateur tente de changer de chaine, il va s'abonner à un groupe de diffusion, au lieu de recevoir toutes les chaines pour ensuite en sélectionner une comme pour les DVB. Le protocole IGMP se décline en deux parties, une partie hôte installée sur les STB et susceptible d'émettre des requêtes d'appartenance à un ou plusieurs groupes multicast, et la deuxième partie est la partie routeur, sont principal rôle est d'enregistrer les requêtes des hôtes et d'ériger le flux multicast selon la base d'enregistrements de ces derniers. La partie routeur est généralement implantée sur les serveurs du CDN ou sur les serveurs du local

office, quoiqu'il ne soit pas rare de voir des serveurs implémentant les deux parties, de cette manière il devient possible de construire une structure hiérarchique du groupage multicast. Le protocole IGMP a connu au cours de son évolution trois versions, IGMP v1 [59], IGMP v2 [60], et IGMP v3 [61], la version 2 est la plus répondue, néanmoins de part ça nouveauté la version 3 commence à prendre du terrain. En ce qui concerne le transfert proprement dit du flux vidéo, le Real-time Transport Protocol (RTP) est le plus utilisé, c'est un protocole qui définit un format normalisé pour la livraison des paquets audio et vidéo sur Internet. Il a été développé par l'IETF (Internet Engineering Task Force) et publié pour la première fois en [62] et amélioré après en [63]. Le RTP est habituellement utilisé en conjonction avec le protocole RTP Control Protocol (RTCP) [64]. Bien que le RTP s'assure du transport du flux média, le RTCP est utilisé pour le contrôle du transport et de la qualité de service. Lorsque les deux protocoles sont utilisés conjointement, le RTP généralement prend un numéro de port pair, alors que le RTCP utilise le numéro de port impair qui suit directement celui du RTP. Pour ce qui est de la gestion des sessions, deux protocoles sont communément utilisés, le Session Initiation Protocol (SIP) [65] et le H.323 [66]. Le protocole SIP peut être utilisé pour créer, modifier et mettre fin à une session à deux partis (unicast) ou multipartis (multicast) d'un ou de plusieurs flux médias. La modification de session peut impliquer la modification d'adresses/ports, d'invitation de participants, l'ajoutant ou la supprimant d'un ou plusieurs flux de médias... etc. L'un des principaux avantages du SIP par rapport au H.323 c'est qu'il est conçu pour être indépendant de la couche transport du modèle OSI sous-jacent, il peut fonctionner avec le protocole TCP, UDP, ou même SCTP [67] (Stream Control Transmission Protocol : protocole de la couche transport couvrant le transport de flux multimédia). Pour le H.323, plus qu'un protocole, il ressemble davantage à une association de plusieurs protocoles différents et qui peuvent être regroupés en trois catégories : la signalisation, la négociation de codec, et le transport de l'information. Le H.323 a été un protocole pionnier de la téléphonie sur IP, mais peu adapté pour l'IPTV, tandis que le SIP de conception un peu plus récente, vient du monde de l'Internet (IETF) et s'intègre sans doute un peu mieux sur les réseaux IP.

Il faut savoir que les techniques utilisées précédemment sont plus ou moins communes dans le domaine de l'IPTV, quoique chaque FAI ou développeur de solution IPTV prend ces propres choix en ce qui concerne les technologies à utilisées ou l'architecture réseau à déployer. En d'autres mots, ça n'obéit à aucune norme ou standard connu, étant donné que l'IPTV est une technologie moderne, les normes et standards prennent du temps avant d'être techniquement prêts, et que les majors de télécommunication et fournisseurs de solution IPTV prennent en considération ces normes. À l'heure actuelle les deux organisations de standardisation l'UIT-T et ETSI travaillent sur ce qu'on appelle les normes IMS. Les avantages de cette approche sont flagrants, puisque les réseaux de télécommunication seront en mesure d'offrir à la fois la voix et des services IPTV au cours de la même infrastructure de base, et la mise en œuvre des services combinant des services de télévision conventionnelle avec la téléphonie deviendra totalement facile.

5.5. IP Multimedia Subsystem

L'IP Multimedia Subsystem (IMS) est une architecture standardisée pour la distribution de services multimédias sur des réseaux IP à la fois temps réels comme la voix, la vidéo, la visioconférence…, et non temps réel comme le Push To Talk, la messagerie et la messagerie instantanée. Elle a été spécifiée pour la première fois par la *3rd Generation Partnership Project* (3GPP: association de groupes de télécommunication pour la spécification des normes 3G pour les technologies cellulaires) pour une intégration des services Internet aux normes 3G. Ensuite il a été étendu par ETSI dans le cadre de ses travaux sur les réseaux *Next Generation Network* (les NGN visent à unifier sur un seul réseau le support de différents services comme la voix et la transmission

de données conjointement sur des réseaux fixes et mobiles). Un sous-organisme de normalisation de l'ETSI appelé le *Telecommunications and Internet converged Services and Protocols for Advanced Networking* (TISPAN) standardise l'IMS en tant que sous-système de NGN.

L'architecture IMS est une architecture dite de *bout-en-bout* (end-to-end), ca veut dire qu'elle doit supporter plusieurs types d'équipements différents en allant de la source jusqu'à l'usager. De cette caractéristique découle aussi pour l'IMS la qualification d'architecture à *service agnostique*, ce qui se traduit par une prestation de service qui doit être indépendante de la technologie d'accès sous-jacente. L'une des vertus de cette caractéristique est la facilité de l'implémentation du *Roaming* (la capacité du terminal mobile à changer de réseau sans réinitialiser la connexion, même pour des réseaux hétérogènes). Il est évidant que le protocole IP se présente comme le meilleur candidat pour cette caractéristique, étant donné qu'il est le prédominant dans l'Internet et commence à prendre du terrain sur d'autres réseaux.

L'une des spécificités importante de l'architecture IMS est la qualité de service. Au fait, le niveau du QoS qui peut être fourni dans l'architecture IMS détermine les services qui peuvent être déployés dans ses réseaux. En conséquence, les fonctionnalités de gestion du QoS sont intégrées dans l'architecture IMS. Selon [68] la hauteur du QoS dans l'IMS réside au niveau *Voice and Video*.

Une autre spécificité pour les IMS c'est leurs architectures horizontales. Dans une architecture horizontale, une couche fournit un ensemble de fonctions communes appelées habiliteurs de service, ces fonctions doivent être utilisées comme appui par les services d'une couche supérieure. En peut citer comme fonctionnalités, la gestion des groupes/listes, la présence/absence des usagers, la facturation clientèles...etc. Cela rend la mise en œuvre des services beaucoup plus facile, plus flexible et plus rapide. Et permet en outre une interaction relativement étroite entre plusieurs services qui peuvent a priori être totalement disjoints, puisque plusieurs d'entre eux peuvent utiliser un habiliteur de service commun. Il s'agit d'une amélioration notable par rapport à la plupart des architectures actuelles, qui suivent généralement des architectures verticales. L'architecture verticale ou appelée aussi *tuyau de poêle* est une architecture de couches superposées dépendantes les unes des autres. La *figure*5-2 donne une illustration de comparaison entre les deux architectures.

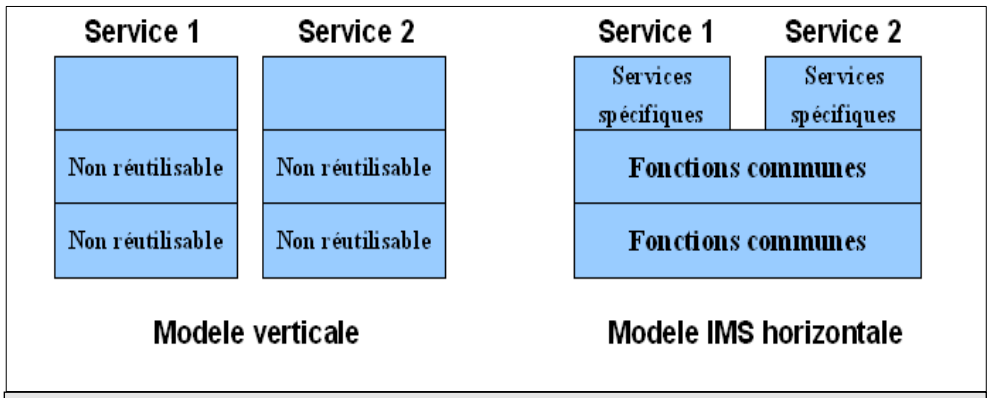

Figure 5-2 : *Comparaison entre une intégration horizontale et une intégration verticale.*

5.5.1. Structure de l'architecture IMS

L'architecture IMS est une infrastructure de services. On peut voir son diagramme fonctionnel sur la *figure*5-3 :

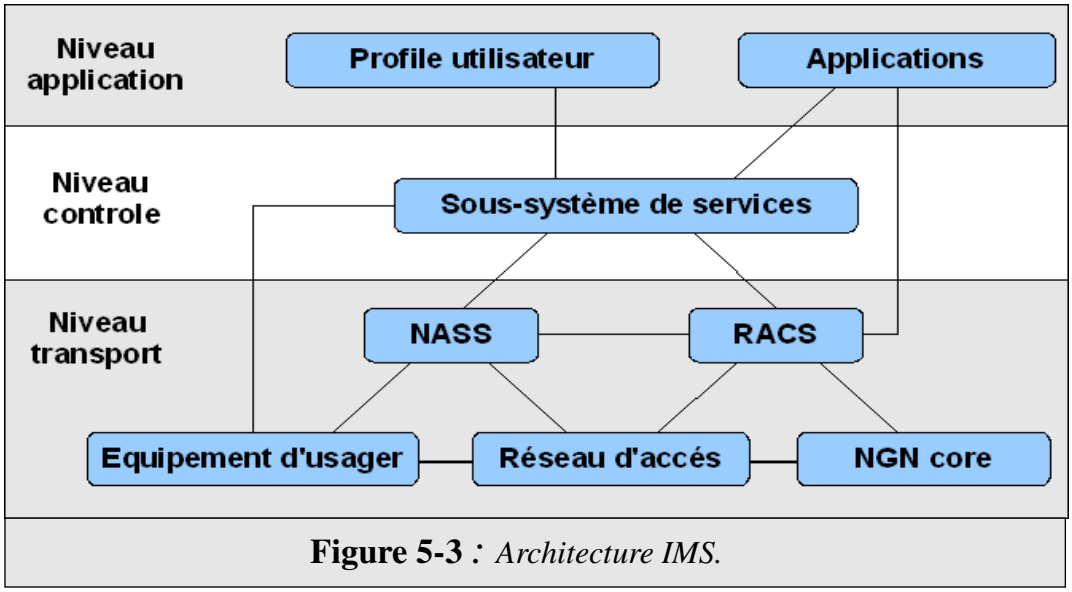

Figure 5-3 *: Architecture IMS.*

Sur le diagramme on peut distinguer 3 niveaux :

- **Le niveau Application** : qui ce compose du *Serveur d'applications* qui détient les services IMS, avec le serveur du profile utilisateur ou appelé aussi le *Home Subscriber Server* (HSS).

- **Le niveau Contrôle** : ce compose de plusieurs habiliteurs de service dont le plus important est l'*IMS core*, avec le plus souvent en coexistence avec le PSTN-ES (Public switched telephone network - Emulation Service). Le rôle du PSTN-ES est d'émuler les services d'un réseau PSTN pour une compatibilité avec les réseaux téléphoniques commuté.

- **Le niveau Transport** : il est composé de plusieurs compartiments fonctionnels dont, le NASS et le RACS (qui serons détaillés plus loin), équipement d'usagers, réseau d'accès, et le NGN core.

La *figure*5-3 donne une représentation fonctionnelle de l'architecture de l'IMS. Contrairement à une représentation matérielle qui expose une distribution physique des machines dans l'environnement, la représentation fonctionnelle dévoile la distribution logique des fonctionnalités. Ainsi plusieurs fonctionnalités peuvent être implantées sur la même machine. Ou l'inverse, une fonctionnalité distribuée sur plusieurs machines. Les sections qui suivent vont aborder les fonctionnalités les plus importantes dans une architecture l'IMS.

Le NASS (Network Attachment Subsystem):

Le NASS est une unité fonctionnelle qui a le rôle principal de fournir les adresses IP et les paramètres de configuration aux équipements des usagers d'une manière dynamique. Ses fonctionnalités peuvent être grossièrement résumées de la manière suivante, le NASS joue le rôle d'un serveur DHCP, d'un client RADIUS et fournit des fonctionnalités de gestion de localités.

Plus précisément, il peut fournir :

- Les adresses IP et les paramètres de configuration.
- L'authentification des usagers.
- Les autorisations aux réseaux d'accès selon les profiles des usagers.
- La gestion de la localité.

Le RACS (The Resource Admission Control Subsystem):

Le RACS est le sous-système responsable de la mise en œuvre des politiques de contrôle du transport, en s'appuyant sur des procédures et mécanismes qui traitent la réservation des ressources et le contrôle d'admission, à la fois pour le trafic unicast et multicast pour les deux entités fonctionnelles : réseaux d'accès et NGN core [69]. En plus d'agir comme un contrôleur de ressources, les RACS comprennent également un support pour le Network Address Translation (NAT) pour les réseaux d'accès.

Le RACS est divisé en deux blocs fonctionnels, le *Serving Policy Decision Function* (S-PDF) et le *Access Resource and Admission Control Function* (A-RACF). Le S-PDF exécute les décisions de politique de gestion et envoi ses demandes d'allocation de ressources au A-RACF (il peut aussi communiquer les décisions de politique avec le serveur d'application pour indiquer à ce dernier la politique de gestion à suivre). Le A-RACF effectue des contrôles d'admission, ça veut dire qu'il vérifie les ressources demandées s'ils sont alloués pour l'accès concerné ou non. Ensuite il retourne le résultat du contrôle de l'accès au S-PDF.

L'IMS core :

Considéré comme le plus important des habiliteurs de service, l'IMS core se charge de la plus importante partie dans l'IMS réservées à la gestion des sessions et du contrôle multimédia. Comme, l'établissement, le monitoring, le support, et la fermeture des sessions multimédia. Ou encore, le transcodage et l'adaptation du contenu multimédia aux terminaux des usagers. Il est aussi responsable de la gestion des interconnexions avec le PSTN-ES pour l'interaction avec les réseaux téléphoniques commutés. Il peut aussi être responsable de la gestion des passerelles multimédias dans la couche transport.

Autres compartiments de l'IMS :

- **Le Serveur d'Applications :** est un ensemble d'entités logicielles faisant office de serveurs de médias. Ces serveurs de médias font aussi office de serveurs SIP pour pouvoir établir des sessions avec les équipements des usagers.

- **Le Home Subscriber Server (HSS) ou le serveur du profile utilisateur :** est un serveur pour une base sécurisée d'informations sur les profiles utilisateurs. Elle est généralement consultée par les habiliteurs de service du sous-système de services.

- **Le Réseau d'accès :** c'est tout réseau à base d'IP pouvant supporter les terminaux IMS. Comme le réseau Internet, les réseaux filaires (DSL, câble modem, Ethernet), les réseaux sans fils (WLAN, WiMax), et les réseaux cellulaires (GSM, GPRS…etc).

- **Equipement d'usager :** ce sont les terminaux IMS qui utilisent le réseau d'accès. Il est impératif qu'il implémente le protocole standard IP et le protocole de session SIP. Ça peut inclure les téléphones portables, les PDA, les ordinateurs…etc.

- **Le NGN core :** c'est le noyau qui gère le réseau d'accès. Son principal rôle est de configurer les équipements des usagers pour les intégrer dans le réseau d'accès. Les équipements des usagers sont organisés sur plusieurs domaines que le NGN core assure l'interconnexion.

5.5.2. Les protocoles utilisés dans l'IMS

La majorité des protocoles utilisés dans l'IMS sont des protocoles standardisés par l'IETF. Dans les sections qui suivent, on va faire un détour sur les plus importants protocoles qu'utilise l'IMS.

Le protocole SIP :

L'un des plus importants protocoles en relation avec l'IMS est sans doute le SIP. Il a été choisi pour l'IMS en grande partie pour son indépendance vis-à-vis de la couche transport, en sachant qu'il a la faculté d'être implémenté quasiment sur tous les protocoles de cette couche, comme le TCP, UDP, ou même le *Stream Control Transmission Protocol* (SCTP). Son mécanisme de fonctionnement est à base de messages textuels (s'inspirant des protocoles HTTP et SMTP) qui décrivent le flux multimédia (adresse, port, type de média, encodage…etc) dans l'objectif de soutenir l'établissement, la modification, et la clôture d'une session multimédia. Des fonctionnalités additives ont été ajoutées au SIP dans le but d'accroitre son interopérabilité avec l'IMS, comme la gestion d'abonnés, prise en charge du QoS, facturation, gestion des ressources…etc. la plupart de ces fonctionnalités ont été définies dans [70-72].

Le protocole Diameter :

Le protocole Diameter [73] fait partie des récents protocoles AAA (Authentication, Authorization, Accounting : Authentification, Autorisation, et Traçabilité) qui remplace l'ancien protocole RADIUS [74]. Ses fondations de sécurités sont basées sur l'un des deux protocoles, l'IPsec ou le TSL. L'IPsec opère sur la couche Réseau du modèle OSI, tandis que le TSL opère sur la couche Transport (TSL est le successeur du SSL). Dans l'architecture IMS, le protocole Diameter est utilisé le plus souvent par le Serveur d'Application et l'IMS core pour la communication des profiles utilisateurs avec le HSS. Il est aussi utilisé dans la communication entre le RACS et le Serveur d'Application.

Le protocole Common Open Policy Service (COPS) :

Le protocole COPS [75] est un protocole qui spécifie un simple modèle client/serveur pour le support des politiques de contrôle pour les *protocoles de signalisation de QoS*. En d'autre terme, il peut fournir des règles de contrôle aux protocoles responsables de la gestion des ressources dans un réseau pour le maintien d'un niveau de QoS donné. On peut mentionner comme protocole de signalisation de QoS, le protocole RSVP [76] (Resource ReSerVation Protocol).

Le protocole MeGaCo (H.248) :

Le protocole MeGaCo [77] ou ***Me****dia* ***Ga****teway* ***Co****ntrol Protocol* est un protocole né de la collaboration des deux organismes de standardisation l'IETF et l'ITU-T. La nomenclature MeGaCo concerne l'organisme IETF tandis que le H.248.1 est sa spécification dans l'organisme ITU-T. MeGaCo est un protocole de contrôle pour passerelles de médias dans l'architecture IMS.

Les protocoles Real Time Protocol (RTP) et Real Time Control Protocol (RTCP):

Comme signalé auparavant, le RTP fournit des fonctionnalités de transport pour la transmission des données en temps réel. Il est généralement utilisé conjointement avec le protocole de contrôle RTCP, afin de permettre un suivi de la livraison des données et fournir un minimum de fonctionnalité pour le contrôle et l'identification.

Le protocole IPv6 :

Il est à noter que dans les dernières versions de spécification des réseaux IMS, l'utilisation de l'IPv6 est obligatoire, quoiqu'il ne soit pas rare de voir des équipements implémentant les deux protocoles IPv4 et IPv6.

5.6. Conclusion

L'intégration de l'Internet Protocole, le protocole le plus utilisé dans les réseaux informatiques dans la diffusion télévisuelle, rend cette dernière plus interactive et plus flexible que jamais. Il devient évident que l'avenir de la Télévision passera par les réseaux IP. Pour l'instant, la Télévision IP n'est qu'à c'est début, et il reste beaucoup de chemin à faire avant de déceler toutes les possibilités offertes par le protocole IP. Toutefois certaines initiatives ont fait leurs apparitions et la notion de télévision par IP prend peu à peu de l'ampleur. L'exemple flagrant de cela est sans doute l'internet TV. En dépit de ses performances jugées faibles, elle offre néanmoins une facilité d'accès accru, avec un simple PC connecté à Internet, l'usager peut accéder à un nombre important de chaines sans demander une installation spécifique. Il y a aussi l'IPTV fournie par les fournisseurs d'accès à internet, qui ressemble beaucoup plus à la télévision usuelle et qui représente le mieux la notion de télévision IP. La première partie de ce chapitre était consacrée à cette dernière, tandis que la deuxième partie faisait un détour sur l'architecture IMS. Cette architecture tend à construire un support normalisé rassemblant tout les média de télécommunication sur la base du protocole IP. IPTV n'en fait pas exception, il fait aussi partie de l'architecture IMS, quoique cette partie du chapitre ne le traite nullement, pour la simple raison que la normalisation de l'IMS est loin d'être mature, et que l'implémentation de l'IPTV n'est qu'a un stade très avancé, seul quelques travaux de recherche en font mention. Dans le chapitre suivant, nous allons aborder une facette toute aussi intéressante, c'est le P2PTV, qui a l'avantage de distribuer le flux multimédia avec une installation matérielle extrêmement légère.

Le P2PTV 6

6.1. Introduction

Le grand déploiement des connexions résidentielles haut débit comme l'ADSL et l'ADSL2 a rendu la TV over IP une opportunité alléchante pour la diffusion de la télévision sur Internet. Le plus usuel dans cette technologie est d'utiliser une architecture client/serveur, dans laquelle un client établit la connexion avec le serveur et le flux vidéo est directement irrigué du serveur vers le client. Cependant cette architecture présente plusieurs inconvénients, dont le manque de flexibilité, l'appétence à la bande passante, une installation conséquente…etc. D'où l'émergence de l'architecture P2PTV. Dans cette architecture et contrairement à l'architecture client/serveur qui s'appuie sur des serveurs et routeurs puissants pour irriguer le flux, c'est le client qui prend cette responsabilité, il prend le rôle du client et du serveur en même temps. De cette manière la structure de l'architecture devient beaucoup plus flexible et moins gourmande en ressources. Ce chapitre, en première partie, va débuter avec quelques définitions et un état de l'art sur les architectures P2PTV. La deuxième partie, plus conséquente sera consacrée à la classification des architectures P2PTV et à l'exposition de quelques-uns des algorithmes les plus connus.

6.2. Définition

Il existe une définition sur le P2PTV [78] selon laquelle "le Peer-to-Peer TV est un moyen de transmettre la vidéo en streaming ou en VoD à l'aide d'une architecture peer-to-peer, dans laquelle un utilisateur qui réceptionne le flux vidéo peut également fonctionner comme un serveur pour les autres utilisateurs. Le système P2PTV est à l'opposé en streaming des systèmes P2P de partage de fichiers tels que le BitTorrent. Dans le P2PTV les utilisateurs regardent les émissions de télévision dont ils sont téléchargés et stockés temporairement pour un affichage quasi instantané plutôt que de télécharger la totalité du fichier pour pouvoir l'utiliser ultérieurement".

En d'autres termes. La philosophie du P2P est d'encourager les utilisateurs à agir autant que clients et serveurs en même temps. Ainsi dans les réseaux P2P, un pair (les utilisateurs dans un réseau P2P sont souvent appelés *pairs*) peut non seulement télécharger les données du réseau, mais il peut aussi les transférer (les uploader) à d'autres pairs sur le réseau. De cette manière, la bande passante sortante (ou la bande passante montante) est utilisée pour réduire la charge totale sur le serveur principal. Le système du P2P a été initialement exploité pour le partage de fichiers, et cette application connait ses derniers temps un grand succès sur Internet [79]. Pour le P2P streaming dont le P2PTV en fait partie, la philosophie du P2P reste la même, quoiqu'elle soit appliquée de telle sorte que le flux de donné partagé entre pairs soit délivré d'une façon séquentielle, avec une restriction sur le temps dans lequel les séquences de flux vidéo doivent être délivrées aux pairs. Cette nouvelle technologie de streaming vidéo commence à prendre de l'ampleur, de part sa facilité de déploiement et ça capacité à utiliser des serveurs de diffusion avec des bandes passantes inferieurs à ce qui ce fait pour la diffusion TV over IP usuelle.

6.3. État de l'art sur les P2PTV

Comme mentionné dans le chapitre précédent, la solution de base pour la TV over IP est l'utilisation de l'architecture client-serveur. Dans cette architecture, le client établit la connexion avec le serveur, et le contenu multimédia est directement irrigué depuis le serveur jusqu'aux clients. Pour la diffusion du contenu multimédia dans l'architecture client/serveur, l'utilisation de l'IP Multicast est certainement la solution la plus probable. La technologie IP Multicast utilise en générale le protocole IGMP, ce dernier opère sur la couche réseau du modèle OSI, et sera dans la plupart du temps implanté sur les nœuds internes du réseau (en d'autres termes, les routeurs) pour pouvoir canaliser le flux multimédia. Cependant, plusieurs chercheurs dont [80] et [81], pensent que la couche réseau n'est pas la couche la plus appropriée pour l'implémentation du multicast. Deux principaux inconvénients en sont la cause. Premièrement le manque de flexibilité, car si un routeur venait à tomber en panne, c'est tous ses fils qui seront démuni du flux multimédia. Et deuxièmement, l'IP Multicast est un service best effort, ce qui rend l'implémentation des services haut niveau tel que la sécurité, la gestion de la congestion ou la gestion du flux, relativement difficile. De ce fait, la majorité des chercheurs ont opté pour une architecture avec un Multicast au niveau application. Dans cette dernière, le réseau dit *overlay network* (surcouche réseau) est constitué de plusieurs nœuds faisant office de nœuds Multicast. Le Multicast est effectué au niveau application tandis que la communication direct nœud à nœud se fait par de simple connexions IP unicast. La structure du réseau est ainsi construite au tour de nœuds implémentant le Multicast au niveau application et communicant au niveau réseau.

Les architectures P2PTV peuvent être classifiées selon la structure de la surcouche réseau en deux catégories. Les architectures basées arbre et les architectures basées mailles. L'architecture basée arbre suit la même structure que celle de l'IP Multicast, elle construit son réseau en forme d'arbre, et transmet généralement le flux vidéo du pair à ses fils par la stratégie *push*, dont laquelle la transmission se fait du père au fils sans demande de préavis à ce dernier. Contrairement à la stratégie *pull* généralement utilisé par les architectures basées mailles, dans laquelle la transmission des donnés du transmetteur vers le récepteur ne se fait qu'avec la demande explicite du récepteur. L'inconvénient majeur des architectures basées arbre par rapport aux architectures basées mailles est leur sensibilité aux déconnections. La rupture d'un nœud dans le réseau va temporairement suspendre la diffusion de la vidéo pour tous les pairs dans le sous-arbre subventionné par le nœud rompu, et un laps de temps relative au nombre de nœuds du sous-arbre est nécessaire pour la restructuration globale de l'arbre. Dans les architectures basées mailles, les pairs s'allient en général sous forme de groupes, dans lesquels ils ne sont pas limités par une structuration statique comme celle de l'architecture arbre, mais plutôt à une constriction/rupture de liens entre pairs basés sur plusieurs critères selon l'algorithme utilisé, comme par exemple, la disponibilité/absence du contenu multimédia, l'abondance en bande passante…etc. La classification des architectures P2PTV sera plus détaillée dans la section suivante.

6.4. Classification des architectures P2PTV

Comme signalé auparavant, les systèmes P2PTV peuvent être classifiés en deux catégories, le P2PTV basé arbre et le P2PTV basé mailles. Une sous-classe de la catégorie basée arbre peut aussi être différentiée, c'est la classe multi-arbres ou foret.

6.4.1. Les P2PTV basés arbre

Le P2PTV basé arbre s'inspire de l'architecture d'un arbre de diffusion IP multicast (formé par les routeurs au niveau réseau). Dans le P2PTV basé arbre, les utilisateurs participants à une session de diffusion vidéo forment un arbre à la couche application, dans laquelle chaque utilisateur s'inclut à un certain niveau de l'arborescence. Il reçoit du flux vidéo à partir de son père se trouvant à un niveau supérieur et distribue le flux vidéo pour ses enfants dans le niveau inferieur.

Pour la construction d'un arbre avec un nombre donné de pairs, il existe un nombre élevé de possibilités pour composer le graphe de l'arbre qui relit tous les pairs. Chaque algorithme P2PTV utilise sa propre stratégie pour le faire, bien que la majorité des algorithmes se focalisent sur les deux principaux facteurs qui sont, la profondeur de l'arbre (le nombre de niveaux dans l'arbre) et le débit de diffusion des nœuds internes (ou le nombre de fils que peut porter un nœud interne, qui est en relation avec la bande passante montante), pour la principale raison qu'un schéma de construction avec le minimum de niveaux dans un arbre de diffusion vidéo, a l'important avantage de minimiser les délais de transmission du flux. Toutefois, il est évident que ces deux facteurs sont en étroite relation, car pour que la topologie de l'arbre soit la moins profonde possible, elle doit être aussi large que possible pour chaque niveau, ce qui implique que le débit de diffusion des nœuds internes doit être exploité au maximum. Toutefois, dans le but de produire un équilibrage de charge correct et une tolérance aux fautes acceptable, chaque algorithme P2PTV pèse sur le nombre maximum qu'un nœud interne peut supporter comme enfants.

En ce qui concerne la tolérance aux fautes, si un nœud venait à quitter l'arbre, c'est toute sa descendance qui serait démunie de flux vidéo. Dans ce cas, un processus de maintenance se déclenche pour la réaffectation des nœuds orphelins. Le problème avec les systèmes P2PTV c'est l'instabilité temporelle de ses nœuds (ce phénomène est appelé *churn* en anglais), on ne sait jamais quand un nœud va quitter l'arbre. En plus il existe deux sortes d'abandons, le premier qui est relativement cerné par les algorithmes de maintenance, puisque le nœud dans ce cas quitte l'arbre d'une façon normale (quand l'usager éteint son terminal par exemple), et déclenche un signal d'abandon que l'algorithme de maintenance peut récupérer. C'est le deuxième cas qui pose problème, étant donné que le nœud dans ce cas quitte l'arbre d'une façon anormale (coupure de connexion ou défaillance du nœud par exemple). L'algorithme de maintenance peut le détecter par l'absence du signal de vie, que les nœuds lui renvoient périodiquement. Le problème, c'est que si la période de signal de vie est trop petite, le flux de contrôle devient trop encombrant, et si cette dernière est trop grande, ça risque de prendre trop de temps avant le déclanchement de l'algorithme de maintenance.

Nous allons voir dans ce qui suit quelques aperçus de systèmes P2PTV basé arbre :

SpreadIt - L'architecture SpreadIt est exposée dans [82,83], son algorithme vise à construire un arbre multicast au niveau application en utilisant une constitution entièrement distribuée. Chaque nœud dans le réseau entretient des informations locales sur les autres nœuds, ça se résume en général par le nœud racine, son nœud parent, et ses enfants directs. Quand un nouveau nœud veut adhérer à l'arbre multicast, il envoie une requête à tous les nœuds déclarés dans l'arbre, si un nœud reçoit une demande d'adhésion, il vérifie s'il n'est pas saturé, ainsi il accepte la demande et érige un canal pour transmettre le flux au nouveau nœud. Dans le cas contraire, il transmet la demande à d'autres pairs (en d'autres termes a son nœud parent et ses enfants immédiats). On ce qui concerne l'algorithme de maintenance, il utilise une politique relativement basique pour le maintien de la topologie de l'architecture, elle s'appuie principalement sur une politique de redirection de requêtes, ainsi si un nœud quitte l'arbre multicast, une requête est diffusée et redirigé vers les nœuds concernés.

Narada - Le protocole Narada est décrit dans [84,85]. Narada tend à toucher un large éventail d'applications, tels que la IPTV, le streaming média et les jeux en réseau. Narada construit l'arbre multicast dans un processus en deux étapes. Tout d'abord, il construit un graphe fortement connecté, appelée maille. La maille est construite en exigeant que chaque pair génère périodiquement un rafraîchissement de messages à ses voisins dans le but de maintenir une meilleure appréciation des métriques et défaillances du réseau. Dans la deuxième étape, Narada construit un arbre couvrant (c'est un arbre qui se compose d'une sélection d'arêtes du graphe qui forme un arbre couvrant tous les sommets du graphe) dans la maille qui va faire office d'arbre multicast. En raison des besoins considérables d'interactions entre nœuds, Narada ne peut cibler que des réseaux de petite ou moyenne taille, ils ne conviennent pas aux réseaux de grande échelle.

Nice - est un protocole décrit dans [86], conçu pour les applications multicast hiérarchique. Dans Nice, les nœuds sont partitionnés en un ensemble de clusters. Chaque cluster est de taille entre k et 3k-1 nœuds (k est un nombre constant donné comme paramètre à l'algorithme). Chaque cluster a un nœud leader du cluster, il est celui qui a le minimum de distance maximale par rapport à tous les autres nœuds du cluster. Ces clusters forment le niveau-0 (le plus bas) de l'hiérarchie de l'arbre. Les leaders de chaque cluster du niveau-0 forment les nœuds du niveau-1. Ils sont groupés dans des clusters avec la même stratégie utilisée pour le niveau-0 (la taille du cluster est toujours entre k et 3k-1). Les niveaux supérieurs de la hiérarchie sont construits récursivement de la même manière. Il ne faut pas oublier que le fait de joindre et d'enlever un nœud va changer la taille du groupe, de ce fait le protocole Nice prévoit un algorithme de maintenance dynamique pour garder l'équilibre dans les clusters.

Il faut signaler aussi que les systèmes P2PTV possèdent un autre inconvénient majeur par rapport aux autres systèmes P2PTV, c'est que les nœuds terminaux ou les feuilles de l'arbre ne contribuent pas avec leurs bandes passantes sortantes à l'accroissement de la bande passante globale du système. Dans le cas usuel, les nœuds terminaux représentent une grande partie du nombre total des pairs dans un système P2PTV, ça signifie que dans ce genre de système, une grande partie de la bande passante est perdue inutilement. Cette constatation est en grande partie la raison d'être des systèmes P2PTV basés foret.

6.4.2. Les P2PTV basés foret

Pour combler les lacunes des systèmes P2PTV basés arbre, plusieurs solutions basées foret ont été proposées. Nous allons voir splitStream [87] et CoopNet [88] comme illustration dans les sections qui suivent. Dans l'approche foret le flux multimédia est subdivisé en plusieurs sous-flux (si par exemple en prend un flux subdivisé en deux sous-flux, le premier aura le flux des paquets pairs du flux original, et le deuxième comprendra les paquets impairs), et au lieu d'avoir un seul arbre, plusieurs sous-arbres seront créés pour chaque sous-flux. Un nœud fait partie de chaque sous-arbre crée, quoiqu'il prenne diverses positions dans chaque sous-arbre. Il peut être positionné autant que nœud interne ou nœud terminal, néanmoins pour des propos d'optimisation, la majorité des systèmes basées foret fons en sorte que le nombre de fois où le nœud doit paraitre autant que nœud interne, est en relation avec sa bande passante sortante, elle doit être utilisée au maximum. De ce mécanisme, chaque nœud recueille les différents sous-flux des différents sous-arbres qu'il appartient, pour assembler les sous-flux et synthétiser le flux original.

Pour mieux illustrer le fonctionnement des systèmes P2PTV basés forêts, on va utiliser un exemple fictif dans lequel le réseau se compose de 7 nœuds plus un serveur, avec une bande passante sortante pour chacun de 2 mégabit. Le serveur partitionne le flux multimédia en deux sous-flux de 1 mégabit pour les déverser aux deux sous-arbres représentés sur la *figure*6-1. On peut voir que les nœuds 1, 2, et 3 sont des nœuds internes pour le sous-arbre de droite, alors qu'ils sont des nœuds terminaux pour le sous-arbre gauche. De même pour les nœuds 4, 5, et 6. Ils sont comme nœuds internes dans le sous-arbre gauche et nœuds terminaux dans le sous-arbre de droite. Le flux vidéo est de 2 mégabits, chaque pair reçoit 1 mégabit des deux sous-arbres. Il les assemble pour obtenir le flux complet.

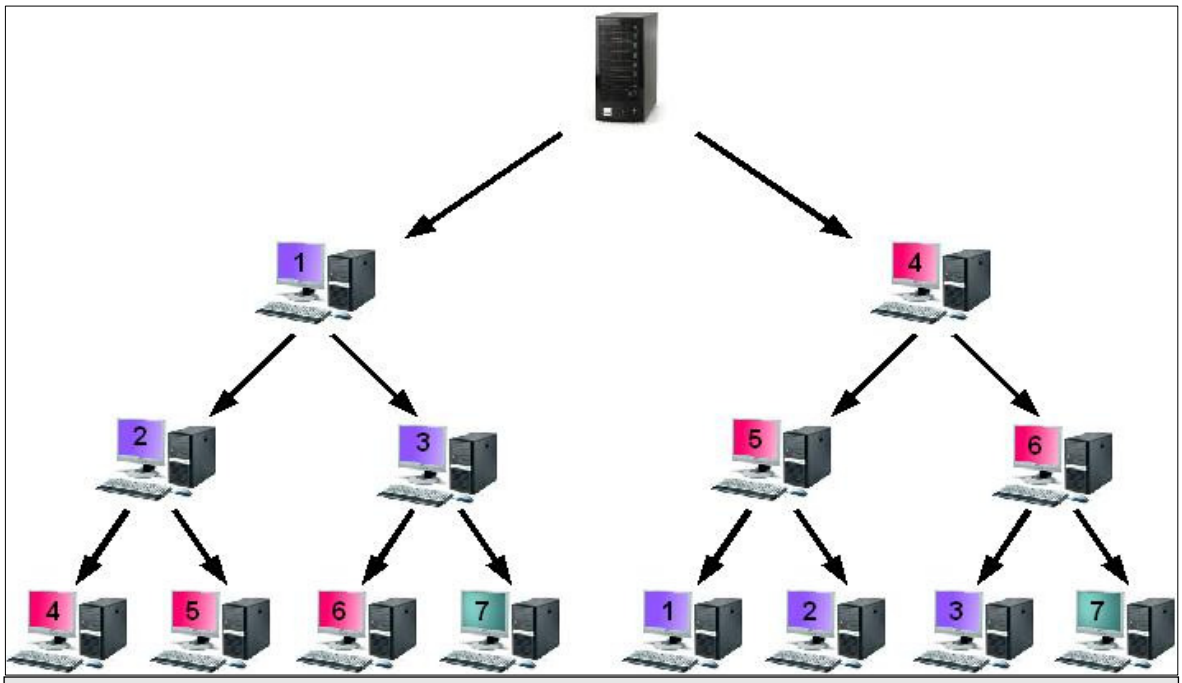

Figure 6-1 : *Exemple d'une architecture P2PTV basée foret.*

SplitStream – est un système P2PTV basé foret, dans lequel le flux de donné est subdivisé en plusieurs bloques disjoints appelés *stripe*. Comme c'est usuel dans les architectures basées foret, un nœud doit appartenir à chaque sous-arbre, et chaque sous-arbre est affecté a un stripe. La différence dans splitStream par rapport aux autres architectures foret, c'est que le nœud est assigné autant que nœud interne à un seul sous-arbre, tandis qu'il est comme nœud terminal dans les autres sous-arbres. De cette manière, l'algorithme assure une certaine équité, dans le sens où un nœud n'a de responsabilité que sur un seul stripe. Cette manière d'affecter les nœuds garantie aussi une certaine tolérance aux fautes, puisque si un nœud venait à quitter le système, au pire un seul stripe est affecté.

CoopNet - est un protocole relativement différent des autres protocoles basés foret dans la mesure où il utilise les MDC (Multiple Description Coding) pour délivrer le flux vidéo à ses clients. Le MDC [89] est une technique de codage pour flux vidéo qui segmente ce dernier en plusieurs sous-flux vidéo appelée description. Pour la lecture du média n'import quelle description peut être utilisée, quoique la qualité du média se dégrade si toutes les descriptions ne sont pas réunies. L'avantage du MDC par rapport à un découpage du flux usuel c'est sa tolérance aux fautes. Même avec un nombre réduit de descripteurs, la construction du flux original ne sera pas interrompue et le flux continue avec une perte de qualité. Dans CoopNet la gestion des sous-arbres repose sur une approche centralisée impliquant un ensemble de nœuds serveurs (ou le nœud source).

6.4.3. Les P2PTV basés mailles

Comme indiqué auparavant. Dans les architectures basées arbres, un nœud a seulement un seul parent pour télécharger tout son contenu vidéo (ou une partie du flux vidéo dans les

architectures foret). Ainsi, si ce parent venait à quitter l'arbre, c'est tous ses descendants qui seront démunis du flux vidéo, et ne peuvent plus l'avoir que par leurs réaffectations à d'autres parents. Pour éviter ce genre d'inconvénient, plusieurs travaux de recherches dont DONet[90] et Chainsaw[91] ont adopté une toute autre approche, dans laquelle un pair n'est plus contraint de récupérer le flux d'un seul nœud mais de plusieurs. Dans l'approche basée mailles il n'y a plus de structure statique, le pair construit et détruit ses connexions d'une façon totalement dynamique. Un pair détient un nombre donné de partenaires (appelés aussi voisins), avec lesquels il peut télécharger ou transmettre du contenu vidéo. Dans le cas où un voisin viendrait à quitter sa liste de partenaires il peut tout aussi télécharger le contenu média d'autres pairs. Le degré d'un pair est le nombre de voisins que peut contenir sa liste, avec un grand nombre de voisins le système P2PTV devient extrêmement robuste aux churn. Cette constatation a été signalée dans l'étude de comparaison entre les systèmes basés arbre et les systèmes basés mailles [92].

Pour le maintien d'une topologie avec une architecture P2PTV basé mailles, deux méthodes sont généralement utilisées, une méthode centralisée et une méthode distribuée. Dans la méthode centralisée le système tourne au tour d'un serveur qui détient les informations sur tous les pairs du système. Quand un nouveau pair rejoint l'architecture, il envoie ses propres informations au serveur, (adresse IP, Port…). Le serveur enregistre ces informations dans sa base de données, et retourne ensuite à ce dernier une liste de pairs choisies aléatoirement dans l'ensemble des pairs du système. Le pair qui reçoit cette liste et en relation avec son ordre, va choisir un nombre de pairs parmi la liste fournie pour les assigner comme voisins. Après l'établissement des connexions avec ses voisins, il peut commencer l'échange de données avec ses derniers. En ce qui concerne l'arrivé et l'abandon des pairs, en général les pairs rafraîchissent leurs listes régulièrement en invoquant le serveur, ou en échangeant leurs listes avec les listes de leurs voisins. Si un pair quitte le système normalement, il envoie au serveur un message d'abandon de telle sorte que ses informations peuvent être supprimées du serveur. Dans le cas contraire, la détection de l'abandon se fait par l'arrêt d'envoi des signaux de vie au serveur par le pair déconnecté du système. La méthode distribuée sera exposée dans la section réservée à DONet.

Le transfert de la vidéo dans les architectures basés mailles se fait généralement par le découpage de la vidéo en plusieurs morceaux, chaque morceau détient une petite partie de la vidéo, 1 seconde de vidéo pour chaque morceau par exemple. Ces morceaux sont numérotés séquentiellement dans le temps, avec les numéros séquentiels les plus bas pour les morceaux les plus antérieurs dans le flux vidéo, chaque morceau est ainsi disséminé pour tous les pairs à travers le système. En sachant que les morceaux prennent des chemins distincts, le plus probable c'est qu'ils arrivent selon un ordre aléatoire. Pour que le pair puisse lire le flux vidéo il doit les stocker d'une façon séquentielle dans un buffer pour que le lecteur du pair puisse les assembler et les lire sous forme d'un flux vidéo. Sur la *figure*6-2 on peut voir que le Buffer suit le flux par l'intervalle de N morceaux, l'intervalle N est généralement appelé fenêtre, et la quête du pair ne se tient qu'aux morceaux du flux dans la limite de cet intervalle. Une tout autre structure avec autant d'importance dans le système entre en jeu, c'est le Buffer Map. Le Buffer Map est une simple structure de donnée représentant l'état du Buffer. À vrais dires, c'est une variable de N bites, chaque bite de la variable indique la présence ou l'absence d'un morceau du flux vidéo dans le Buffer (1 = présence, 0 = absence). Le Buffer Map contient aussi une autre variable qui représente le numéro séquentiel du premier morceau dans le Buffer, ça permet de connaître l'état de la lecture du pair.

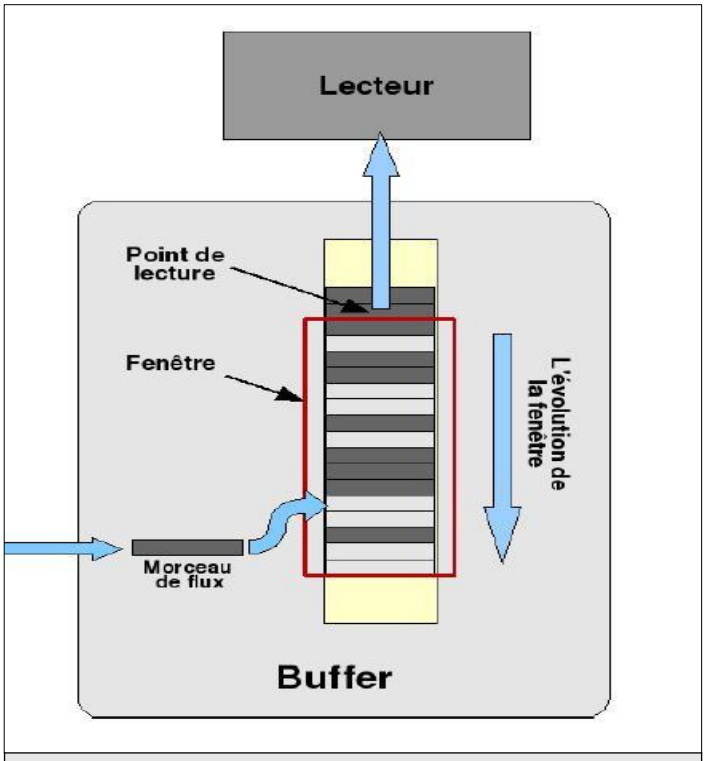

Figure 6-2 : *Illustration du Buffer d'un nœud dans une architecture P2PTV basée mailles.*

Nous avons abordé précédemment les deux stratégies de transfert de flux vidéo, la stratégie push et la stratégie pull. On a mentionné que les architectures basées arbre utilisent généralement la stratégie push, en raison de la nature mono-direction (du père vers le fils) de la structure arbre. Pour les architectures basées mailles, la relation père-fils est inexistante, et une stratégie push peut poser problème, pour la principale raison qu'un pair peut aveuglement transmettre un morceau vidéo que le voisin dispose déjà. De ce fait, la stratégie push a le sérieux inconvénient de gaspiller la bande passante montante des pairs avec la redondance des morceaux vidéo, par conséquent cette dernière est peu utilisée dans les architectures basée mailles (comme exemple l'employant on peut citer le protocole GridMedia [93]). La stratégie pull tend à régler ce problème, puisque c'est le pair en question qui exprime explicitement sa demande du morceau vidéo dont il a besoin. Pour qu'un pair puisse savoir la répartition des morceaux lui manquant parmi ses voisins, les pairs dans un système P2PTV s'échangent régulièrement leurs Buffer Map, ça permet premièrement d'avoir une appréciation sur la distribution des morceaux vidéo parmi leurs voisins, et deuxièmement de connaitre l'état fonctionnel d'un pair, étant donné qu'un pair qui n'émet pas son Buffer Map présente probablement une anomalie. L'inconvénient de la stratégie pull par rapport à la stratégie push est sans doute le surplus en messages de signalisation (les Buffer Map et les requêtes de sollicitations de morceaux).

Dans les systèmes P2PTV basés mailles, il existe deux entités d'une grande importance, la différentiation entre les protocoles basés mailles se fait généralement sur ces deux entités. La première est le *sélecteur de voisins*, c'est le compartiment responsable de la sélection, la connexion, la déconnexion des voisins. Sa stratégie pour effectuer ces opérations se base principalement sur la disponibilité des ressources comme le nombre de connexions dans un pair, la bande passante montante et descendante, le CPU, la mémoire…etc. et la qualité de la connexion avec le voisin comme le délai, le jitter…etc. D'après ces critères un sélecteur de voisin peut choisir le voisin qui

convient le plus au pair. La deuxième entité est le *scheduler*, que ça soit pour la strategie push ou la strategie pull, son algorithme est le principal facteur de performance pour un système basé mailles. Son rôle est de choisir le voisin dont le pair doit télécharger le morceau vidéo pour les stratégies pull, et le voisin dont le pair doit lui transmettre le morceau vidéo pour les stratégies push. En général, le scheduler se base sur les critères suivants : La disponibilité du morceau parmi ses voisins, le deadline du morceau vidéo (le temps où le morceau doit être joué), la bande passante du voisin…etc. Comme exemple de système P2PTV basé mailles, le système DONet sera plus ou moins détaille dans ce qui suit.

Le protocole DONet

DONet est l'un des protocoles P2PTV basé mailles les plus connu dans le domaine de la recherche, il a été décrit dans [90] et [84]. Pour le maintien de sa topologie, DONet utilise une méthode distribuée appartenant aux méthodes *épidémiques*, plus précisément il utilise le protocole Scalable Gossip Membership protocol (SCAMP) [95]. Le principe de base dans le protocole SCAMP est qu'un nœud diffuse un message pour un ensemble de pairs choisis aléatoirement, ces derniers font la même chose pour d'autres ensembles de pairs et ainsi de suite, jusqu'à ce que le message soit distribué à toute la population du système. L'utilisation d'une distribution aléatoire par rapport à une distribution structurée a l'avenage d'être très robuste aux défaillances aléatoires des pairs, et permet une décentralisation quasi uniforme. La *figure*6-3 schématise le diagramme fonctionnel des principaux compartiments d'un nœud DONet. L'accent sera mis sur les trois principaux compartiments : le gestionnaire de pairs, le sélecteur de voisins, et le scheduler.

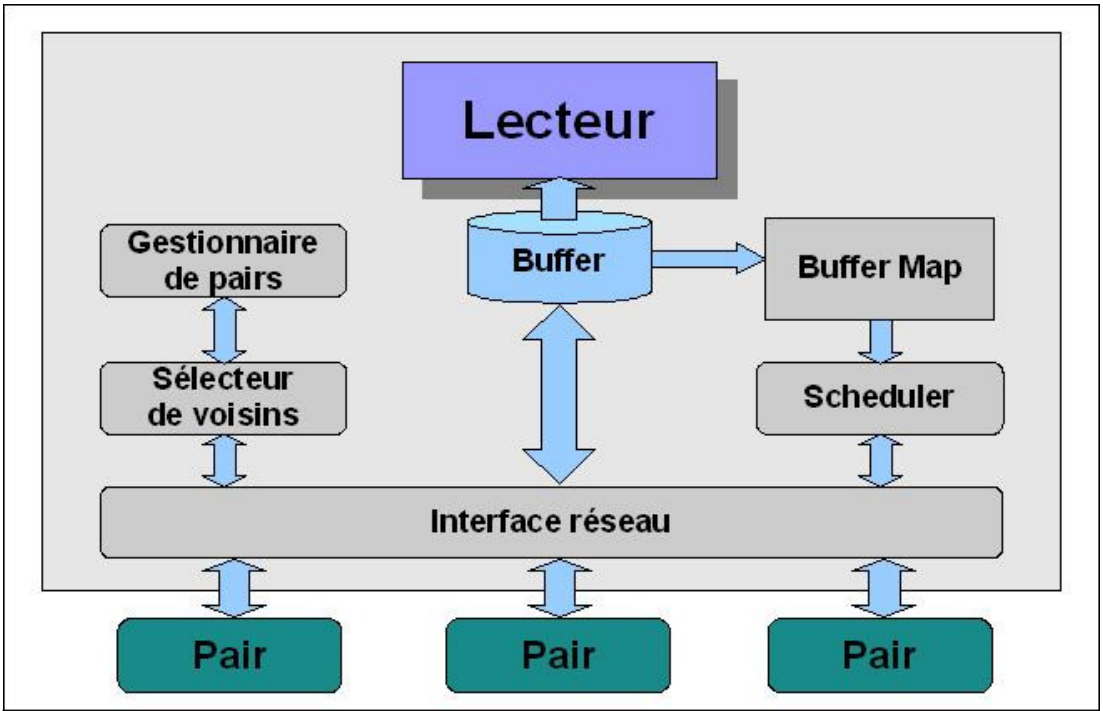

Figure 6-3 *: Diagramme fonctionnel dans un nœud DONet.*

Le gestionnaire de pairs permet aux nœuds d'avoir une vue partielle sur les autres nœuds du système, parmi lesquels il peut choisir un nombre donné comme voisins. Chaque nœud se caractérise par un identificateur unique (son adresse IP par exemple), et maintient un cache pour la liste des identificateurs des nœuds actifs dans le système. Si un nœud venait à joindre le système, il contacte premièrement le nœud racine qui va choisir aléatoirement parmi la liste des nœuds du système un nœud délégué. Le nouveau nœud va acquérir le cache de ce dernier et contacte les nœuds de ce cache pour former sa liste de voisins. En raison de la dynamique du réseau, un pair dans système DONet génère périodiquement un message pour annoncer son existence, chaque message est constitué comme suite <num-séq , ID , nombre-voisins , time-to-live>, où num-séq est le numéro séquentiel du message, l'ID est l'identificateur du nœud, nombre-voisins est son nombre de voisins, time-to-live est le temps qui reste au message avant sa péremption. Quand un nœud reçoit un message avec un nouveau numéro séquentiel, il met à jour son entré dans le cache, dans le cas où le message arriverait avec un nouveau ID, il crée une nouvelle entrée. Une entrée dans un cache est constituée de cinq champs <num-séq , ID , nombre-voisins , time-to-live , temps dernière mis a jour>, où les 3 premiers champs sont copiés du message reçu, le cinquième est le temps locale du message antérieur, et le time-to-live est décrémenté par le *temps local - temps dernière mis a jour*.

Le sélecteur de voisins est le compartiment responsable de la formation et le maintien des liens du pair avec ses voisins. Dans le protocole DONet, un pair raffine sa liste de voisins périodiquement dans le but de garder un nombre stable de voisins et de chercher des voisins de meilleure qualité. La qualité dans DONet est calculée par la fonction $MAX(S_{i,j} , S_{j,i})$, où $S_{i,j}$ est la moyenne du nombre de morceau vidéo que le nœud i reçoit du nœud j par unité de temps. En d'autres mots, un pair choisit son voisin en fonction du débit entrant ou sortant de ce dernier. Il est probable qu'un pair rejette un voisin de sa liste si ce dernier ne présente pas de bonnes qualités par rapport aux autres pairs.

Le scheduler est responsable de la planification de la transmission vidéo entre pair et voisins. Il possède comme donné principal dans son entré, l'ensemble des Buffer Map de ses voisins. De cette entée le scheduler génère le planing des morceaux vidéo qui doivent être téléchargés et de quel voisin. Dans le cas général, les algorithmes pour scheduler visent à résoudre un problème d'optimisation mathématique, dont les contraintes sont le deadline des morceaux vidéo et l'hétérogénéité des bandes passantes pour chaque voisin, si la première contrainte ne peut pas être satisfaite, le nombre de morceaux manqués doit être tenu au minimum. Ce genre de problème est connu généralement sous le nom de *Parallel machine scheduling*, ce sont des problèmes NP-hard (Non-deterministic Polynomial-time hard) [96], et une solution canonique prend trop de temps pour des réseaux réputé dynamique. Ainsi, des heuristiques sont généralement utilisées. Pour le DONet, l'heuristque est relativement simple, étant donné que le morceau avec le moins de voisins est le plus probable de manquer son deadline, l'algorithme détermine le fournisseur de chaque morceau en commençant par ceux avec un seul fournisseur potentiel, puis ceux avec deux, et ainsi de suite. Parmi les multiples fournisseurs potentiels, celui qui est avec la plus grande bande passante et le moins de charge est sélectionné. La complexité de l'algorithme est de $O(B^2 \times M)$, où B est le nombre de bits dans un Buffer Map, et M est le nombre de voisins du pair.

6.5. Conclusion

Le P2PTV est une technologie pour la télévision-over-IP qui présente de bonnes caractéristiques pour la flexibilité, le support des serveurs avec un faible débit, et une installation à faible coût. Les topologies P2PTV sont sommairement catégorisées en deux genres, les architectures à bases d'arbres et les architectures à bases de mailles. Les architectures à base d'arbre par rapport aux architectures à base de mailles ont l'avantage d'avoir un faible délai (la différence de temps entre la lecture du média sur le serveur et la lecture sur les pairs) et l'inconvénient d'être moins robuste aux churn. Tout au long de la partie réservée à la classification des différentes architectures, chaque classe était illustrée par quelques algorithmes des plus célèbres dans le domaine de la recherche. L'algorithme DONet était plus au moins détails par rapport aux autres en raison de sont utilisation ultérieure dans ce mémoire.

Spécification de l'SBC APF9328 7

7.1. Introduction

La carte mère *APF9328* est un SBC muni d'un processeur ARM9 tournant à 200 MHz et un FPGA de 200 k portes, fonctionnant sous Linux et commercialisé par la société Armadeus Systeme [97]. Le développement de l'SBC est maintenu par l'association non lucrative Armadeus Project [98]. Cette dernière maintient le soft, les sources, les packages et même les schématiques sous licence GPL. D'après la société Armadeus Système, l'APF9328 cible les applications de consommation électrique minime, comme les dispositifs de communication, les dispositifs de contrôle industriel, et ce qui concerne notre cas, les applications multimédias. L'SBC APF9328 fait partie de la catégorie des Macrocomponent, une carte Macrocomponent contient l'essentiel des fonctionnalités comme le processeur, la mémoire vive, la ROM, et quelques contrôleurs de périphériques indispensables, condensé sur une petite carte électronique. Cette dernière est branchée sur un circuit imprimé lui servant d'hôte. La société Armadeus Système fournit une carte hôte suivant la norme industrielle EPIC, le nom de cette carte est APF9328DevFull et vise principalement le prototypage d'application embarqué destiné à l'APF9328, elle comporte essentiellement les ports de sorties des connectiques que l'APF9328 peut disposer. Ce chapitre peut être sommairement scindé en deux parties, la première concerne la description de l'architecture APF9328 et de ses différents composants, une grande partie de celle-ci s'attarde sur le processeur, étant donné que c'est l'élément le plus riche en fonctionnalité sur toute la carte. La deuxième partie, concerne la carte hôte APF9328DevFull, et traite principalement les composants additifs que cette dernière ajoute à l'APF9328.

7.2. Nomenclature de l'APF9328

La référence exacte du SBC utilisé dans notre mémoire est : APF9328-200-M16F16-AEF200. La signification de ses différentes parties est expliquée sur la *figure*7-1. Donc, d'après cette dernière, on peut constater que notre SBC dispose d'un processeur à 200 MHz, d'une mémoire SDRAM de 16 Mo, d'une mémoire Flash de 16 Mo, d'un contrôleur Ethernet, des convertisseurs Analogique/Numérique Numérique/Analogique, et d'un FPGA de 200 k portes.

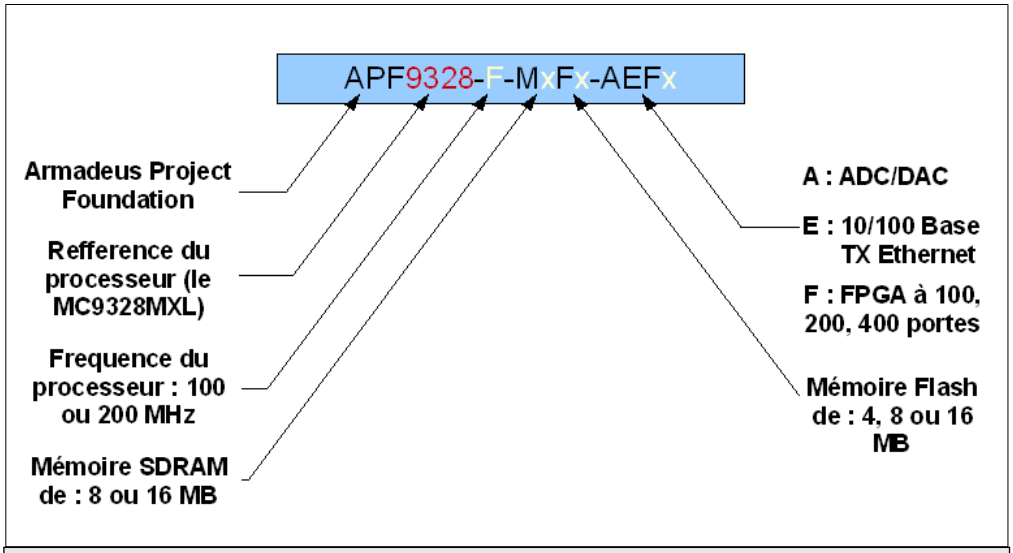

Figure 7-1 : *Nomenclature de l'SBC APF9328.*

7.3. Spécificité mécanique

L'SBC APF9328 expose un encombrement de 71 mm x 39 mm (2.8 pouces x 1.5 pouces). La face supérieure (illustrée sur la *figure*7-2) contient trois importants circuits intégrés : le circuit du processeur, le circuit de la mémoire centrale et le circuit de l'FPGA, elle détient en plus un circuit DAC (Convertisseur Numérique/analogique) et deux régulateurs pour soutenir l'alimentation de l'FPGA. La face inferieur (illustrée sur la *figure*7-3) maintient le contrôleur Ethernet et le circuit de la Mémoire Flash, ainsi que deux connecteurs lui permettant d'être enfiché sur l'APF9328DevFull. Sans oublier le ADC (Convertisseur Analogique/numérique) et les deux émetteurs/récepteurs USB et RS232 qui ont pour rôle d'assurer la fonction d'interfaçage entre les broches USB et UART du processeur avec les ports USB et RS323 implantés sur l'APF9328DevFull. Les caractéristiques de tous ces composants seront plus approfondies dans les sections qui suivent.

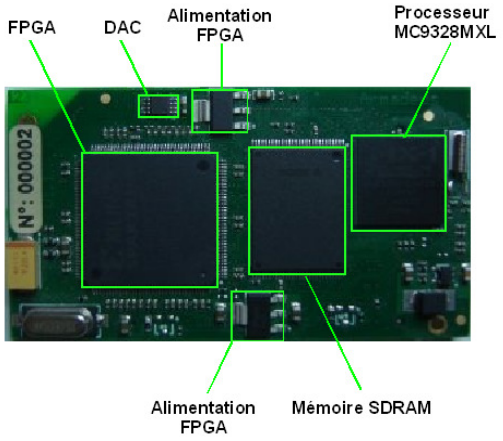

Figure 7-2 : *L'SBC APF9328 vue de face.*

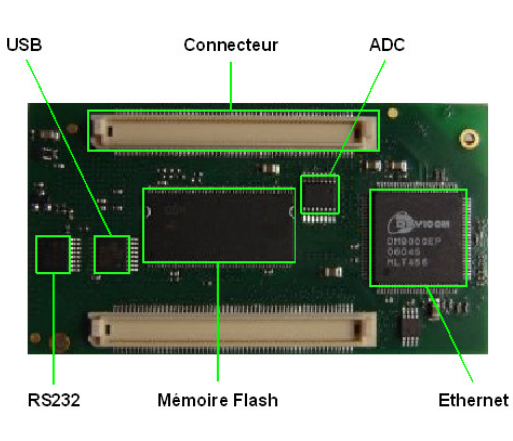

Figure 7-3 : *L'SBC APF9328 vue de dos.*

7.4. Architecture matérielle

Cette section donne une perception plus ou moins détaillée sur l'architecture de l'SBC APF9328 [99] ainsi que sur les différents éléments qui la composent. Sur La *figure*7-4 on peut apercevoir les différents composants ainsi que leurs interactions avec le processeur. Au fait, on peut facilement remarquer que le processeur est au centre de l'architecture et toutes les interactions passent par ce dernier. Car contrairement aux architectures PC qui utilisent leurs shipset pour assurer la connectivité avec les périphériques, dans les architectures embarquées, c'est le processeur en général qui assume la fonctionnalité de gestion des périphériques. La raison de ce fait est assez économique, en sachant que le tout intégré revient forcement moins chère dans une production en série que le tout séparé. Le processeur MC9328XML ne déroge pas à cette règle, et présente une grande richesse connectique. La connectique processeur/contrôleur dans un système embarqué est généralement subdivisée en deux catégories, premièrement la connectique dense où le processeur nécessite un grand flux de communication, dans la majorité des cas c'est le bus d'adresse/donné/contrôle qui est impliqué. Notre architecture utilise ce bus pour la mémoire centrale, la mémoire flash, l'FPGA et l'Ethernet, ainsi que pour le contrôleur USB additif sur la carte APF9328DevFull. La seconde catégorie est celle des connecteurs usuels, dans laquelle les connecteurs sont directement liés aux ports externes de la carte. Dans notre cas ce sont les bus I^2C, UART, SPI, CSI, LCD, SSI, USB, MMC/SD, TIMER, PWM. Il n'est pas rare de voir ces bus utiliser un composant passif (qui n'a pas de faculté de calcule) intermédiaire entre le port externe et le processeur, le composant passif a dans la plupart des cas un rôle de régularisation électrique. C'est le cas pour le port USB et UART. Il est aussi très fréquent de trouver des composants actifs entre la liaison processeur-port externe, dans ce cas le bus a une faculté de contrôle, en d'autres mots, le processeur utilise son bus pour contrôler un composant de l'architecture. On peut voir ce cas de figure avec le DAC et la ADC qui sont respectivement contrôlés par les bus I^2C et SPI, ainsi que les contrôleurs audio et vidéo implantés sur la carte APF9328DevFull qui sont contrôlés par le processeur de la même manière. Une description de la majorité des composants du SBC APF9328 est donnée dans les sections qui suivent.

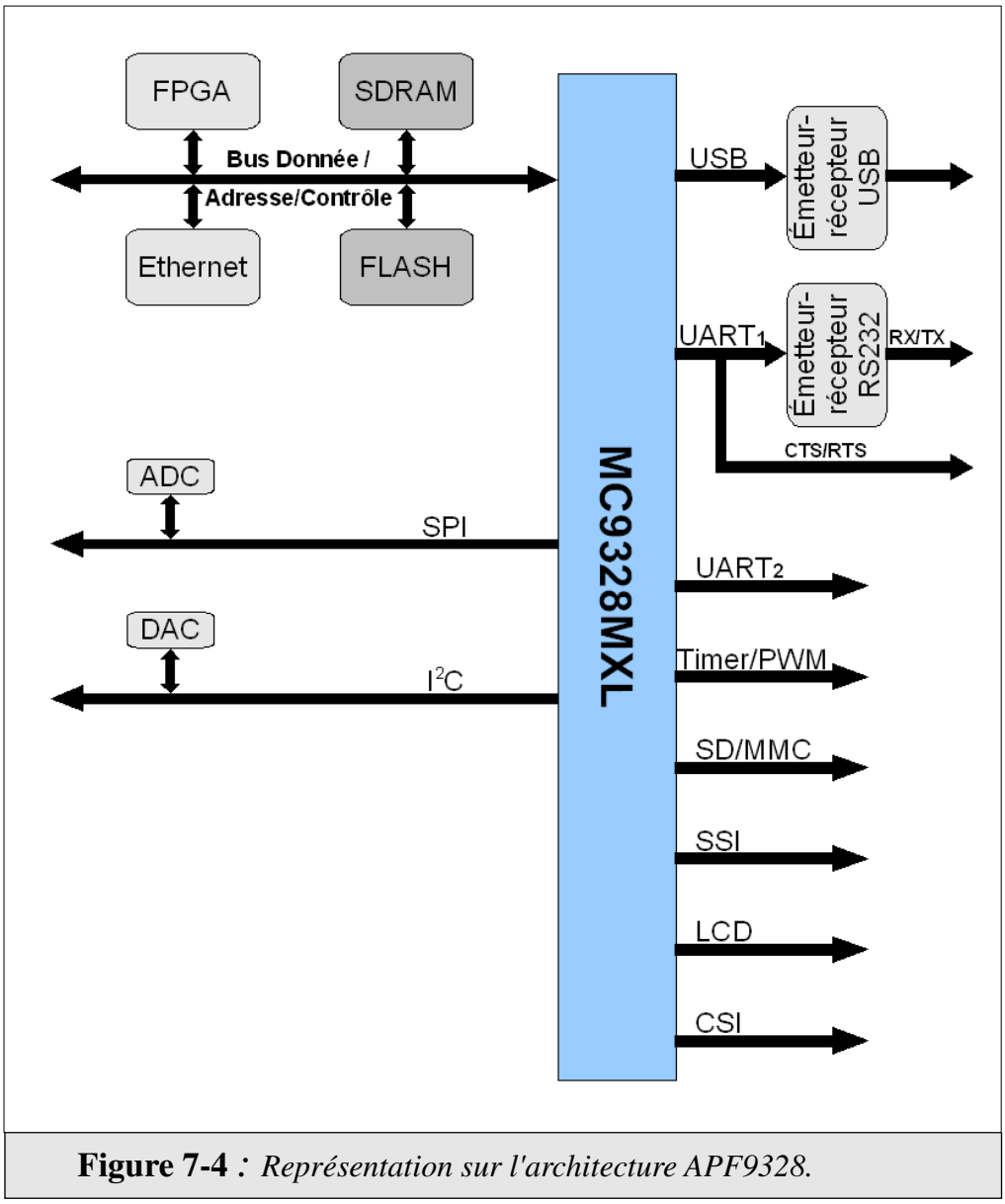

Figure 7-4 : *Représentation sur l'architecture APF9328.*

7.4.1. Le processeur

L'APF9328 est équipé d'un processeur MC9328MXL [100] du fabricant Freescale appartenant à la série des i.MX (Media eXtention : la gamme orientée multimédia). Dans la série des i.MX le MC9328MXL est référencé par i.MXL et c'est souvent l'appellation la plus utilisée dans la littérature des processeurs embarqués. Le noyau du processeur est en architecture ARM9 avec une vitesse d'horloge programmable de 0 à 200 MHz, et un bus système programmable de 0 à 100 MHz. D'après la *figure*7-5, le processeur se subdivise en six principaux blocs fonctionnels : le principal bloque est le noyau, qui contient l'ARM9, deux mémoires caches, et des fonctionnalités importantes responsables de la gestion du processeur. Un bloc de contrôle système responsable de la gestion de l'alimentation, le débogage, et le bootstrap. Un bloc pour les Entrées/Sorties standards. Un bloc pour la connectivité. Un bloc pour le multimédia. Et finalement un bloc pour l'interfaçage homme/machine. Dans les sections qui suivent nous donnerons une description sommaire sur les différents composants du processeur.

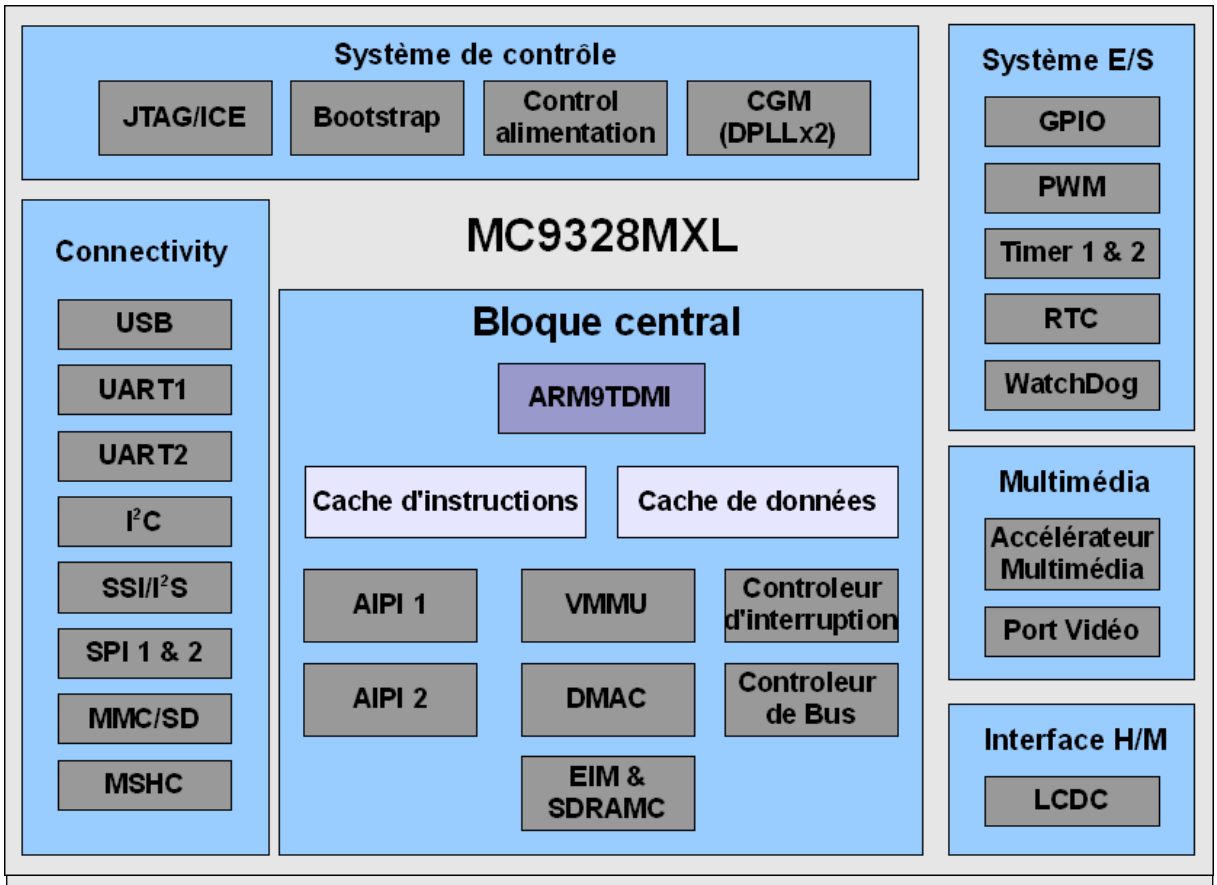

Figure 7-5 : *Représentation de l'architecture interne du processeur MC9328MXL.*

Le bloc central

Le bloque central se constitue des composants suivants :

ARM9 : le noyau du processeur i.MXL est à base d'ARM (Advanced RISC Machine) et comme son nom l'indique, c'est une architecture RISC (Reduced Instruction Set Computer) avec un nombre d'instructions réduit qui varie dans la majorité des cas entre trente et quarante instructions. Il faut noter que les processeurs ARM sont développés par le fabricant du même nom ARM, ce dernier développe une grande partie de ses architectures au tour d'un jeu d'instruction à 32 bit, quoique certains d'entre eux procèdent un deuxième mode de fonctionnement à 16 bit appelé *16-bit Thumb*. Ce mode a l'avantage de réduire la taille des programmes et a été essentiellement introduit par le fabricant dans le principal but de combler le manque de déploiement de ses processeurs dans les secteurs de marché pour applications embarquées à mémoire restreintes. L'ARM utilisé dans l'i.MXL est un modèle ARM9TDMI [101] ce caractérisant par une consommation électrique minime et un encombrement mécanique réduit. Avec le passage du processeur à la gamme d'ARM9, le fabricant à effectuer un grand changement dans la structure architecturale de ses processeurs, puisque c'est avec ces derniers qu'il abandonne l'architecture la plus usuelle de Von Neumann en migrant vers l'architecture de Hardvard. Dans cette dernière, et contrairement à l'architecture Von Neumann, les instructions et les données sont séparées en

deux bloques mémoires adressables distincts, ainsi le processeur dans un objectif de performance peut lire une instruction et écrire/lire une donnée en même temps. Ce genre de procédé est généralement il est applicable pour des processeurs avec une complexité réduite et avec un nombre d'étages pipelines limité, ce qui convient aux processeurs ARM9, puisque ces derniers implémentent une structure pipeline à seulement cinq étages. L'étage de recherche, de décodage, d'exécution, de mémoire, et d'écriture.

Cache d'instruction et Cache de données : ce sont des mémoires caches à accès rapide que le processeur utilise à des fins de performance. Les deux caches sont de 16 KB de capacité.

Contrôleur d'interruption : L'AITC (ARM9 InTerrupt Controller) est utilisé pour la gestion de 64 sources d'interruptions dans le processeur MC9328MXL. Il permet la distinction entre les interruptions lentes et les interruptions rapides, l'assignation de priorités aux interruptions, le masquage d'interruption et le support de 16 interruptions logicielles.

Contrôleur de bus : le processeur utilise un bus interne AHB (ARM Advanced High-performance Bus) qui suit les spécifications du protocole AMBAv2 (Advanced Microcontroller Bus Architecture version 2) développé par la société ARM. Les spécifications d'AMBA fournissent des normes pour la communication des bus intra-puce (bus internes à une puce électronique). Ces bus sont utilisés dans des composants comme les microprocesseurs, les microcontrôleurs ou les System-on-Chip.

AIPI : ou le AHB to IP Bus Interface. Ce module sert comme passerelle d'interfaçage entre le bus AHB et les bus des bloques IP caractérisés par des débits réduit. Les bloques IP (semiconductor Intellectual Property core) sont des unités fonctionnelles logique réutilisables développées par des parties tierces pour être intégrées dans des processeurs, des microcontrôleurs ou des SoC.

VMMU : ou le Virtual Memory Management Unit. C'est l'unité responsable de la gestion de la mémoire virtuelle. En d'autre terme, l'unité responsable de l'utilisation de la mémoire de masse comme mémoire centrale. L'APF9328 n'utilise pas mémoire virtuelle.

DMAC : ou le Direct Memory Access Controller est le contrôleur responsable de l'accès direct à la mémoire. Dans cette technologie, le transfert de données entre périphériques et mémoire centrale est effectué directement sans intervention du processeur. Le DMAC dispose de 11 canaux avec le support de mémoire linéaire, mémoire en 2 dimensions (organisée en lignes/colonnes), et mémoire FIFO (organisé en file).

EIM : ou External Interface Module est le module responsable de l'interfaçage du bus d'adresse/donné/contrôle avec les composants externes au processeur comme la mémoire et les périphériques. Il peut supporter jusqu'à 6 unités externes avec un adressage de 32 Mo pour chaqu'une. Il est aussi responsable de la sélection de la ROM d'où le programme d'amorçage doit être chargé et exécuté.

SDRAMC : le contrôleur SDRAM est responsable de la gestion de la mémoire centrale. Il supporte la norme PC100 SDRAM (Synchronous Dynamic RAM), cette dernière stipule que la mémoire fonctionne à une fréquence de 100 MHz avec un bus de 64 bit et une tension de 3.3 Volt.

Le système de contrôle

Le système de contrôle est constitué comme suite :

JTAG/ICE : (Joint Test Action Group / In-Circuit Emulator) est une unité qui permet le test et le débogage d'un processeur. En d'autres mots, ça permet d'avoir un accès direct à l'intérieur de ce dernier (points d'arrêt, lecture et écriture des registres internes, des mémoires internes et externes …etc) sans perturber ses interactions avec les composants externes.

Bootstrap : Dans un processus de démarrage usuel, l'EIM ce charge de transférer et d'exécuter le programme d'amorçage directement de la ROM. Tandis que le bootstrap permet de contourner la procédure usuelle en téléchargeant les programmes du port UART du processeur. Cette procédure reste comme seule alternative pour booter son système en cas de défaillance du programme d'amorçage ou de la mémoire ROM.

Contrôle d'alimentation : Ce module est responsable de l'alimentation du processeur, il a la faculté de faire basculer ce dernier en trois états distincts, arrêt, marche, mis-en-veille. Pour des raisons d'économies d'énergie, il peut aussi désactiver des blocks non utilisés dans ce dernier.

Contrôle d'horloge : Ce module est responsable de la gestion de la multitude d'horloges que le processeur peut avoir besoin. Le générateur d'impulsions principales est un quartz cadencé à 32.768 KHz, il est utilisé pour générer n'importe quelle fréquence d'horloge que le processeur est susceptible d'avoir besoin.

Le système d'entrée/sortie standard

Le système d'entrée/sortie standard est constitué des composants suivants :

GPIO : ou General Purpose Input Output. Ce sont 97 pins de 3.3 Volt chaqu'un, utilisés pour un usage général. Ils sont fournis par le i.MXL à l'utilisateur pour lui permettre de programmer ses propres interfaces d'entrée/sortie.

PWM : (Pulse Width Modulation) est une technique couramment utilisée pour synthétiser des signaux continus à l'aide de circuits fonctionnant avec des signaux discrets. Le principe de fonctionnement du PWM est d'appliquer une succession de signaux discrets pendant des

durées courtes et bien choisies, la moyenne de ces derniers fournit un signal de n'importe quelle valeur intermédiaire continue. Le MC9328MXL dispose d'un PWM avec une résolution maximum de 16 bit, et un buffer FIFO de 4 bit x 16. L'une des applications les plus rependue du PWM est de l'utiliser comme générateur audio pour les applications multimédias.

Timer : Un Timer est un périphérique matériel permettant de mesurer le temps. Il se présente généralement sous forme d'un compteur d'entiers, initialisé à une certaine valeur, il se décrémente périodiquement pour générer une interruption lorsque il atteint la valeur zéro. L'i.MXL possède 2 Timer de 32 bit avec une granularité minimale de 10 nanosecondes.

RTC : (Real-Time Clock) est un circuit intégré d'une horloge numérique, son rôle est de garder une trace de l'évolution du système dans le temps. L'i.MXL fourni une horloge avec un quartz de 32 KHz ou de 32.768 KHz, avec la prise en charge des secondes, minutes, heurs, et jours, avec un maximum de 512 jours.

WatchDog : ou le chien de garde, est un circuit intégré utilisé pour s'assurer que le système ne reste pas bloqué à une étape particulière dans le traitement qu'il effectue. C'est une protection destinée généralement à redémarrer le système dans le cas où une action définie n'est pas exécutée dans un délai imparti. L'i.MXL fourni un WatchDog avec une granularité de 0.5 seconde, l'intervalle permit pour une période de test est entre 0.5 seconde jusqu'à 64 secondes.

Le système Multimédia

Le système multimédia est constitué comme suite :

Accélérateur Multimédia : l'accélérateur multimédia est utilisé conjointement avec l'i.MXL pour effectuer des opérations répétitives sur des applications multimédias, comme le décodage MPEG, décodage/encodage MP3, et la compression/décompression de la parole.

Port Vidéo : le rôle du port vidéo est de faire entrer de la vidéo capturé d'un dispositif CSI (CMOS sensor interface) au système embarqué.

L'interface homme/machine

L'interface homme/machine est constituée de :

LCDC : le Liquid Crystal Display Controller est un module permettant l'affichage vidéo sur des surfaces LCD. Le LCDC est capable de supporter des LCD noire et blanc, des LCD à niveau de gris, des LCD passive-matrix color, et des LCD active-matrix color.

La connectivité

La connectivité de l'i.MXL est fort riche, elle se constitue de :

USB : l'USB présent sur le processeur est un USB révision 1.1 slave, ainsi il ne peut supporter qu'une fréquence maximale de 12 MHz. Il est possible d'avoir l'USB en master avec l'ajout d'un contrôleur d'USB master dédié, c'est le cas dans la carte APF9328DevFull.

UART : ou Universal Asynchronous Receiver Transmitter est un circuit électronique qui a pour rôle de sérialiser les transmissions parallèle. En d'autres mots, c'est un circuit qui prend les bits sur plusieurs fils d'une liaison parallèle et les transmis les uns après les autres sur un seul fil de liaison série. Le i.MXL offre 2 UART avec une fréquence programmable jusqu'à 1 MHz, avec une transmission série de 7 ou 8 bits, 1 ou 2 bits de stop, et 1 bit de parité programmable (pair, impair ou non utilisé).

I^2C : ou Inter Integrated Circuit est un bus série bifilaire bidirectionnel multi-master/multi-slave. L'I^2C est convenable pour les applications exigeant des communications occasionnelles et à faible débit dans un espace réduit. Il est dans la plupart du temps utilisé pour la communication des composants dans les SBC et les cartes mères industrielles.

SSI/I^2S : (Serial Synchronous Interface/Integrated Inter-chip Sound) est un bus série full-dupex qui permet au i.MXL de communiqué avec une multitude de périphériques comme les microprocesseurs, les DSP, les périphériques implémentant le port SPI, et les codecs audio implémentant l'I^2S.

SPI : ou Serial Peripheral Interface est un bus série synchrone mono-master/multi-slave qui opère en full-dupex. Le i.MXL comporte 2 SPI, l'SPI_1 est configurable pour être master /slave, tandis que SPI_2 ne peut être que master. Le transfert de données peut être programmé pour monter jusqu'à 16 bits par mots. Les SPI sur i.MXL utilisent un buffer FIFO de 8 bit x 16 pour les deux directions transmission/réception.

MMC/SD Host Controller : (MultiMedia Card / Secure Digital Card). L'MMC est un dispositif de stockage de masse sous forme de carte mémoire à base de mémoire flash. Son principal avantage est l'aisance de l'utilisation de son interface de communication, d'ailleurs, mis à part le stockage de donnés cette interface peut aussi être utilisé comme un bus de communication. La SD est une évolution de la carte MMC qui vise à accroitre la sécurité, la capacité, et les performances de ce modèle de carte. Le contrôleur MMC/SD implanté avec le processeur i.MXL peut gérer jusqu'à 10 interfaces MMC et une interface SD, et supporte un transfert de données allant de 20 Mbps à 80 Mbps.

MSHC : (Memory Stick Host Controller) est aussi un contrôleur pour carte mémoire (les cartes MS), il offre un débit en lecture/écriture aux alentours de 20 Mbps. Il intègre un buffer FIFO de 8 octet x 4 et un support de la DMA.

7.4.2. L'Ethernet

L'APF9328 dispose d'un contrôleur Ethernet du modèle Davicom DM9000 [102]. Le module offre une interface 10/100 Base TX (liaison Ethernet de 10 ou 100 Mbit/s avec l'utilisation de la totalité de la bande passante fréquentielle sur un câble de pair torsadé non blindé), munit d'un gestionnaire de couche physique (PHY) chargée de la conversion entre bits et signaux électriques, et le gestionnaire MAC (Media Access Control) responsable de la résolution de conflits dans le domaine de collusion. Le contrôleur est directement relié au processeur par les bus de données / adresses / contrôle, tandis que sa sortie se réduit aux connecteurs pour l'RJ45 et une LED qui fournit des informations sur l'état de la connexion.

7.4.3. L'FPGA

Les circuits Field-Programmable Gate Array sont constitués d'une matrice de portes logiques reliée par un réseau d'interconnexion programmable. L'APF9328 embarque un FPGA Xilinx Spartan3 [103] avec les caractéristiques suivantes :

- 200 portes logiques organisées en 24 lignes et 20 colonnes.
- Une RAM de 216 Ko
- Une connexion directe avec le processeur à 16 bit et 25 MHz.
- 59 connecteurs d'entré/sortie utilisateur.
- L'alimentation de l'FPGA est fournie par deux régulateurs, le principal à +1.2 Volt et le secondaire à +2.5 Volt.

7.4.4. L'ADC

L'*Analog to Digital Converter* ou *convertisseur analogique numérique* est un dispositif qui convertit un signale numérique en un signale analogique. L'APF9328 utilise un ADC Maxim MAX1027 [104] avec une résolution de 10 bits et un buffer FIFO de 16 entrées. Comme illustré sur la *figure*7-4, le processeur utilise le port SPI_1 pour contrôler l'ADC.

7.4.5. Le DAC

Le *Digital to Analog Converter* ou *convertisseur numérique analogique* est le contraste du ADC, il convertit le signal analogique en signal numérique. L'APF9328 utilise un DAC Maxim MAX5821 [105]. Le MAX5821 est un circuit duel DAC avec une résolution de 10 bits, et communicant avec le processeur en utilisant le bus I^2C avec une fréquence de 400 KHz.

7.5. L'APF9328DevFull

La carte APF9328DevFull est une carte de développent spécifique à l'SBC AFP9328, elle suit la norme APIC et présente un encombrement mécanique de dimensions 115mm x 165mm (4,528 pouces x 6,496 pouces). L'APF9328DevFull est conçu de telle sorte que la majorité des connectiques que le processeur i.MXL et l'APF9328 possèdent soit disponible aux développeurs, leurs dispositions sont montées sur la *figure*7-6, ce sont : les connecteurs USB 1.1, JTAG, RS232, I^2C, SSI, UART, LCD, MMC/SD, RJ45, DAC, ADC. En plus des connectiques du processeur et de l'APF9328, l'APF9328DevFull possède ses propres contrôleurs qui sont dans la majorité des cas standards pour un processus de développement visant les applications multimédias embarquées, ces contrôleurs sont : le contrôleur host USB 2.0, le contrôleur de Sortie Audio Stéréo, le contrôleur de TouchScreen et le contrôleur de Sortie Vidéo. Ces derniers sont aussi illustrés avec leurs connectiques sur la *figure*7-6. En raison d'absence de quelques composants fournis en options que notre APF9328DevFull ne dispose pas, nous ferons que mentionner leurs existences : une SRAM pour l'FPGA, un contrôleur RTC secondaire et un contrôleur de bus CAN.

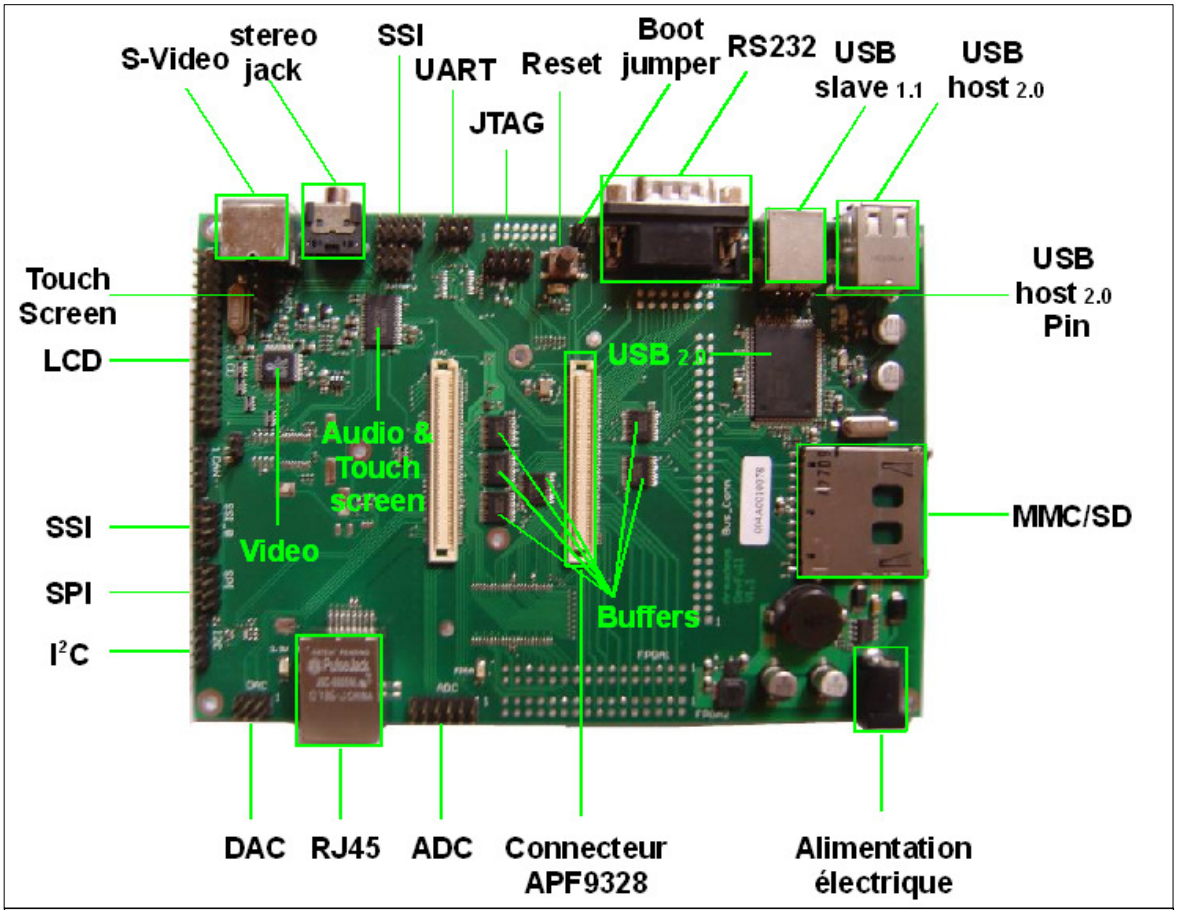

Figure 7-6 : *Vue de face la carte APF9328DevFull.*

7.5.1. Le Contrôleur Audio Stéréo & Touch Screen

Le Texas Instruments TSC2102 [106] est un contrôleur pour trois unités fonctionnelles distinctes. L'unité Touch Screen (écran tactile) a pour rôle la détection et la localisation tactile sur

une zone d'affichage, elle est compatible Touch Screen 4-wire 12 bit (une des technologies Touch Screen très répondue). Cette unité possède une interface SPI qui lui permet de communiquer avec le processeur, le processeur i.MXL utilise son bus PSI_2 pour la contrôler. La deuxième unité est un processeur intégré, son principale rôle est minimiser la charge de calcule sur le processeur centrale et d'enlever le surplus de communication sur le bus SPI_2. Le processeur intégré opère principalement sur les opérations de traitement de son, comme la génération de timing, la conversion de résolution allant jusqu'à 12 bits et l'échantillonnage avec un taux allant jusqu'à 125 KHz. Le TSC2102 intègre aussi une unité de conversion analogique/numérique audio avec un échantillonnage de 16, 20, 24, 32 bits stéréo (deux canaux audio) avec un débit de lecture maximum de 48 Kbit/s. L'unité audio du TSC2102 peut être programmée pour le support d'un headphone (casque à microphone) ou des signaux level-line (signaux générés par les lecteurs CD, lecteur DVD, lecteur MP3…etc).

7.5.2. Le Contrôleur Vidéo

Le Chrontel CH7024 [107] est un encodeur TV visant les appareils à taille réduite, comme les caméras numériques, les téléphones portables multimédias, les smartphones…etc. Le contrôleur est capable d'encoder le signal vidéo numérique et générer un signal de synchronisation pour les standards NTSC (National Television System Committee) et PAL (Phase Alternating Line) avec une résolution maximale de 720x480. Les ports de sortie supportés sont le CVBS (Color Video Blanking Synchronisation : appelé souvent port *composite vidéo*) et le S-Vidéo (Super Video : c'est ce dernier qui est implémenté sur l'APF9328DevFull). Le CH7024 arrive a communiquer avec le processeur par l'intermédiaire de deux bus, le bus I^2C pour le contrôle et le bus LCD pour le transfert de donné (flux vidéo). Le CH7024 a aussi l'avantage de supporter les deux formats de codage vidéo les plus populaires, le RGB (Red Green Blue) et YCrCb (Luminance, Chrominance red, Chrominance blue).

7.5.3. Le Contrôleur USB

Le NXP ISP1760 [108] est un contrôleur USB host version 2. Il intègre un contrôleur EHCI (Enhanced Host Controller Interface). L'EHCI est un ensemble de normes pour contrôleurs USB, leurs spécifications décrivent l'interface hardware/software entre le soft et le contrôleur. L'ISP1760 dispose de trois émetteurs-récepteurs, sur la carte APF9328DevFull le contrôleur affecte deux émetteurs-récepteurs à deux ports USB standard et le troisième à un port pin (*figure*7-6). Le processeur i.MXL gère le contrôle et le transfert de données avec l'ISP1760 en utilisant son bus d'adresse / donné / contrôle. En sachant que le contrôleur gère la DMA, Le transfert de données peut aussi être effectué directement avec cette dernière.

7.6. Conclusion

Ce chapitre a été consacré à l'SBC APF9328 et sa carte hôte l'APF9328DevFull, il se focalise principalement sur les trois points suivants, la structure architecturale, la connectique et le comportement fonctionnel des différents composants appartenant à ces deux cartes. L'APF9328 comporte la partie la plus riche en composants, tendit que l'APF9328DevFull est perçu beaucoup plus comme un support connectique qu'une unité fonctionnelle appart entière. De ce fait, la majeure partie de ce chapitre était consacré à la description de l'APF9328, dont une grande partie était

consacré à son processeur i.MXL, pour la simple raison qu'il rassemble en lui tout seul plus de fonctionnalités que les deux cartes réunies. Chaque unité de ce dernier était brièvement décrite, ainsi que les composants les plus importants de l'APF9328 et de l'APF9328DevFull. Le chapitre suivant sera consacré à l'étude de l'installation du support logiciel primaire pour la gestion de l'SBC APF9328, en d'autres mots, l'installation du programme d'amorçage et du système d'exploitation.

Intégration du logiciel de gestion 8

8.1. Introduction

Pour qu'un système embarqué soit capable d'accomplir ses fonctionnalités, le logiciel de gestion doit être correctement installé. Le logiciel de gestion s'appuie généralement sur le système d'exploitation et le programme d'amorçage, qui ont pour rôle de soutenir le matériel pour qu'il puisse être utilisé convenablement par l'applicatif. Le rôle du logiciel de gestion ne se résume pas en une plate-forme de communication avec le hardware pour le software, il a plusieurs autres vertus comme par exemple l'initialisation et la vérification du matériel au démarrage, la gestion des processus, la communication inter-processus, la gestion du système de fichier...etc. Toutefois pour les systèmes embarqués et contrairement aux PCs ordinaires, l'installation de ce dernier n'est pas une chose aisée. La principale raison en comparaison avec les PCs, c'est que les systèmes embarqués ont un matériel spécifique souvent faible en ressources mémoire et de calcul, la plus part du temps il faut tailler le système d'exploitation et le programme d'amorçage d'une façon minutieuse pour qu'il colle parfaitement avec le hardware du système embarqué.

Dans ce chapitre nous allons suivre la procédure d'installation sur l'SBC APF9328 du logiciel de gestion, qui se compose principalement du programme d'amorçage U-Boot et du système d'exploitation Linux. Nous allons voir qu'une grande partie de ce processus se fait à l'aide de l'environnement de développent Armadeus SDK. Ce processus sera plus ou moins détaillé dans les trois importantes étapes qui sont, l'étape de développement, l'étape du transfert et l'étape de l'installation. À la fin de ce chapitre nous aurons un aperçu sur la technique utilisée par Armadeus pour se procurer du hard real time sur sa gamme de carte SBC.

8.2. Concepts et terminologies

Le développement des systèmes embarqués, obéis à des concepts et une terminologie totalement différente de celle utilisée dans le développement usuel. Dans ce qui suit, nous allons faire le tour sur les principaux termes/concepts dans le domaine.

Compilation croisée : La compilation croisée tend à produire des exécutables pour une architecture processeur différente de celle où la compilation a été effectuée. L'importance de la compilation croisée réside dans le fait de créer un exécutable pour un système qui ne dispose pas de compilateur, et c'est souvent le cas pour les systèmes embarqués, car contrairement aux systèmes usuels, ces derniers ne disposent pas des ressources nécessaires pour effectuer la compilation.

Plateforme hôte et plateforme cible : une plateforme dans le contexte des systèmes embarqués désigne une combinaison de deux entités, système d'exploitation et architecture matérielle. Dans les systèmes embarqués, on parle souvent de deux plateformes distinctes, la plateforme hôte et la plateforme cible. La première est celle où le développement croisé prend place, la deuxième est celle où l'application sera implantée (la cible). Dans notre projet, la plateforme hôte est un x86/Linux et la plateforme cible est un ARM/Linux.

Chaines d'outils (Toolchains) : une chaine d'outils est un ensemble d'outils utilisés pour la construction d'une application ou d'une image mémoire du soft qui sera implanté sur l'SBC. L'appellation vient du fait que les outils sont généralement utilisés en chaines, la sortie de l'un fait l'entrée de l'autre, quoique le terme puisse aussi être utilisé pour désigner tout ensemble d'outils de développement.

Outils de construction automatique : il n'est pas rare de trouver des outils d'automatisation dans les enivrements de développement embarqués. La raison vient principalement de la nature difficile du développement embarqué, elle est souvent due au nombre important d'outils utilisés et de la complexité conséquente de la configuration dans les Toolchaines. Les outils de construction automatique tentent de rendre le processus de construction logicielle plus facile et plus accessible aux développeurs pour systèmes embraqués.

8.3. Développement embarqué

Développer et mettre au point une application pour un système embarqué est un processus un peu particulier. Il se base principalement sur le concept du développement croisé, avec une machine hôte qui maintient le processus de développement, et une machine cible, qui va à la fin du processus recevoir l'exécutable sous forme d'image mémoire qui va être au final implanté sur la ROM de cette dernière. L'image mémoire est constituée en

général du programme boot, du système d'exploitation, des applications, outils…etc. Sur la *figure*8-1 on peut apercevoir le processus de développement propre à notre travail, la machine hôte est un PC standard fonctionnant sous un système d'exploitation Linux Ubuntu, intégrant l'environnement de développement *Armadeus 3.2* et communicant avec la machine cible (l'APF9328) en utilisant une connexion série (RS232) et/ou une connexion réseau Ethernet. Le câble série étant de faible débit, son utilisation ne se résume en général qu'à la commande de l'SBC, tandis que la connexion Ethernet pour son haut débit sera dans la plupart du temps utilisée pour le transfert de données.

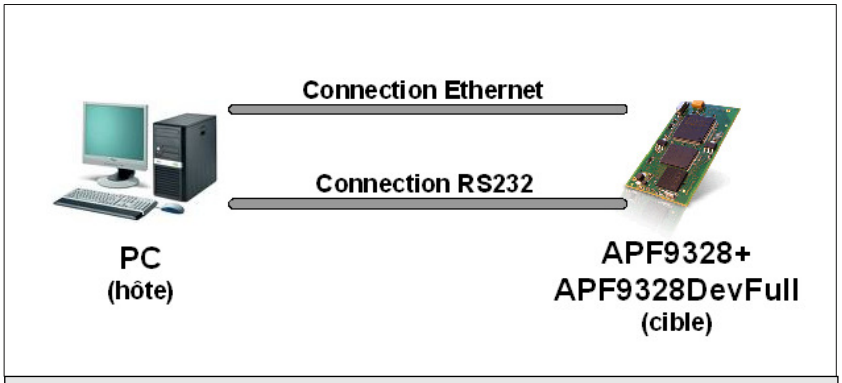

Figure 8-1 : *Développement croisé sur l'SBC APF9328.*

8.4. Enivrements de développement Armadeus

L'environnement de développement propre à Armadeus s'intitule *Armadeus 3.2 SDK* (Software Developpement Kit). C'est un environnement de développement intégré qui a pour rôle de construire une image mémoire système destinée aux systèmes embarqués produit par Armadeus System. En d'autres termes, l'Armadeus SDK permet aux développeurs de construire des images mémoires modelées selon le besoin précis du projet, en offrant le choix entre les fonctionnalités du système d'exploitation, les modules de périphérique, les packages, les outils, les applications et tout autres softs que le système est susceptible d'en avoir besoin. La *figure*8-2 présente le schéma structurel de l'Armadeus SDK, dans lequel en peut remarquer que l'environnement se constitue d'un ensemble d'outils formant une Toolchain dans laquelle l'élément le plus important est le *Buildroot*. Buildroot est un utilitaire qui détient tous les paramètres configurables pour la construction d'un projet donné, son principe de fonctionnement consiste a réunir tout les sources du projet qui sont fournis a priori par l'environnement de développement, comme ceux du noyau linux, du programme d'amorçage *U-Boot*, du *frimware* de l'FPGA et du gestionnaire de fichier *fsroot*. Ou les sources fournies par l'utilisateur, comme ceux des packages et des applications, et selon la configuration ajustée par le développeur, produit un fichier *makefile* qui sera utilisé par l'utilitaire *make*. make est l'utilitaire de la chaine d'outils responsable de la construction automatique du projet, il fait en sorte d'appeler le compilateur GCC et de compiler tous les sources mentionnés par Builroot. Il exécute ensuite *binutils* associé aux bibliothèques *uClibc* pour au final créer des exécutables en fichiers binaires mappable en mémoire. Il faut noter que tous les utilitaires utilisés dans Armadeus SDK sont des applications open source sous la licence GPL, ces utilitaires seront détaillés dans les sections qui suivent.

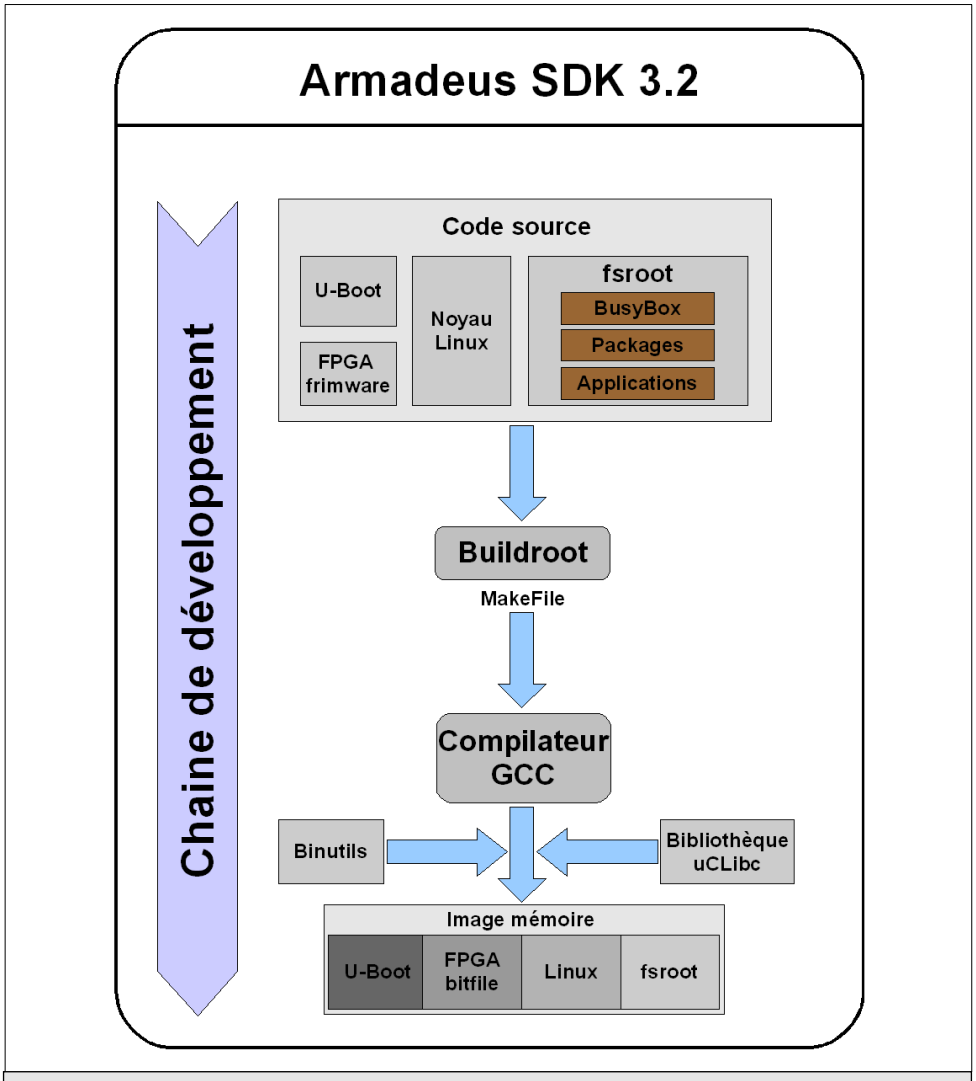

Figure 8-2 : *Schéma structurel de l'environnement Armadeus SDK.*

make : est un utilitaire Linux utilisé pour la construction automatique d'exécutables, de programmes et de bibliothèques a partir de leurs codes sources, il se sert des commandes trouvées dans le fichier makefile pour créer les dépendances et générer la cible que ça soit programme, bibliothèque, ou même de la documentation.

makefile : est le fichier qu'utilise make pour produire sa cible. C'est un fichier texte qui contient le plus souvent des commandes, des directives, des dépendances, des macros, des appels récursifs a d'autres fichiers makefile et même des scriptes shell.

Buildroot : est un utilitaire utilisé par les deux parties, développeurs et producteurs/constructeur. Il permet aux producteurs de cartes pour systèmes embarqués de réaliser des Toolchains de compilation croisé facilitant la tâche de développement aux clients de leurs produits. Tandis que les développeurs considèrent Buildtool comme un gestionnaire de fichiers makefile pour une base de codes sources

fournie par le producteur, ce gestionnaire leurs permet avec un menu relativement aisé de produire et de modeler des cibles adéquates aux besoins de leurs projets.

GCC (Gnu Compiler Collection) : c'est un compilateur fournit par la fondation GNU project. Il s'agit d'une collection intégrée de compilateurs capables de compiler divers langages de programmation, dont le C, C++, Objective-C, Java, Ada et Fortran. Il supporte aussi un nombre conséquent de processeurs dont x86, ARM, DEC Alpha, M68k, MIPS, PowerPC, SPARC, Hitachi H8.

binutils (Binary Utilities) : est un ensemble d'outils de développement logiciel fournit aussi par la fondation GNU project, Ils sont surtout considérés comme des outils spécialisés dans la manipulation des fichiers code objets produits par le compilateur. Leurs deux utilitaires les plus importants sont **ld** l'éditeur de liens et **as** l'assembleur.

uClibc : c'est une bibliothèque standard C destinée au développement de systèmes embarqués tournant sous Linux. uClibc est l'équivalente dans son interface à la bibliothèque Glibc utilisée sous Linux, quoiqu'elle diffère de cette dernière dans la taille mémoire, puisque uClibc est largement inferieur a Glibc. Et la deuxième différence est celle du support des processeurs sans MMU, cette dernière caractéristique découle du fait que uClibc résulte du projet uCLinux visent a crée un noyau linux supportant les architectures avec processeur sans MMU.

U-Boot (Universel Boot) : est un Bootloader (programme d'amorçage) qui a comme principal rôle de charger le système d'exploitation de la ROM vers la RAM et de l'exécuter. L'avantage majeur de l'utilisation de U-Boot est sa nature open source, ce qui le rend extrêmement flexible, les producteur de cartes embarqués n'ont qu'a le façonner pour le rendre portable sur leurs cartes SBC. U-Boot ce caractérise aussi d'un large éventail de commandes et d'utilitaires additionnels qui facilite considérablement la vie aux développeurs, dont les deux les plus utilisés, qui sont le client TFTP et le client NFS pour le transfert aisé des programmes/données entre le hôte et sa cible.

fsroot (File System Root) : ou système de fichiers racine. C'est le système de fichier principal qui contient le BusyBox les Packages et les applications qui seront montées par Linux. Dans les systèmes embarqués, deux systèmes de fichiers sont communément utilisés, le CRAMFS (Compressed ROM File System) qui est un système de fichier compressable utilisé en lecture seule destiné principalement aux EPROM. Et le JFFS2 (Journalling Flash File System version 2) destiné aux mémoires flash NOR et NAND et c'est ce dernier qui sera utilisé sur la mémoire Flash de notre SBC.

BusyBox : est un utilitaire qui regroupe un grand nombre de commandes standards Linux. Il est compacté sous un unique fichier exécutable de taille relativement réduite, avec sur son compte plus de deux cents programmes en un seul fichier exécutable.

8.5. Les étapes d'intégration logicielle

Le processus d'intégration logiciel se déroule selon un procédé constitué de trois étapes. La première étape consiste à construire sur la plateforme hôte l'image mémoire du système logiciel. La deuxième étape se focalise sur la préparation de l'SBC à recevoir l'image mémoire. La troisième étape traite le transfert et l'installation de l'image mémoire sur l'SBC. Il faut noter que le processus d'intégration décrit ci-après n'est pas générique, il est plutôt spécifique aux besoins propre à notre projet, ceci dit une configuration pour la construction d'un projet Set-Top-Box avec une sortie vidéo.

8.5.1. Construction logicielle

Dans cette étape nous allons suivre les commandes permettant la construction des fichiers images mémoire nécessaires à la construction du logiciel de notre système. À la fin de cette étape nous aurons à notre disposition plusieurs fichiers dont trois d'entre eux représentent les fichiers images mémoires. Ces fichiers sont le fichier image du boot, le fichier image du système d'exploitation et le fichier image du système de fichier.

En premier lieu, il faut se procurer le code source d'Armadeus SDK, pour y arriver on utilise la commande `git` (`git` est un logiciel distribué de gestion de code source).

```
$ git clone
git://armadeus.git.sourceforge.net/gitroot/armadeus/armadeus
armadeus
```

Cette commande va créer un répertoire nommé `armadeus-3.2`, dans le quelle on utilise la commande `make` pour construire à partir des sources l'environnement Armadeus SDK.

```
$ cd armadeus-3.2
$ make apf9328_defconfig
```

Après la construction c'est le menu de Buildroot qui va être affiché (*figure*8-3), et c'est ce dernier qui va nous permettre de sélectionner les configurations adéquates pour notre projet.

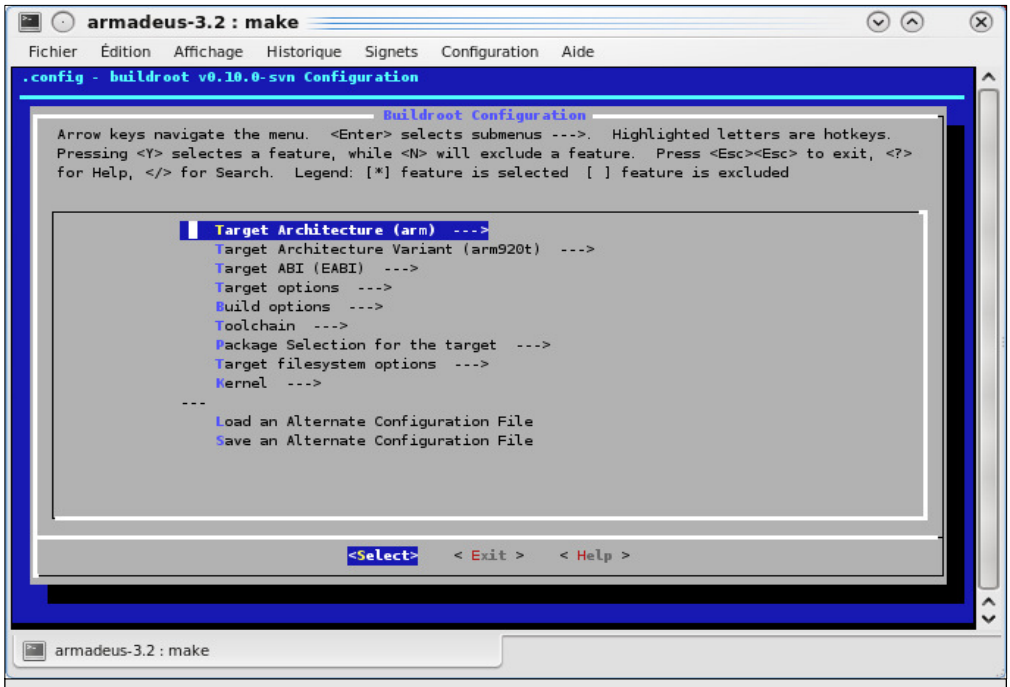

Figure 8-3 : *Menu de configuration principale de Buildroot.*

Plusieurs menus de configuration sont à la disposition du développeur, la plus importante d'entre elles est sans doute celle du choix du processeur, elle est effectuée selon l'arborescence :

```
Target Architecture (arm)  --->  (X) arm
```

Ensuite vient le choix du modèle du processeur, dans notre cas c'est l'ARM920T :

```
Target Architecture Variant (arm920t)  --->  (X) arm920t
```

Apres vient une série de menus imbriqués indiquant l'SBC utilisé, dans notre cas c'est APF9328 de 16 Mo de RAM.

```
Target options  --->  [*] Armadeus Device Support  --->  Armadeus
target device (apf9328)  --->  (X) apf9328
Target options  --->  [*] Armadeus Device Support  --->  Target
SDRAM size (16MB)  --->  (X) 16MB
```

Les deux menus en dessous sont ceux de `Build options` et `Toolchain`, ce sont des menus de configuration qui concerne le compilateur et la chaine d'outils. Nous les laisserons avec leurs configurations par défaut.

Le menu suivant est `Package Selection for the target`, il est responsable de la sélection et l'installation des différents package pour le système cible. Pour notre projet, les deux packages qu'on aura besoin sont celui de la gestion du contrôleur vidéo ch7024 et celui du lecteur vidéo `lecteur mplayer`. L'arborescence des deux menus est comme suit :

```
Package Selection for the target  --->  [*] Hardware handling /
blockdevices and filesystem maintenance  --->  [*]  ch7024ctrl
Package Selection for the target  --->  [*] Graphic libraries and
applications (graphic/text)  --->  [*]  mplayer
```

Le menu d'après est celui de la configuration du système de fichier `Target filesystem options`. L'option que notre système a besoin est celle du choix du système de fichier, notre projet utilise le JFFS2.

```
Target filesystem options  --->  [*] jffs2 root filesystem
```

Le menu suivant est celui de la configuration du noyau Linux. L'option qui nous concerne est celle de la version du noyau utilisé. Selon Armadeus system l'APF9328 prend totalement en charge la version 2.6.29.

```
Kernel -->  Linux Kernel Version (Linux 2.6.29)  --->  (X) Linux
2.6.29
```

Il faut noter qu'il est possible d'avoir un menu de configuration spécifique au noyau linux, quoique il est impératif d'utilisé sa commande `make` avant d'utiliser la commande `make` du menu `Buildroot`. La commande est comme suite :

```
$ make linux26-menuconfig
```

La phase finale qui vient après avoir sauvegardé et quitté le menu `Buildroot`, est celle de la compilation, elle utilise encore une fois l'outil de construction automatique `make`.

```
$ make
```

Après la compilation, ce sont cinq fichiers en sortie représentant l'image mémoire globale du système, ils sont disposés dans le répertoire `buildroot/binaries/apf9328/` et on peut les visualiser sur le listing si dessous en utilisant la commende `ls`.

```
$ ls -l

1687632  2009-10-02 13:04 apf9328-linux.bin
5898240  2009-10-02 13:20 apf9328-rootfs.arm.jffs2
10567680 2009-10-02 13:20 apf9328-rootfs.arm.tar
173220   2009-10-02 12:56 apf9328-u-boot.bin
466505   2009-10-02 13:20 apf9328-u-boot.brec
```

Sur le listing, la première colonne donne la liste des tailles mémoires des différents fichiers. Tandis que la dernière colonne donne la liste des fichiers présents dans le répertoire. Ils sont décrits comme suite :

`apf9328-u-boot.bin` : L'image mémoire du Bootloader U-Boot.

`apf9328-u-boot.brec` : L'image mémoire utilisée en cas de disfonctionnement de U-Boot.

`apf9328-linux.bin` : L'image mémoire du noyau Linux.

`apf9328-rootfs.arm.jffs2` : L'image mémoire du system de fichier fsroot.

`apf9328-rootfs.arm.tar` : system de fichier pour l'NFS (Network File System : système de fichier sur réseau) ou MMC/SD.

Dans la construction de l'image mémoire du système, la taille des différents fichiers images est un élément qu'il faut prendre en considération. Car le mappage mémoire exige que chaque image doive prendre avec précision la place qui lui est allouée, étant donné qu'il est totalement probable de construire des images supérieures à leurs tailles d'allocation, ce qui va faire chevaucher les images les unes sur les autres et détruire l'image globale du système. Le *Tableau*4 donne la distribution de ces images sur la mémoire flash de l'APF9328 à 16 Mo de mémoire flash.

Image mémoire	Plage mémoire (taille)
U-Boot	0x000000 - 0x03FFFF (256KB)
Variable U-Boot	0x040000 - 0x5FFFF (128KB)
FPGA bitfile	0x060000 - 0x9FFFF (256KB)
Noyau Linux	0x0A0000 - 0x29FFFF (2MB)
fsroot	0x2A0000 - 0xFFFFFF (~13.5MB)

Tableau 4 : *La distribution des images mémoires sur la mémoire flash du APF9328.*

8.5.2. La préparation pour le transfert

La seconde étape est celle de la préparation du transfert des images mémoires du PC hôte vers la cible APF9328. Cette étape débute avec la préparation du câble série (ou RS232) et du câble Ethernet. Le câble série est de type Null Modem, son schéma de liaison est illustré sur la *figure*8-4, tandis que le câble Ethernet est un câble réseau croisé à connecteurs RJ45. Le câble réseau est surtout utilisé pour le transfert des fichiers lors de l'installation du software, alors que le câble série est généralement utilisé pour la commande de la carte. La principale raison pour ce fait, réside dans l'inégalité de la vitesse de transfert entre les deux câbles ; le câble réseau étant largement plus rapide que le câble RS232. Il faut aussi noter que le câble série est le seul moyen d'accéder à la carte en cas de disfonctionnement du Bootloader ou lors du premier contact avec une carte à mémoire vierge (le cas d'une carte toute neuve). Cette situation a été signalée dans le chapitre précédent, dans laquelle pour communiquer avec l'APF9328 de mémoire vierge, il faut faire entrer son processeur en mode bootstrap, et laisser ce dernier utiliser son port UART/RS232 pour l'acquisition des premières instructions de démarrage.

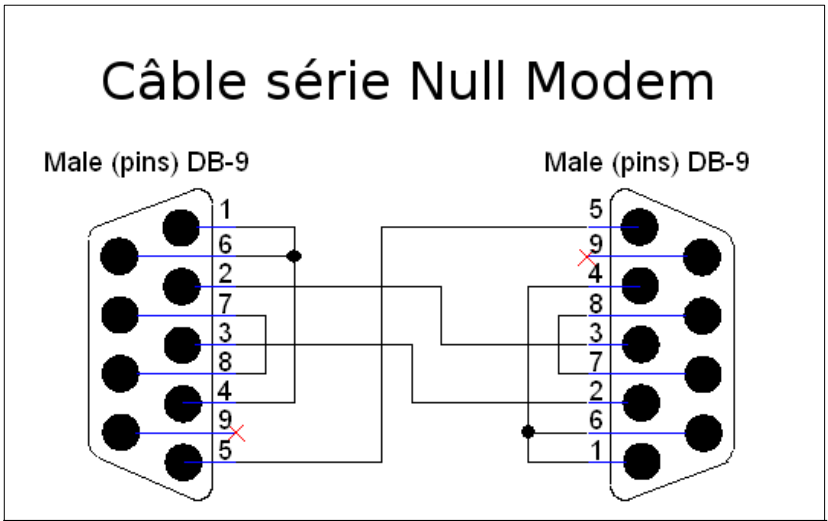

Figure 8-4 : *Schéma d'un câble série Null Modem.*

La procédure pour faire entrer le processeur i.MXL en mode Bootstrap exige la fermeture du Jumper Bootsrap sur la carte APF9328DevFull comme illustré sur la *figure*8-5, connecter la cible avec le hôte en utilisant le câble série, puis exécuter les commandes du listing si dessous. La première commande dans le listing installe l'utilitaire **`python-serial`** qui donne aux scripts pythons installés sur le hôte la capacité d'utiliser le port série de ce dernier. La deuxième commande interprète le script python **`apf9328_recover`** fournie avec l'Armadeus SDK. Ce dernier utilise le fichier de boot `apf9328-u-boot.brec` pour booter la carte. Apres le démarrage le développeur peut user des outils fournis avec le boot pour faire transférer les fichiers images du hôte vers la cible.

```
$ sudo apt-get install python-serial
$ python apf9328_recover.py
```

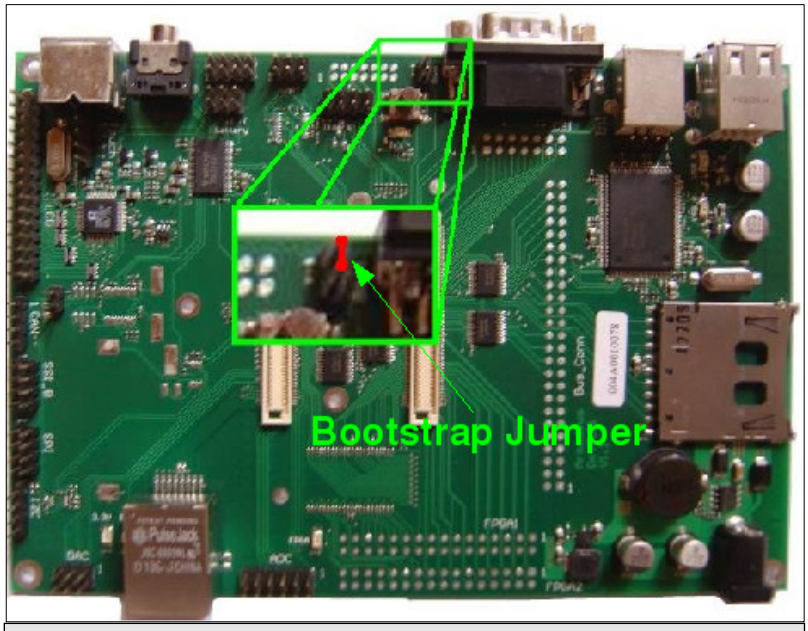

Figure 8-5 : *Jumper du Bootstrap sur l'APF9328DevFull.*

La communication entre la cible et le hôte en utilisant la liaison série suit un mode un peu particulier, que ca soit avec le boot de `apf9328_recover.py`, U-Boot ou Linux, elle suit plus exactement le mode de communication utilisé avec les serveurs Unix, dans le quelle des terminaux passifs (écran plus clavier sans aucune unité de traitement) envoient les commandes à l'unité centrale, qui les traite et renvoi en retour les résultats désirés. La communication en série avec les SBC d'Armadeus se fait de la même manière, le PC hôte fait office de terminal et la cible fait office de serveur Unix. C'est pour cette raison que des utilitaires doivent être installés sur le PC hôte lui permettant l'émulation d'un terminal passif. L'utilitaire que nous avons utilisé pour venir à bout de cette tâche est le logiciel GTKTerm, il transforme un ordinateur personnel en un terminal avec comme simple composant fonctionnel le clavier et l'écran.

8.5.3. Transfert et installation

Après l'étape de la construction des images mémoires et l'établissement du premier démarrage de la carte, il ne reste que l'étape du transfert et de l'installation des fichiers images sur la mémoire flash de la cible. La première opération a faire est de régler la configuration réseau de l'SBC. Cette opération est effectuée avec la commande `setenv` qui permet l'affectation de paramètres aux variables d'environnement du Bootloader. Le listing qui suit nous montre la configuration effectuée.

```
BIOS> setenv netmask 255.255.255.0
BIOS> setenv ipaddr 192.168.0.2
BIOS> setenv serverip 192.168.0.1
BIOS> setenv rootpath "/buildroot/binaries/apf9328"
BIOS> saveenv
```

La première commande ajuste le masque réseaux de l'SBC, la deuxième ajuste l'adresse IP de l'SBC sur 2, tandis que la troisième ajuste l'adresse du serveur (PC hôte) sur 1, et la quatrième commande ajuste le chemin du répertoire sur le serveur TFTP d'où doit se trouver les fichiers images a transférer. La dernière commande `saveenv` est celle de la sauve garde des variables d'environnement. Il faut noter que le Bootloader détient un client TFTP, alors que le serveur TFTP doit être installé sur le PC hôte avec comme répertoire de partage le répertoire d'Armadeus SDK.

L'étape suivante est celle du transfert des fichiers images en mémoire centrale et de les copier en mémoire flash. Le listing suivant utilise deux commandes avec la première `tftpboot` qui est celle du client TFTP pour le transfert de la première image mémoire (celle de U-Boot) sur la RAM en commençant à l'adresse ${loadadre}. ${loadadre} est une variable d'environnement pointant sur adresse du début de l'espace libre dans la RAM de l'APF9328. La deuxième commande **`run`** exécute le fichier script **`flash_uboot`** responsable de la copie de l'image **`apf9328-u-boot.bin`** à l'emplacement adéquat sur la mémoire flash. L'opération de copie d'une image mémoire de la RAM vers la ROM est communément appelée *Flashage* dans le domaine des systèmes embarqués.

```
BIOS> tftpboot ${loadaddr} apf9328-u-boot.bin
BIOS> run flash_uboot
```

Le flashage des images mémoires noyau Linux et fsroot est identique à celui de U-Boot

```
BIOS> tftpboot ${loadaddr} apf9328-linux.bin
BIOS> run flash_kernel
BIOS> tftpboot ${loadaddr} apf9328-rootfs.arm.jffs2
BIOS> run flash_rootfs
```

A ce stade, l'intégration du logiciel de gestion est terminé, et il ne reste plus qu'a rebooter l'SBC pour démarrer avec un Linux faisant office de système d'exploitation embarqué avec le quelle toute application linux embarqué peut être transférée et exécutée.

8.6. Construction d'une application

Pour construire une application destinée a être exécutée sur l'APF9328, deux méthodes sont généralement utilisées. La première consiste à intégrer les sources de cette application avec les sources du fsroot, et sera compilée lors de la compilation des images mémoires. La seconde consiste à la compiler séparément puis la transférer sur la mémoire de la carte. Nous allons dans cette section décrire la deuxième méthode.

L'environnement de développement Armadeus SDK fournit un compilateur ARM pour une compilation hors chaine de développement usuel. C'est le `arm-linux-g++` pour le langage C++ et le `arm-linux-gcc` pour le langage C. Les commandes de compilation sont comme suite :

```
$ arm-linux-gcc -o Programme_C Programme_C.c
$ arm-linux-g++ -o Programme_CPP Programme_CPP.cpp
```

Bien sûr l'utilisation des fichiers makefile est toujours possible pour des projets plus complexes.

Pour le transfert du fichier entre le hôte et la cible, l'opération est relativement simple puisque à ce stade la carte fonctionne avec un système d'exploitation Linux, et le transfert entre les deux systèmes Linux est relativement aisé (en utilisant NFS ou TFTP par exemple).

8.7. Les cartes Armadeus et le hard real-time

À ce point de développement, la question qui se pose est celle de savoir si notre système embarqué est temps réel ou non. Selon [1] un système Linux modelé pour une utilisation embarquée est un système temps réel souple, ce qui implique que notre carte est considérée comme un système embarqué temps réel souple, donc tolérée pour une application multimédia tel que le Set-Top-Box. Quoique l'utilisation du temps réel dur reste toujours possible avec les cartes fournies par Armadeus système. La solution que propose cette dernière pour faire tourner du temps réel sur ses cartes, est de livrer avec son kit de

développement une solution open source se basant sur une utilisation conjointe d'un noyau Linux et d'un conoyau temps réel dur appelé Xenomai. Xenomai se présente sous forme d'une extension libre pour le noyau Linux lui apportant des fonctionnalités hard real-time. Il se caractérise par le support de plusieurs API temps réel puisées d'autres RTOS comme celle de VxWorks et pSOS ainsi que l'API temps réel POSIX. Cette caractéristique rend le portage d'application temps réel de ces systèmes considérablement plus facile. Quoique pour faire cohabiter les deux noyaux Linux et Xenomai sur la même architecture physique sans faire de conflit de ressources, une couche d'abstraction de ressources matérielles doit mis en place, cette tâche est accomplit par la couche d'abstraction Adeos. Ce dernier implémente une queue de signaux de ressource matérielle, et à chaque fois qu'une ressource matérielle émet un signal Adeos l'envoi au premier noyau dans la chaine des noyaux disponible, si le signal n'est pas accepté par ce dernier, il sera renvoyer au second noyau et ainsi de suite. Bien sûr dans notre cas, le premier noyau est Xenomai étant donné que c'est le noyau temps réel donc le plus prioritaire par rapport au noyau Linux.

8.8. Conclusion

Le logiciel de gestion est une interface logicielle entre le matériel et l'applicatif utilisateur, il est généralement constitué de deux entités distinctes, le Bootloader et le système d'exploitation. Son rôle est de paramétrer le matériel lors du démarrage, coordonner les différentes applications sur le système, faire partager les ressources matérielles pour les applicatifs…etc. Dans ce chapitre nous avons essayé de suivre le déroulement de la procédure d'installation du logiciel de gestion sur la carte SBC APF9328. Plusieurs points importants ont été abordés, le plus important est celui de l'analyse du processus du développement embarqué, nous avons vu que ce dernier se constitue principalement de plusieurs étapes, l'étape du développement croisé, l'étape du transfert des images mémoires du hôte vers la cible, et l'étape de l'installation proprement dite des images sur la mémoire flash de l'APF9328. La dernière partie du chapitre était consacrée au temps réel, malgré qu'il n'ait pas été implanté sur le système. Vu son importance, on a brièvement expliqué son principe de fonctionnement sur les SBC propre à Armadeus. Le chapitre suivant sera consacré au développement de l'applicatif qui sera implanté sur le logiciel de gestion de la carte Armadeus APF9328.

Implémentation de l'algorithme P2PTV 9

9.1. Introduction

Ce chapitre décrit l'implémentation du protocole DONet qui va être implanté sur la carte SBC APF9328. La question qui se pose dans ce chapitre est pourquoi choisir DONet parmi beaucoup d'autres protocoles P2PTV qui font légion dans le domaine de la recherche. La réponse se relate spécialement aux différents avantages qu'il peut offrir par rapport à d'autres protocoles. Le principal avantage de DONet est sans doute sa simplicité d'implémentation, car par rapport aux autres protocoles, DONet n'exige pas une structure globale complexe pour maintenir son architecture, il se base en grande partie pour son maintient sur le protocole épidémique SCAMP, que même s'il n'est pas totalement intégré par DONet conserve toujours son caractère simpliste et totalement distribué. Le deuxième avantage de DONet non négligeable est son efficacité, qui se présente par la révocation de la redondance dans le transfert de paquets entres pairs. DONet y parvient en laissant la liberté à ses pairs d'exprimer eux-mêmes leurs besoins en flux auprès d'autres pairs, contrairement à d'autres protocoles dans lesquels des entités externes au pair décide de l'envoie d'un paquet donné a ce dernier, et ce genre de procédé engendre en général l'envoi du même paquet à plusieurs reprises pour le même pair. Le troisième avantage de DONet est sa robustesse ; étant un protocole à base de mailles sa tolérance aux churn est considérablement accrue par rapport aux protocoles à base d'arbres. Il présente même de bonnes performances en comparaison à d'autres protocoles basés mailles, cela revient principalement pour son utilisation d'un algorithme de gestion de partenaires relativement rapide dans sa détection et son réajustement de la liste de ses partenaires face aux abandons de ces derniers. DONet présente aussi comme avantage un scheduler intelligent quasi temps réel capable d'assurer un taux élevé de continuité de flux même avec une population relativement instable.

Ce chapitre est scindé globalement en trois parties, la première est consacrée à la description de l'implémentation du protocole DONet, dans laquelle le fonctionnement de l'ensemble des compartiments d'un nœud DONet est exposé. La deuxième partie est consacrée à la description du protocole SCAMP, étant donné que ce dernier représente une grande partie de la procédure du maintient de l'architecture DONet. La troisième et la dernière partie est consacrée au travail de simulation effectué sur le pair implémentant DONet sur l'SBC APF9328. Ce travail de simulation est nécessaire pour notre projet parce que à notre avis c'est le moyen le plus académique pour exposer l'état de marche du Set-Top-Box développé dans notre mémoire.

9.2. Le protocole DONet

DONet est l'un des protocoles P2PTV basé mailles les plus reconnu dans le domaine de la recherche. Il fonctionne selon un principe reposant sur une stratégie relativement simple, dans laquelle chaque nœud échange périodiquement des informations de disponibilité sur les données qu'il détient avec un ensemble de partenaires appelés voisins. Ces informations sont utilisées pour récupérer les données qui lui manque du groupe de ses partenaires, et en contre partie il leurs fournit les données dont ils ont besoin. Pour le maintien de sa topologie, DONet utilise une méthode distribuée appartenant aux méthodes *épidémiques*, plus précisément il utilise le protocole Scalable Gossip Membership protocol (SCAMP). Le principe de base dans le protocole SCAMP est qu'un nœud diffuse un message pour un ensemble de pairs choisis aléatoirement, ces derniers font la même chose pour d'autres ensembles de pairs et ainsi de suite, jusqu'à ce que le message soit distribué à toute la population du système. L'utilisation d'une distribution aléatoire par rapport a une distribution structurée a l'avenage d'être très robuste aux défaillances aléatoires des pairs, et permet une décentralisation quasi uniforme. La *figure*9-1 schématise le diagramme fonctionnel des principaux compartiments d'un nœud DONet, ces derniers seront détaillés dans les sections qui suivent.

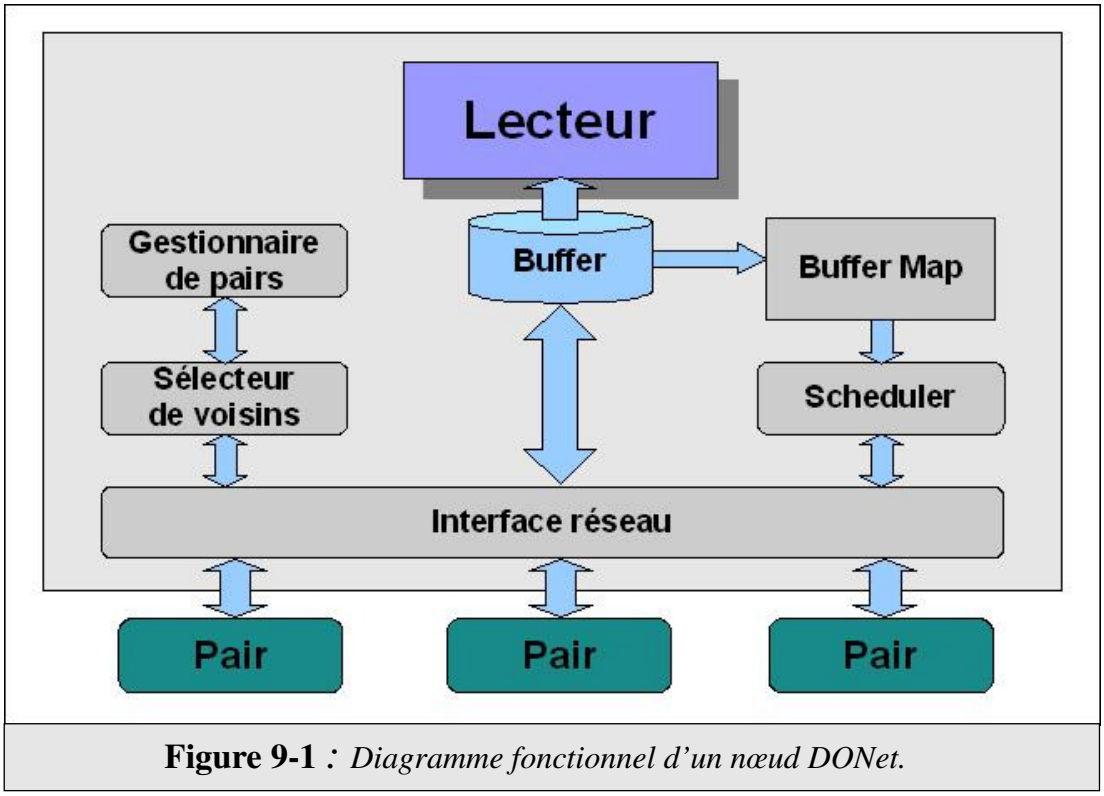

Figure 9-1 : *Diagramme fonctionnel d'un nœud DONet.*

9.2.1. Le gestionnaire de pairs

Chaque nœud DONet possède un identificateur unique, dans notre implémentation c'est son adresse IP. Il maintient un cache pour une vision partielle (vue locale) sur les différents pairs dans le système. Lorsqu'un nouveau nœud veut adhérer au système, il envoie une requête d'adhésion au nœud d'origine (le nœud serveur). Ce dernier choisit aléatoirement un nœud de son cache pour lui servir de nœud de substitution, le nouveau nœud va être redirigé vers ce nœud de substitution qui va lui retransmettre sa vue locale. En utilisant cette vue le nouveau nœud peut contacter les pairs en suivant les étapes de la procédure d'inscription de SCAMP et former son propre groupe de voisins (la procédure d'inscription SCAMP sera détaillée un peu plus loin dans ce chapitre). Il faut noter que dans cet algorithme chaque nouveau nœud doit impérativement en premier lieu contacter le nœud d'origine, pour la simple raison que ce dernier est le seul nœud a perdurer tout au long de la session vidéo, c'est aussi le seul nœud avec un identificateur connu universellement par tous les nœuds du système. Les concepteurs de DONet ont songé a utiliser la redirection aléatoire vers un substitut du nœud d'origine pour deux principales raisons, premièrement ça permet de minimiser la charge sur le serveur d'origine, et deuxièmement de s'appuyer sur le facteur aléatoire pour unifier la distribution de la sélection des voisins.

L'algorithme du maintient du cache des pairs dans le gestionnaire des pairs suit la procédure suivante. Chaque nœud génère périodiquement des messages d'adhésions aux pairs de son cache en suivant l'algorithme de diffusion épidémique SCAMP. Le message est constitué des 4 champs suivants [`numéro_séquentiel` , `identificateur` , `nombre_voisins` , `time_to_live`] où `numéro_séquentiel` est le numéro séquentiel du message, `identificateur` est l'adresse IP du nœud, `nombre_voisins` est son nombre de voisins, et le `time_to_live` donne le temps de péremption du message. De l'autre côté, lorsqu'un nœud reçoit un message d'adhésion avec un numéro séquentiel supérieur ou égal au numéro séquentiel actuel, il met à jour l'entrée de son cache si l'entrée du message existe sinon il crée une nouvelle entrée. Les entrées du cache se compose de cinq champs, il sont listé comme suite [`numéro_séquentiel` , `identificateur` , `nombre_voisins` , `time_to_live` , `derniere_MAJ`]. Les trois premiers sont directement copiés du message d'adhésion, tandis que le champ `time_to_live` est décrémenté par la valeur de la soustraction `temps_actuel` - `derniere_MAJ`, en d'autres mots il décrémente le temps écoulé entre le message actuel et le message précédent. Le champ `derniere_MAJ` reçoit la valeur de `temps_actuel` (il faut noter que les valeurs de `temps_actuel` et `derniere_MAJ` sont des valeurs temporelles acquises de l'horloge locale du nœud). Dans le cas où la valeur de `time_to_live` est inferieur à zéro l'entrée sera supprimé, sauf si le nœud en question appartient a la liste des voisins du nœud, ou si le message doit être retransmit selon l'algorithme de diffusion de SCAMP.

9.2.2. Le Buffer et le Buffer Map

Le flux vidéo dans le protocole DONet comme tout autre protocole P2PTV doit être segmenté en petits morceaux et diffusé sur le réseau. Pour que le pair puisse lire le flux vidéo il doit rassembler les morceaux vidéo et les stocker d'une façon séquentielle dans un buffer, permettant ainsi au lecteur du pair de lire dedans les segments sous forme d'un flux vidéo

assemblé (comme illustré sur la *figure*9-2). Le Buffer Map est une structure de donnée indiquant la présence ou l'absence d'un segment vidéo dans le Buffer. Comme on peut le voir sur la *figure*9-2, c'est une variable de N bites, chaque bite de la variable indique l'état d'un morceau vidéo dans le Buffer (1 = présence du morceau, 0 = absence du morceau). Le Buffer Map contient aussi une autre variable qui représente le numéro séquentiel du morceau actuellement en lecture par le pair (premier morceau dans le Buffer), cette variable permet aux schedulers de déduire l'intervalle des numéros séquentiels des morceaux dans un Buffer donné. Chaque pair échange périodiquement avec ses voisins son Buffer Map pour la principale raison de permettre au scheduler de décider de quelle voisin il peut se procurer un morceau vidéo donné, ça permet aussi aux pairs de connaitre l'état fonctionnelle de ses voisins. Notre implémentation assume que chaque morceau vidéo contient 1 seconde de flux vidéo, avec un Buffer pour chaque nœud de 120 secondes. Ainsi le Buffer Map est de 120 bit et la variable qui pointe sur le morceau en cours de lecture est de 2 octets, avec laquelle le flux vidéo peut être segmenté en plus de 18 heures de lecture vidéo.

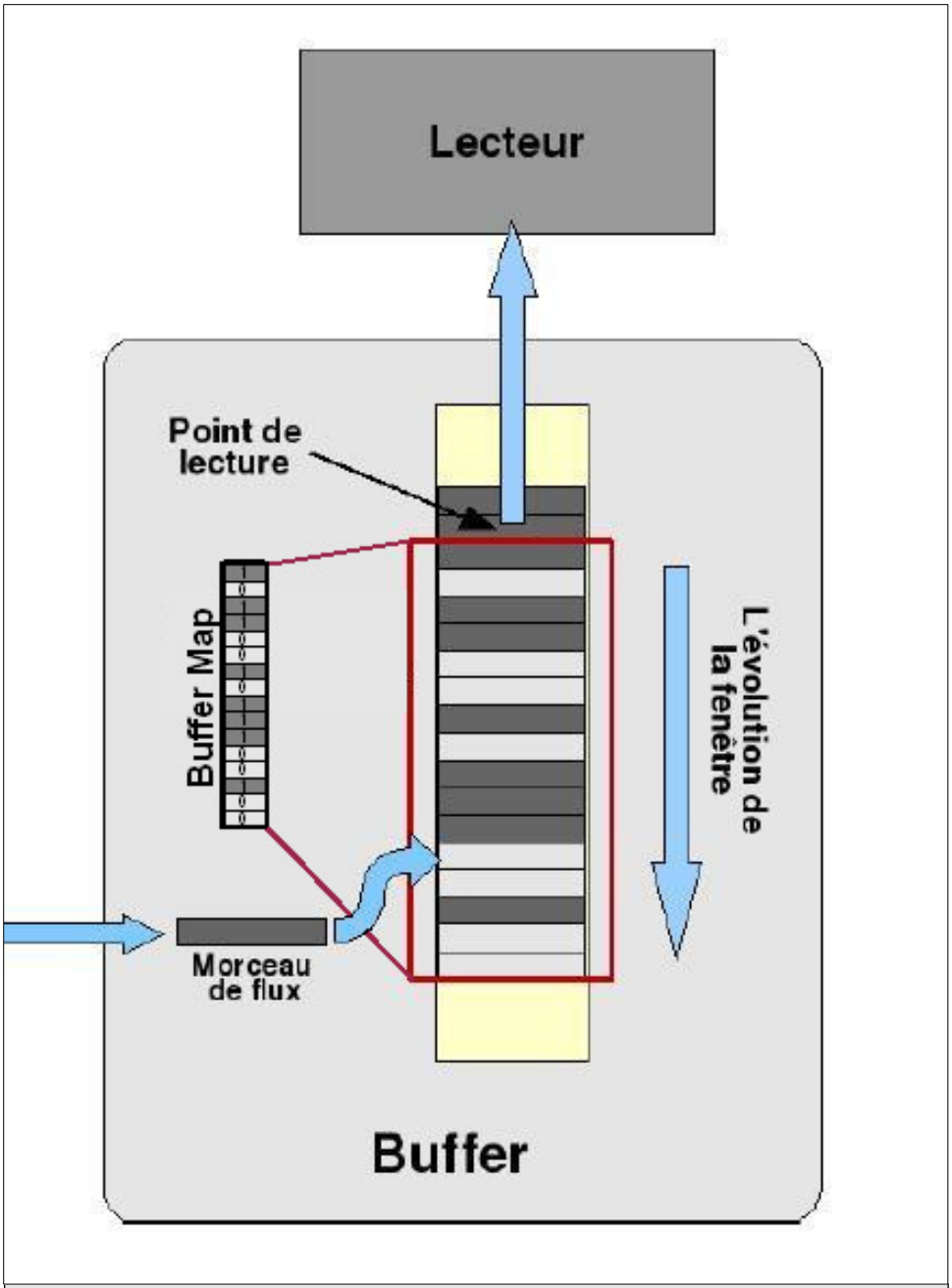

Figure 9-2 : *Schéma représentatif du Buffer et du Buffer Map.*

9.2.3. L'algorithme du scheduler

Le scheduler est l'entité logicielle responsable de la planification de la transmission vidéo entre pair et voisins. Le scheduler du DONet possède comme données principales dans son entrée, l'ensemble des Buffer Map des voisins du pair, leurs bandes passantes, et le deadline de ses propres morceaux. De cette entée le scheduler génère le planing des morceaux vidéo qui doivent être téléchargés et de quel voisin. Tandis que la sortie de l'algorithme est une file d'attente dans laquelle chaque élément est un binôme constitué du morceau à

télécharger et de l'identificateur du voisin dont il doit être téléchargé. Pour qu'un voisin puisse connaitre les morceaux vidéo à transférer, le shuduler du pair lui envois une structure identique à celle du Buffer Map, dans laquelle chaque bit indique que le pair désire recevoir le morceau du voisin s'il est à 1, et le cas contraire si le bit est à 0 (cette structure est appelée Bufer Map Like ou BM-Like), plus une liste qui indique l'ordre du transfert. Dans le cas général, les algorithmes pour scheduler visent à résoudre un problème d'optimisation mathématique, dont les contraintes sont le deadline des morceaux vidéo et l'hétérogénéité des bandes passantes pour chaque voisin, si la première contrainte ne peut pas être satisfaite, le nombre de morceaux manqués doit être tenu au minimum. Ce genre de problème est connu généralement sous le nom de *Parallel machine scheduling*, ce sont des problèmes NP-hard, et une solution canonique prend trop de temps pour des réseaux réputé dynamique, ainsi des heuristiques sont généralement utilisées. Le scheduler DONet utilise une heuristique relativement simple, il va en premier lieu calculer le nombre de voisins potentiels pour chaque morceau manquant au pair. Le morceau avec le moins de voisins potentiels est probablement le plus susceptible d'atteindre le deadline, c'est pour cette raison que l'algorithme commence avec les voisins fournissant un seul morceau et les inserts en premier dans sa file d'attente, pour continuer ensuite avec ceux à deux morceaux, à trois et ainsi de suite. Pour les morceaux avec des voisins potentiels supérieur ou égale à deux, l'algorithme a de multiple choix dans la liste des voisins, dans ce cas il choisit celui avec la plus grande bande passante, à condition que ce dernier puisse transférer le morceau sans manquer son deadline. Le listing du pseudo-code de l'algorithme est exposé ci-dessous.

```
Entrée:

      bande[k]     : bande passante du voisin k      ;
      bm[k]        : buffer map du voisin k          ;
      deadline[i]  : deadline du segment i           ;
      segSize      : taille du morceau vidéo         ;
      numVoisins   : nombre de voisins du nœud       ;
      ensVoisins   : ensemble des voisins du nœud    ;
      segManquant  : ensemble des morceaux vidéos manquants   ;

Algorithme:

      Pour  (morceau i ∈ segManquant)
            n ← 0     ;
            Pour   j allant de 0 à numVoisins
                  T[j,i] ← deadline[i] - tempsActuel     ;
                  //chaque cellule de la matrice T contient le temps
                  //restant avant la lecture du morceau i s'il doit être
                  //téléchargé du voisin j.
                  n ← n + bm[j,i]     ;
                  //n contient le nombre des voisins potentiels pour le
                  //téléchargement du morceau i.
```

```
            FinPour

            Si (n = 1)      //morceau avec un seul voisin fournisseur.
                  k ← arg_r{bm[r,i] = 1}     ;
                  //arg_r cherche l'argument r dans la colonne bm[.,i] qui
                  //remplisse la condition bm[r,i]=1, donc retourne le
                  //voisin qui contient le morceau i.
                  fileDAttente ← [k,i]        ;

                  Pour  (j ∈ segManquant) et (j > i)
                        T [.,j] ← T [.,j] - segSize/bande[k]    ;
                        //si i doit être téléchargé de k, le temps restant
                        //avant le téléchargement des autres morceaux après
                        //i   doit   être   diminué   par   le   temps   de
                        //téléchargement de i.
                  FinPour

            Sinon
                  ensDup[i] ← arg_r{bm[r,i] = 1}     ;
                  //ensDup est l'ensemble des voisins contenant le morceau
                  //i.
                  fileDAttente ← null           ;
            FinSi
      FinPour

      Pour  (i avec n > 1)   //morceau avec plusieurs voisins fournisseurs.

            k ← arg_r{ bande[r] > bande[s] telque T[r,i] > segSize/bande[r]
            et T[s, i] > segSize/bande[s] avec r, s ∈ ensDup[i]}     ;
            //arg_r retourne le pair avec la plus grande bande passante à
            //condition que le temps restant pour i reste suffisant pour
            //le téléchargement de ce dernier.

            Si (k ≠ 0)
                  fileDAttente ← [k,i]     ;

                  Pour  (j ∈ segManquant) et (j > i)
                        T [.,j] ← T [.,j] - segSize/bande[k]    ;
                  FinPour
            FinSi
      FinPour
```

Sortie:

```
fileDAttente : la file qui contient les morceaux a téléchargé et de
quel voisin     ;
```

9.2.4. Gestionnaire de voisins

Le Gestionnaire de voisins est le compartiment responsable de la formation et le maintien des liens du pair avec ses voisins. Dans le protocole DONet, un pair construit sa liste de voisin à travers une sélection aléatoire parmi les pairs de sa vue locale, cette opération présente deux avenages, premièrement chaque nœud peut maintenir un nombre stable de voisins tolérant ainsi les abandons fréquents qui peuvent avoir lieu dans un enivrement dynamique, et deuxièmement ça permet aux pairs de chercher des voisins de meilleure qualité en rejetant le voisin qui présente la plus mauvaise qualité par rapport aux autres. La qualité dans DONet est calculée par la fonction MAX($S_{i,j}$, $S_{j,i}$), où $S_{i,j}$, c'est la moyenne du nombre de morceaux vidéo que le nœud *i* reçoit du nœud *j* par unité de temps. En d'autres termes, un pair choisi son voisin en fonction de la moyenne de son débit, que sa soit entrant ou sortant. Des expérimentations effectuées par les concepteurs de DONet sur le nombre de voisins pour un pair donné ont montré que ce nombre était adéquat pour la valeur 4. Ces expérimentations ont été effectuées sur l'indice de continuité, ce dernier représente le taux de morceaux vidéo présents dans le buffer avant ou pendent la lecture de ces morceaux. De ce fait, nous avons opté dans l'implémentation de DONet pour ce mémoire une valeur égale à 4 pour le nombre de voisins maximal qu'un pair peut tolérer.

9.3. Tolérance aux churn

Dans les protocoles P2PTV, deux genres de churn (abandon de pairs) sont susceptibles d'avoir lieu, les abandons volontiers et les abandons accidentels. Dans le protocole DONet les cas d'abandon volontiers doivent générer un message d'abandon identique à ceux des messages d'adhésion, avec la seule différence dans le champ `nombre_voisins` qui est mis sur la valeur -1. Pour le cas des abandons accidentels, la détection se fait par les échanges de Buffer Map entre voisins, lorsque un voisin dépasse un certain délai avant d'émettre son Buffer Map, les autres voisins vont conclure que ce dernier est hors fonctionnement, dans ce cas les messages d'abandons vont être générés par ses voisins. Les messages dans le cas d'abandons accidentels sont identiques dans leurs formats à ceux générés par les abandons volontiers, et ils sont aussi comme ces derniers diffusés en utilisant le protocole SCAMP. Il est fort probable que plusieurs voisins détectent l'abandon et génèrent leurs messages quasi instantanément, dans ce cas les nœuds recevant ces derniers ne vont prendre en compte que le premier message qui va à son tour être rediffusé et supprimer les autres. Chaque nœud recevant un message d'abandon va supprimer le nœud de son cache, et mettre à jour la liste de ses voisins.

9.4. Le protocole épidémique SCAMP

Le principe de base dans le protocole SCAMP est qu'un nœud diffuse un message pour un ensemble de pairs choisis aléatoirement dans sa vue locale, ces derniers font la même chose pour d'autres ensembles de pairs et ainsi de suite, jusqu'à ce que le message soit distribué à toute la population du système. L'utilisation d'une distribution aléatoire par rapport a une distribution structurée a l'avenage d'être très robuste aux défaillances aléatoires des pairs, et permet une décentralisation quasi uniforme. Pour la diffusion des messages, SCAMP ne précise pas une stratégie bien claire pour le faire, et laisse le champ libre aux concepteurs de choisir la stratégie qui leurs convient. Les stratégies de diffusion de messages épidémiques se différencient en général sur trois points, la fonction aléatoire du choix du pair, la fonction aléatoire du nombre de pairs choisit dans la vue locale, et le nombre de tours de propagation, ce dernier paramètre représente le nombre de fois que le même message revenant au pair doit être rediffusé. Dans notre implémentation nous avons opté pour une stratégie relativement simple, dans laquelle le choix et le nombre de pairs est une fonction probabilistique suivant une distribution uniforme, et un nombre de tours égale à 1 (le premier message sera diffusé tandis que les autres serons supprimés). En ce qui concerne la gestion des pairs (il faut bien faire la distinction entre la diffusion des messages et la gestion des pairs dans les protocoles épidémiques), le protocole SCAMP comporte trois entités, celle de l'inscription, de la désinscription, et la tolérance au churn. Ils sont explicités ci-dessous comme suite.

9.4.1. Procédure d'inscription

La procédure d'inscription de SCAMP est utilisée par le gestionnaire de pairs du protocole DONet dans le compartiment gestionnaire de pairs. Les étapes de la procédure pour inscrire un nouveau pair dans une population sont décrites comme suite :

Le contact - quand un nouveau nœud rejoint la population d'un système pair-à-pair il envois une requête de contacte à un pair choisit aléatoirement dans la population, le pair est appelé pair de contact et son rôle est de transmettre au nouveau nœud sa vue locale pour qu'il puisse continuer la procédure d'inscription. Il faut noter que cette procédure était un peu différente dans DONet puisque le nœud de contact dans cette dernière était un nœud de substitution choisi par le serveur, les raisons de ce fait sont expliquées dans la section gestionnaire de pairs.

L'inscription - le nouveau nœud envoie le message de son inscription à tous les pairs dans sa vue locale. Il génère aussi *c* autres messages d'inscriptions pour les envoyer à *c* pairs choisit aléatoirement dans la vue locale (*c* est un paramètre pour l'algorithme SCAMP, il permet d'ajuster la tolérance de l'algorithme par rapport aux churn, dans notre mémoire nous avons pris la valeur 5). Le listing du pseudo-code de l'algorithme est comme suite :

Algorithme d'inscription d'un nouveau nœud

```
Pour  (i ∈ vue locale)
      Envoyer (i , message_inscription , message_a_retransmettre)  ;
      //La fonction 'Envoyer' contient trois arguments : le premier
      //est le pair destinataire du message, le deuxième est le
      //message à envoyer, le troisième est le type de message.
FinPour

Pour   j allant de 1 à c
      //les c messages d'inscriptions additionnels
      Choisir aléatoirement i ∈ vue locale                          ;
      Envoyer (i , message_inscription , message_a_retransmettre)  ;
FinPour
```

Inscription à retransmettre - lorsque un nœud reçoit un message d'une inscription à retransmettre, il intègre l'inscrit dans sa vue locale avec une probabilité *p* qui suit la formule suivante **p = 1 / (1 + la taille de la vue locale)**. S'il ne l'intègre pas dans la vue locale il le retransmet à un autre nœud choisi aléatoirement parmi les nœuds de sa vue locale. Le message d'inscription peut être retransmit à travers plusieurs nœuds mais ne sera jamais détruit jusqu'à ce qu'il soit accueilli par un nœud dans le système. Le listing du pseudo-code de l'algorithme est comme suite :

Algorithme du maintien des messages d'inscription

```
Si (i ∉ vue locale)

      P = 1 / (1 + la taille de la vue locale)

      Si (i ∈ vue locale avec la probabilité P)
            vue locale ← vue locale + {i}    ;
      SiNon
            Choisir aléatoirement j ∈ vue locale        ;
            Envoyer(j , message_inscription ,
            message_a_retransmettre)                    ;
      FinSi

SiNon

      Choisir aléatoirement j ∈ vue locale                          ;
```

```
      Envoyer (j , message_inscription , message_a_retransmettre)  ;

FinSi
```

Accueillir une inscription - l'étape finale dans le processus d'inscription dans le protocole SCAMP est celle de la construction des liens. À vrai dire chaque nœud dans le protocole SCAMP détient deux listes de nœuds, une liste de vue locale qui détient les Ids des nœuds pour lesquels le nœud en question envoie ses messages, et une liste qui détient les Ids des nœuds dont le nœud reçoit leurs messages (appelé liste inView), en d'autres termes la liste des nœuds qui contiennent l'Id du nœud dans leurs vues locales. Donc, lorsqu'un nœud *i* veut inscrire le nœud *j* dans sa vue locale, il place l'Id du nœud *j* dans cette dernière, puis envoie un message au nœud *j* contenant son Id pour qu'il l'intègre dans sa liste inView. Pour alléger le protocole DONet du surplus de messages, les concepteurs de ce dernier non pas intégré cette étape dans leurs protocoles. Le listing du pseudo-code de l'algorithme est comme suite :

```
            Algorithme de construction des liens

Si (i veut intégrer j dans sa vue locale)

      vue locale ← vue locale + {j}                    ;
      Envoyer (j , message_Id , message_pour_inView)   ;
      //message_Id : est le message contenant l'Id de i.
      //message_pour_inView : indique au destinataire d'inclure l'id
      //dans son inView.

FinSi
```

9.4.2. Procédure de désinscription

Nous savons que chaque nœud dans une architecture utilisant le protocole SCAMP possède deux listes de nœuds, la vue locale et l'inView, ces deux listes vont être conjointement utilisées pour effectuer le processus de désinscription d'un nœud. On suppose que la liste des Ids d'une vue locale dans un nœud a désinscrire est énuméré comme suite, $i(1),\ldots,i(l)$ et les Ids dans l'inView est listé comme suite, $j(1),\ldots,j(l`)$. Le nœud a désinscrire va envoyer un message aux $j(1),\ldots,j(l`-c-1)$ pour remplacer respectivement sont Id par les Ids $i(1),\ldots,i(l-c-1)$, et il va simplement informer les $j(l`-c),\ldots,j(l`)$ de supprimer son Id de leurs vues locales mais sans les remplacer avec d'autres Ids. Il se pourrait avec ce processus que les nœuds se retrouvent avec des Ids en redondance ou avec l'Id du pair lui-même, dans ce cas il suffit de le supprimer. D'une manière générale ce procédé tend a transférer les Ids de la vue locale du nœud a désinscrire vers les nœuds de son inView, il permet ainsi aux nœuds de quitter la population sans causer trop de perturbation a cette dernière.

9.4.3. Tolérance au churn

Le procédé utilisé dans le protocole SCAMP pour remédier aux problèmes de churn est celui du heartbeat (battement de cœur), où chaque nœud envoie périodiquement un message de subsistance à tous les pairs dans sa vue locale. De ce procédé, le nœud qui ne reçoit pas ce genre de messages pour une longue période de l'un des nœuds de son inView va conclure que le nœud a quitté le système. La procédure de tolérance aux churn et la procédure de désinscription du protocole SCAMP ne sont pas implémentées par le protocole DONet.

9.5. La simulation du réseau DONet

Pour faire fonctionner l'implémentation du pair DONet installé sur la cible APF9328, nous avons utilisé le PC hôte comme simulateur d'une population de pairs interagissant avec le pair cible comme s'il était intégré dans un vrai environnement DONet constitué d'une population avec une centaine de pairs. La *figure*9-3 schématise l'interaction du pair avec le PC hôte. On peut constater dans cette dernière que l'environnement de simulation est constitué de quatre bloques majeurs, le plus important d'entre eux est sans doute le gestionnaire de réseau, son principal rôle est d'assurer la gestion et l'interaction entre le pair et les autres différents compartiments du simulateur. Le compartiment des morceaux vidéo du flux vidéo n'est qu'un ensemble de blocs de données représentant la vidéo segmentée en morceaux de duré égale à 1 seconde. Le gestionnaire des Buffer Map contient quatre Buffer Map représentant les Buffer Map que le pair devrait acquérir de ses quatre voisins. Et dernièrement le simulateur des pairs, qui simule le comportement des pairs en interaction avec la cible. Les quatre compartiments seront plus expliqués dans les sections qui suivent.

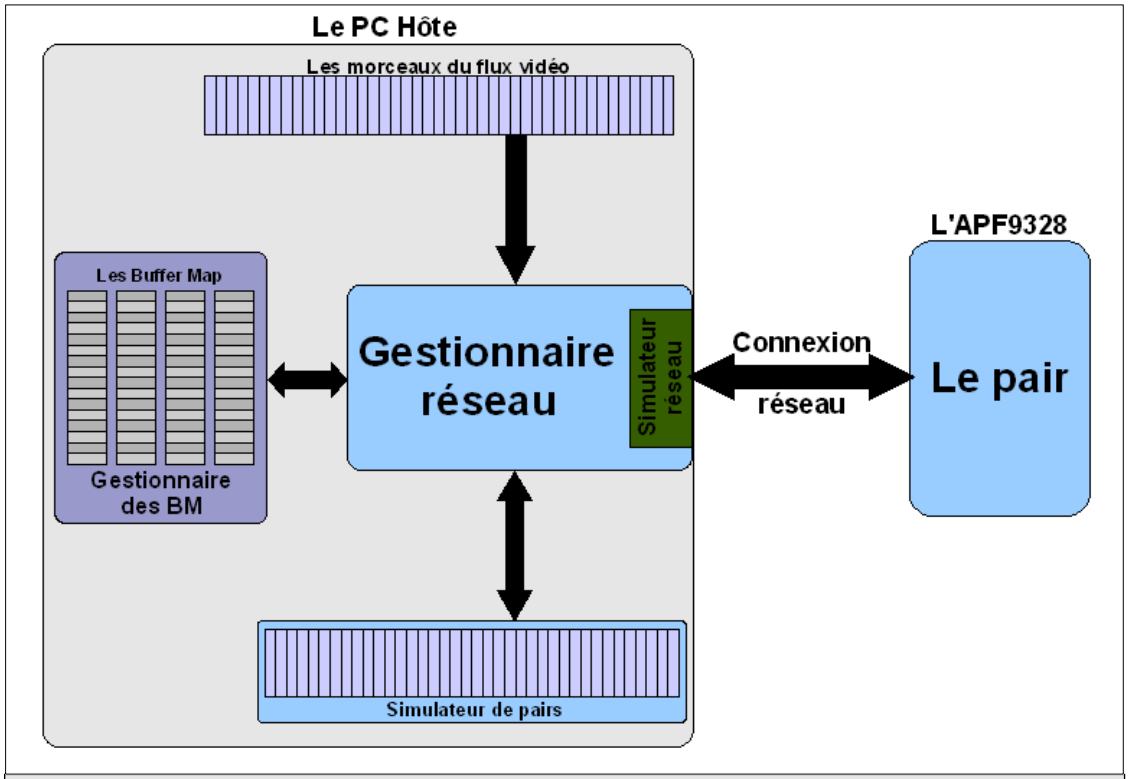

Figure 9-3 : *Schéma du pair DONet connecté au simulateur de population*

Simulateur de pairs - comme signalé auparavant, le simulateur de pairs simule le comportement des pairs en interaction avec la cible. Pour des raisons de simplicité et d'économie de ressources, le simulateur de pairs ne simule que les pairs déjà disponibles dans la vue locale de la cible. La simulation par ce dernier se scinde en trois compartiments de simulation distincts. Premièrement le compartiment responsable de la génération des pairs, ce dernier génère aléatoirement des pairs pour les incérer dans la liste des pairs du simulateur, il envoie ensuite a travers le gestionnaire réseau un message d'adhésion au nœud cible. Le deuxième compartiment est celui des abandons volontiers, dans lequel un pair est choisi aléatoirement dans la liste des pairs du simulateur pour être supprimé, et envoyer ensuite un message d'abandons au pair cible. Le troisième compartiment est celui des abandons accidentels, ce dernier supprime un pair choisit aléatoirement de la liste pour ensuite choisir un nombre aléatoire de pairs parmi les pairs de la liste du simulateur et génère à leurs noms des messages d'abandons destinés au pair cible. Il faut noter que les messages d'en provenance du pair cible ne sont pas traités par le simulateur de pair, d'ailleurs ils sont supprimés bien avant leurs arrivés a ce dernier par le gestionnaire réseau, pour la simple raison qu'ils n'ont aucune influence sur la simulation de l'interaction pairs/cible.

Le gestionnaire des Buffer Map - Le principal rôle du gestionnaire des Buffer Map est de faire office de simulateur pour les quatre voisins du pair cible. Il prend en charge la simulation de leurs quatre Buffer Map en affectant aléatoirement les bits de ces derniers par des valeurs 1 sur une période représentant la moyenne du téléchargement d'un morceau vidéo d'un pair donné. La génération des listes des BM-like se fait aussi d'une façon aléatoire, le gestionnaire des Buffer Map prend un BM-like pour un voisin donné et affecte aléatoirement ses bits par des 1 pour les bits à 0 lui correspondant dans le Buffer Map. Le transfert des Buffer Map et des BM-Like se fait en utilisant le gestionnaire réseau, ce dernier détient 4 variables représentant les Ids des quatre voisins dans le gestionnaire des Buffer Map, il se sert de ces derniers pour notifier l'un des trois compartiments : pair cible, gestionnaire des Buffer Map et simulateur de pairs, de l'abandons du voisin si l'un d'entre eux le détecte ou le simule. Lors du transfert du morceau vidéo du pair cible vers le PC hôte c'est aussi le gestionnaire de pairs qui s'en charge, le morceau vidéo sera détruit à la fin du téléchargement par ce dernier. Dans le cas contraire, lors du transfert du morceau vidéo du PC hôte vers le pair c'est le gestionnaire des Buffer Map qui fait la liaison du transfert du morceau ce trouvant dans le compartiment des morceaux vidéo vers le pair cible. En se qui concerne la formation des voisins, les concepteurs de DONet ne précisent pas une méthode bien claire pour le faire, dans notre travail nous nous somme inspiré de la méthode poigné de main utilisé dans le protocole TCP, dans laquelle le pair envoie premièrement un message d'invitation au dialogue (appelé message SYN dans le protocole TCP) au pair interlocuteur, si ce dernier accepte de dialoguer avec le pair émetteur, il lui revoie un message d'acquittement SYN/ACK, pour finir l'initiation de la connexion l'émetteur doit renvoyer un dernier message d'affirmation a son interlocuteur et c'est le message ACK. Cette solution a le sérieux avantage de permettre à un pair de tenter de nouer plusieurs connections en même temps et d'accepter celles qui lui convient sans devoir se bloquer sur l'une d'elle en attendant son acquittement.

Le gestionnaire réseau - comme signalé a auparavant, le rôle du gestionnaire réseau est d'assurer la gestion et l'interaction entre le pair et les autres différents compartiments du simulateur. Il prend en charge les messages venant et allant à la cible et selon l'action a effectuer il fait appel aux autres compartiments. Le gestionnaire de pairs peut en tout gérer les quatre types de messages de DONet interagissant avec la cible. Le premier message est celui de la requête d'adhésion au nœud d'origine, dans lequel le gestionnaire du réseau renvoie au pair la liste des pairs générés par le simulateur de pairs, ils seront pris par la cible comme vue locale du nœud de substitution pour le nœud d'origine. Le deuxième message est le message d'adhésion, il est aussi directement redirigé vers le pair cible s'il est d'en provenance du simulateur de pairs et détruit dans le cas inverse. Le troisième message est celui des Buffer Map et des BM-Like qui bi directionnellement sont transmis entre le gestionnaire des Buffer Map et la cible, et c'est pareil pour les messages de formation des groupe de voisins, quoique dans ces deux derniers types de messages, le gestionnaire réseau doit tenir en compte la gestion de la liste des voisins tenue par le pair cible et le gestionnaire des Buffer Map, il doit suivre aussi l'évolution des voisins dans le compartiment simulateur de pair. Pour le transfert des morceaux vidéo entre le pair cible et le compartiment des morceaux du flux vidéo le gestionnaire réseau est aussi responsable de la régulation de ce trafique.

Les morceaux du flux vidéo - Comme son nom l'indique, les morceaux du flux vidéo sont des morceaux vidéo rassemblés d'une façon séquentielle pour former le flux vidéo. Ces derniers sont utilisés par le gestionnaire réseau pour fournir au pair cible les morceaux qu'il doit télécharger.

9.6. Conclusion

Le protocole DONet est l'un des plus célèbres protocoles P2PTV basé mailles. Il se caractérise par un Scheduler intelligent usant de la différence en bande passante entre pairs pour maximiser la continuité du flux vidéo aux différents usagers du système. Ce chapitre dans sa globalité, tend a exposer notre implémentation du protocole DONet sur l'SBC APF9328. Tous les algorithmes utilisés par DONet ont été revus, en commençant par l'algorithme d'adhésion d'un nouveau pair au système et l'algorithme du maintient de l'architecture DONet. En continuant sur la description des deux structures de données les plus importantes de DONet, qui sont le Buffer et du Buffer Map. Les algorithmes du scheduler, de la sélection des voisins et les techniques de tolérance aux churn ont été par la suite analysées pratiquement en détail. Et nous terminons avec la description de l'intégralité du protocole SCAMS malgré que seule la partie de l'inscription de ce dernier soit utilisée par le protocole DONet. Le chapitre se clôture par la description du simulateur utilisé pour offrir a notre pair un environnement DONet capable de le faire fonctionner comme s'il était dans un vrai environnement P2PTV se constituant de plus d'une centaine de pairs DONet.

Conclusion et perspectives

L'intégration de l'Internet Protocole, le protocole le plus utilisé dans les réseaux informatiques dans le monde, dans la diffusion télévisuelle, rend cette dernière plus interactive et plus flexible que jamais. La Télévision IP n'est qu'à ses débuts, et il reste beaucoup de chemin à faire avant de déceler toutes les possibilités offertes par ce protocole. L'une des possibilités alléchantes de l'IP est sans doute le Peer To Peer. C'est une technologie largement répandue sur les réseaux IP, et a le gros avantage de réduire la charge sur un serveur de diffusion, par la contribution d'un ensemble de ses clients en tant qu'aide-serveur pour lui tenir assistance dans son émission. La forme du P2P utilisée dans l'IPTV est appelée P2PTV, ça pose beaucoup de problèmes pour l'instant, mais il est prouvé qu'elle reçoit l'intérêt d'énormément de chercheurs. L'approche P2PTV adoptée dans la plupart des travaux réalisés récemment, dont DONet en fait partie, consistent à diviser le flux vidéo en blocs, et à découper l'ensemble des pairs en petits groupes appelés maille. L'avantage d'une telle architecture, est que les mailles construisent un regroupement local de nœud relativement dense ; ce qui permet à la topologie une meilleure résistance au comportement dynamique des pairs, et a l'utilisation plus efficace de la bande passante, ainsi qu'une simplicité à l'égard de la stratégie de maintenance de l'architecture.

Le travail de ce mémoire s'inscrit dans une solution que nous préconisons, consiste à développer un système embarqué de récepteurs-TV, qui s'appuie sur l'architecture Peer-to-Peer pour la diffusion vidéo. Le nombre important de serveurs, en sera réduit et évitera les craches de surcharges liées aux taux élevés d'utilisation. Ce système embarqué est constitué d'une partie matérielle, le SBC APF9328 et d'une couche logicielle implémentant l'algorithme Peer-to-Peer TV DONet. La question qui se pose est, pourquoi choisir DONet parmi beaucoup d'autres protocoles P2PTV qui font légion dans le domaine de la recherche ? La réponse se relate spécialement aux différents avantages qu'il peut offrir par rapport à d'autres protocoles. Le principal avantage de DONet est sans doute sa simplicité d'implémentation, car par rapport aux autres protocoles, DONet n'exige pas une structure globale complexe pour maintenir son architecture. Le deuxième avantage est son efficacité, ça se présente par la révocation de la redondance dans le transfert de paquets entre pairs. Le troisième avantage de DONet est sa robustesse, c'est un protocole à base de mailles sa tolérance aux désabonnements est considérablement accrue par rapport aux protocoles à base d'arbres, et il présente même de bonnes performances en comparaison à d'autres protocoles basés mailles. DONet présente aussi comme avantage un scheduler intelligent quasi temps réel capable d'assurer un taux élevé de continuité de flux même avec une population relativement instable.

De part ces avantages, le protocole DONet a été profondément testé par ses concepteurs. Ces tests ont été portés par la simulation ainsi que par son déploiement sur un réseau Internet réel. D'après Zhang [90] DONet présente de bonnes performances même avec une large population de pairs. En ce qui nous concerne, nous avons jugé qu'il est inutile d'effectuer des expérimentations sur un protocole déjà intensément testé, mais qu'il valait mieux comme perspective future pour ce travail d'améliorer DONet en le modelant ou d'en créer un tout autre nouveau protocole plus adéquat à un usage spécifique à des pairs appartenant à un même réseau (celui du fournisseur d'accès à Internet). Les pairs se trouvant dans ce genre de situation peuvent présenter quelques caractéristiques avantageuses qu'un protocole P2PTV peut prendre en considération, comme l'homogénéité dans leurs bandes passantes ou le regroupement de plusieurs usagers sur le même nœud (leurs DSLAMs). Dans ce dernier cas, les temps de latence sont énormément réduits, et la bande passante interne au réseau (la bande passante du réseau interne du FAI est généralement supérieure à celle d'Internet) peut être utilisée.

Résumé

L'intégration de l'Internet Protocole, le protocole le plus utilisé dans les réseaux informatiques, dans la diffusion télévisuelle, rend cette dernière plus interactive et plus flexible que jamais. La Télévision IP n'est qu'à ses débuts, et déceler toutes les possibilités qu'offre ce protocole. L'une des possibilités alléchantes de l'IP est sans doute le Peer-To-Peer. C'est une technologie largement répandue sur les réseaux IP, et a le gros avantage de réduire la charge sur un serveur de diffusion, par la contribution d'un ensemble de ses clients en tant qu'aide-serveur pour lui tenir assistance dans son émission. Notre solution consiste à développer un système embarqué (un récepteurs-TV) qui implémente l'architecture Peer-to-Peer au lieu de l'architecture client/serveur pour la télévision par Internet. De ce fait le récepteurs-TV ne serait pas contraint de se procurer la diffusion vidéo du serveur mais d'un autre récepteur qui s'est déjà procuré le flux. Ainsi le FAI n'aurait pas a investir sur un nombre important de serveurs et éviter ainsi les craches causés par des surcharges dans le cas ou le taux d'utilisation est trop élevé. Le système embarqué que nous avons développé est constitué d'une partie hardware se présente sous forme d'un SBC (carte mère pour systèmes embarqués) pour accueillir le soft qui implémente l'algorithme P2PTV DONet.

Mots Clés : Système Embarqué, TV over IP, Single Board Computer, DONet, P2PTV, IPTV

Bibliographie

[1] M. Barr. Embedded Systems Glossary. *Netrino Technical Library,* www.netrino.com. Vu en janvier 2009.

[2] T. Noergaard. *Embedded systems Architecture: A Comprehensive Guide ForEngineers And Programmers Embedded Technology*. Éditions Newnes, 638 pages, Avril 2005.

[3] A. S. Berger. *Embedded Systems Design: An Introduction to Processes, Tools, and Techniques*. Éditions CMP Books, 237 pages, 2002.

[4] R. Cravotta. *2007 EDN Microprocessor/Microcontroller Directory*. EDN, 132 pages, Octobre 2007.

[5] J. Bortolazzi. T. Hirth. T. Raith. Specification and Design of Electronic Control Units. *EURO-DAC '96/EURO-VHDL '96*, Éditions IEEE Computer Society Press, Septembre 1996.

[6] Real-time systems. *The Encyclopædia Britannica 15th edition*, Éditeur Encyclopædia Britannica Inc, 1985.

[7] J. Stankovic. Misconceptions About Real-Time Computing: A Serious Problem for Next-Generation Systems, *IEEE Computer,* Vol. 21, No. 10, Octobre 1988.

[8] H. Hansson, M. Nolin, T. Nolte. *The Industrial Information Technology Handbook*, CRC Press, Editeur: Richard Zurawski, 2005.

[9] J. Xu, D.L. Parnas. Scheduling Processes with Release Times, Deadlines, Precedence, and Exclusion Relations. *IEEE Transactions on Software Engineering*, 1990.

[10] Time Triggered Technologies. http://www.tttech.com. Vu en Janvier 2009.

[11] Arcticus Systems. *The Rubus Operating System*. www.arcticus-systems.com.Vu en Janvier 2009.

[12] H. Hansson, M. Nolin, T. Nolte. *Embedded systems handbook*. CRC Press, Editeur: Richard Zurawski, 2005.

[13] OMG. *CORBA Component Model 4.0*. http://www.omg.org/technology /documents/formal/components.htm. Vu en Janvier 2009.

[14] Microsoft. *Microsoft COM Technologies*. http://www.microsoft.com/com. Vu en Janvier 2009.

[15] Microsoft. *DotNET Home Page*. http://www.microsoft.com/net. Vu en Janvier 2009.

[16] SUN Microsystems. *Introducing Java Beans*. http://developer.java.sun.com /developer/online/Training/Beans/Beans1/index.html. Vu en Janvier 2009.

[17] J. A. Stankovic. VEST - A Toolset for Constructing and Analyzing Component Based Embedded Systems. *Lecture Notes in Computer Science*, 2001.

[18] R. Ommering. The Koala Component Model. *Building Reliable Component Based Software Systems*. Artech House, 2002.

[19] P.O. Müller, C.M. Stich, C. Zeidler. Component-Based Embedded Systems. *Building Reliable Component-Based Software Systems*. Artech House, 2002.

[20] IBM. *Rational Rose RealTime*. http://www-01.ibm.com/software/rational. Vu en Janvier 2009.

[21] IBM. *Telelogic Rhapsody*. http://www.telelogic.com/products/rhapsody. Vu en Janvier 2009.

[22] IBM. *TeleLogic tau*. http://www.telelogic.com/products/tau.Vu en Janvier 2009

[23] Vector. *DaVinci Tool Suite*. http://www.vector.com. Vu en Janvier 2009.

[24] ITEA. *ATESST*. http://www.atesst.org. Vu en Janvier 2009.

[25] J. Xu, D.L. Parnas. Scheduling Processes with Release Times, Deadlines, Precedence, and Exclusion Relations. *IEEE Transactions on Software Engineering*, 1990.

[26] C. Liu and J. Layland. Scheduling Algorithms for Multiprogramming in a Hard Real Time Environment. *Journal of the ACM*, 1973.

[27] B. Sprunt, L. Sha, J.P. Lehoczky. Aperiodic Task Scheduling for Hard Real Time Systems. *Real-Time Systems*, 1989.

[28] M. Spuri, G.C. Buttazzo. Efficient Aperiodic Service under Earliest DeadlineScheduling. *Proceedings of the 15th IEEE Real-Time Systems Symposium*. IEEE Computer Society, December 1994.

[29] M. Spuri, G.C. Buttazzo. Scheduling Aperiodic Tasks in Dynamic Priority Systems. *Real-Time Systems*, 1996.

[30] L. Abeni, G. Buttazzo. Integrating Multimedia Applications in Hard Real-Time Systems. *Proceedings of the 19th IEEE Real-Time Systems Symposium*. IEEE Computer Society, December 1998.

[31] M. Spuri, G.C. Buttazzo, F. Sensini. Robust Aperiodic Scheduling under Dynamic Priority Systems. *Proceedings of the 16th IEEE Real-Time Systems Symposium*. IEEE Computer Society, December 1995.

[32] Wind River Systems Inc. *TornadoVxWorks*. http://www.windriver.com. Vu en Janvier 2009.

[33] Lynuxworks. *LynuxOS*. http://www.lynuxworks.com. Vu en Janvier 2009.

[34] Enea OSE Systems. *Ose*. http://www.ose.com. Vu en Janvier 2009.

[35] QNX Software Systems. *QNX Realtime OS*. http://www.qnx.com. Vu en Janvier 2009.

[36] *Variantes de Linux temps réel*. http://www.realtimelinuxfoundation.org. Vu en Janvier 2009.

[37] Express Logic. *Threadx*. http://www.expresslogic.com. Vu en Janvier 2009.

[38] *CubTrix*. http://sourceforge.net/projects/cubtrix. Vu en Janvier 2009.

[39] Pumpkin Inc. *Salvo*. http://www.pumpkininc.com. Vu en Janvier 2009.

[40] IEEE. *Standard for Information Technology — Standardized Application Environment Profile — POSIX Realtime Application Support (AEP)*. IEEE Standard P1003.13-1998, 1998.

[41] OSEK Group. *OSEK/VDX Operating System Specification 2.2.3*. http://www.osek-vdx.org. Vu en Janvier 2009.

[42] Airlines Electronic Engineering Committee (AEEC). *ARINC 653: Avionics Application Software Standard Interface (Draft 15)*, June 1996.

[43] R. Lehrbaum. *Single Board Computer (SBC) : Quick Reference Guide*. www.linuxdevices.com. Vu en janvier 2009.

[44] PC/104 Embedded Consortium. *SPECIFICATIONS – PCI-104.* www.pc104.org. Vu en janvier 2009.

[45] PC/104 Embedded Consortium. *SPECIFICATIONS - EBX.* www.pc104.org. Vu en janvier 2009.

[46] PC/104 Embedded Consortium. *SPECIFICATIONS - EPIC.* www.pc104.org. Vu en janvier 2009.

[47] Advantech Co. *3.5" Biscuit.* http://www.advantech.com. Vu en janvier 2009.

[48] Ampro, inc. *EnCore MODULE.* http://www.ampro.com. Vu en janvier 2009.

[49] Kontron. *ETX Computer-on-Modules.* http://www.kontron.com. Vu en janvier 2009.

[50] Telematix ,inc. *Zingu.* http://www.punchtelematix.com. Vu en janvier 2009.

[51] Applied Data ressources, Inc. *Bitsy.* http://www.applieddata.com. Vu en janvier 2009.

[52] Aleph One, Inc. *Balloon board.* http://www.aleph1.co.uk. Vu en janvier 2009.

[53] *ITU-T FG IPTV*. http://www.itu.int/ITU-T/IPTV/. Vu en Mars 2009.

[54] *Alcatel-Lucent*. http://www.alcatel-lucent.com/. Vu en Mars 2009.

[55] *Thomson, Inc*. http://www.thomson.net/. Vu en Mars 2009.

[56] *Cascade, Ltd*. http://www.cascade-ltd.com. Vu en Mars 2009.

[57] *Toroid Project*. http://linopoly.com/toroid/. Vu en Mars 2009.

[58] *Multimedia Research Group*. Ink. http://www.mrgco.com/. Vu en Mars 2009.

[59] S. Deering. *RFC 1112: Host extensions for IP multicasting*. IETF, Août 1989.

[60] W. Fenner. *RFC 2236: Internet Group Management Protocol, Version 2*. IETF, Novembre 1997.

[61] B. Cain, S. Deering, I. Kouvelas, B. Fenner, A. Thyagarajan. *RFC 3376: Internet Group Management Protocol, Version 3*. IETF, Octobre 2002.

[62] H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson. *RFC1889: RTP: A Transport Protocol for Real-Time Applications*. IETF, Janvier 1996.

[63] H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson. *RFC3550: RTP: A Transport Protocol for Real-Time Applications*. IETF, Juillet 2003.

[64] T. Friedman, R. Caceres, A. Clark. *RFC 3611: RTP Control Protocol Extended Reports (RTCP XR)*. IETF, Novembre 2003.

[65] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J.Peterson, R.Sparks, M. Handley. *RFC 3261: SIP: Session Initiation Protocol*. IETF, Juin 2002.

[66] ITU-T. Systems and terminal equipment for audiovisual services. *ITU-T Recommendation H.323*, Juin 2006.

[67] R. Stewart. *RFC 4960: Stream Control Transmission Protocol*. IETF, Septembre 2007.

[68] G. Bertrand. *The IP Multimedia Subsystem in Next Generation Networks*. Network Multimedia and Security department (RSM), May 2007.

[69] TISPAN. *ES 282 003: Resource and Admission Control Sub-system (RACS); Functional Architecture*. ETSI, Technical Report, 2006.

[70] J. Rosenberg, H. Schulzrinne. *RFC3262: Reliability of Provisional Responses in the Session Initiation Protocol (SIP)*. IETF, Join 2002.

[71] W. Marshall. *RFC 3313: Private Session Initiation Protocol (SIP) Extensions for Media Authorization*. IETF, Janvier 2003.

[72] M. Garcia-Martin, E. Henrikson, D. Mills. *RFC 3455: Private Header (P-Header) Extensions to the Session Initiation Protocol (SIP) for the 3rd-Generation Partnership Project (3GPP)*. IETF, Janvier 2003.

[73] P. Calhoun, J. Loughney, E. Guttman, G. Zorn, J. Arkko. *RFC 3588: Diameter Base Protocol*. IETF, Septembre 2003.

[74] C. Rigney. *RFC 2866: RADIUS Accounting*. IETF, Join 2000.

[75] D. Durham, J. Boyle, R. Cohen, S. Herzog, R. Rajan, A. Sastry. *RFC 2748: The COPS (Common Open Policy Service) Protocol*. IETF, Janvier 2000.

[76] R. Braden, L. Zhang, S. Berson, S. Herzog, S. Jamin. *RFC 2205: Resource ReSerVation Protocol (RSVP)*. IETF, Septembre 1997.

[77] C. Groves, M. Pantaleo, T. Anderson, T. Taylor. *RFC 3525*: *Gateway Control Protocol Version 1*. IETF, Juin 2003.

[78] Computer Desktop Encyclopedia. *P2PTV definition*. http://www.yourdictionary.com. Vu en Join 2009.

[79] R. Bolla, R. Gaeta, A. Magnetto, M. Sciuto, M. Sereno. *A measurement study supporting P2P file-sharing community models*. Elsevier, 2008.

[80] C. Diot, B. Levine, B. Lyles, H. Kassem, D.Balensiefen. Deployment issues for the IP multicast service and architecture. *IEEE Network*, 2000.

[81] Y. Chu, S. G. Rao, H. Zhang. A case for end system multicast. *ACM SIGMETRICS*, 2000.

[82] M. Bawa, H. Deshpande, H. Garcia-Molina. Transience of peers and streaming media. *ACM Computer Communication Review*, 2003.

[83] M. Bawa, H. Deshpande, H. Garcia-Molina. *Samitreng live media over peers*. Technical report, standord University, 2002.

[84] Y. Chu, S. G. Rao, S. Seshan, H. Zhang. Enabling conferencing applications on the internet using an overlay muilticast architecture. *ACM SIGCOMM*, 2001.

[85] Y. Chu, S. G. Rao, H. Zhang. A case for end system multicast. *ACM SIGMETRICS*, 2000.

[86] S. Banerjee, B. Bhattacharjee, C. Kommareddy Scalable application layer multicast. *ACM SIGCOMM*, pages 205–217, 2002.

[87] M. Castro, P. Druschel, A. Kermarrec, A. Nandi, A. Rowstron, A. Singh. Splitstream: High-Bandwidth Multicast in Cooperative Environments. *ACM SOSP*, 2003.

[88] V. N. Padmanabhan, H. J. Wang, P. A. Chou, K. Sripanidkulchai. Distributing streaming media content using cooperative networking. *ACM NOSSDAV*, 2002.

[89] R. Puri, K. Ramchandran. Multiple description source coding through forward error correction codes. *IEEE Asilomar Conference on Signals Systems and Computers*, 1999.

[90] X. Zhang, J.C. Liu, B. Li, P. Yum. CoolStreaming/DONet: A data-driven overlay network for efficient live media streaming, *IEEE INFOCOM*, 2005.

[91] V. Pai, K. Kumar, K. Tamilmani, V. Sambamurthy, A. Mohr. Chainsaw: Eliminating trees from overlay multicast. *Fourth International Workshop on Peer-to-Peer Systems*. (2005).

[92] N. Magharei, R. Rejaie, Y. Guo. Mesh or multipletree: A comparative study of live p2p streaming approaches. *IEEE INFOCOM*, 2007.

[93] M. Zhang, L. Zhao, Y. Tang, J.-G. Luo, S.-Q Yang. Large-Scale Live Media Streaming over Peer-to-Peer Networks through Global Internet. *ACM workshop on Advances in peer-to-peer multimedia streaming*, 2005.

[94] X. Zhang, J. Liu, B. Li and T.-S. P. Yum, DONet/CoolStreaming: A data-driven overlay network for live media streaming. *Technical Report*, 2004.

[95] A. J. Ganesh, A.-M. Kermarrec, L. Massoulie. Peer-to-peer membership management for gossip-based protocols. *IEEE Transactions on Computers*, 2003.

[96] Zhi-Long *Chen.Solving Parallel Machine Scheduling Problems by Column Generation*. Department of Systems Engineering, University of Pennsylvania, Philadelphia, USA. 1996.

[97] Armadeus Systeme. Armadeus Système Company. http://www.armadeus.com. Vu en Janvier 2010.

[98] Armadeus Project. *Armadeus Project Association*. http://www.armadeus.org. Vu en Janvier 2010.

[99] Armadeus Project. *Datasheet APF9328*. http://www.armadeus.com/_downloads/apf9328/documentation/dataSheet_APF9328.pdf. Vu en Janvier 2010.

[100] freescale Semiconductor. *MC9328MXL*. www.freescale.com/files/32bit/doc/ref_manual/MC9328MXLRM.pdf. Vu en Janvier 2010.

[101] ARM Ltd. *ARM9TDMI Technical Reference Manual*. http://infocenter.arm.com/help/topic/com.arm.doc.ddi0180a/DDI0180.pdf. Vu en Janvier 2010.

[102] DAVICOM Semiconductor Inc. *DM9000 Application Notes V1.0*. http://www.davicom.com.tw, Vu en Janvier 2010.

[103] Xilinx Inc. *Spartan-3 FPGA Family Data Sheet*. http://direct.xilinx.com/bvdocs/publications/ds099.pdf. Vu en Janvier 2010.

[104] Maxim Integrated-Circuit. *MAX1027*. http://pdfserv.maxim-ic.com/en/ds/MAX1027-MAX1031.pdf. Vu en Janvier 2010.

[105] Maxim Integrated-Circuit. *MAX5821*. http://pdfserv.maxim-ic.com/en/ds/MAX5821.pdf. Vu en Janvier 2010.

[106] Texas instruments. *Programmable touch screen controller with intergrated stereo audio dac and headphone amplifier*. http://focus.ti.com.cn/cn/lit/ds/slas379a/slas379a.pdf. Vu en Janvier 2010.

[107] Chrontel Inc. *CH7023/CH7024 TV Encoder*. www.chrontel.com/pdf/7023-7024bs.pdf. Vu en Janvier 2010.

[108] NXP Semiconductors. *ISP1760: Hi-Speed Universal Serial Bus host controller for embedded applications*. www.nxp.com/acrobat_download2/expired_datasheets/ISP1760_4.pdf. Vu en Janvier 2010.

[109] P. Ficheux. *Linux embarqué (deuxième édition)*. Éditions Eyrolles, 330 pages, 2005.

Abstract: The integration of the Internet Protocol, the most used protocol in computer networks, in television broadcasting, making it more interactive and more flexible than ever. IPTV is in its infancy and there remains a long way to go before identifying all possibilities offered by this protocol. One of the great opportunities offered by IP is probably the Peer-to-Peer, which is a widely used technology over IP networks, and has the great advantage of reducing the load on a streaming server, by the contribution of a set of customers as server helpers to help him keeping the right streaming. Our solution is to develop an embedded system (a TV receiver) that implements the peer-to-peer architecture instead of client/server architecture for IPTV, for that reason the TV receiver will not be constrained to procure video diffusion from the server but from another receiver that already containing the video flow. Thus the ISP would not invest on a large number of servers and avoid the spit caused by overloads in the event that the utilization rate is too high. Our embedded system consists of a hardware part or rather a SBC (motherboard for embedded systems) to host the application that implements the P2PTV DONet algorithm.

Keywords: Embedded System, TV over IP, Single Board Computer, DONet, P2PTV, IPTV

الملخص: استعمال الإنترنيت بروتوكول, البروتوكول الأكثر استعمال على شبكات الكمبيوتر, في بث برامج التلفزيون جعل هذا الأخير أكثر تفاعلي و أكثر مرونة من أي وقت مضى. **IPTV** مازال في بداية مشوار تطوره و لا يزال الدرب طويل قبل أن يبدي كل إمكانياته. لا شك أن **P2P** يعتبر من بين أهم إمكانيات **IPTV**, استعماله على شبكة الإنترنيت جد شائع و من أهم خصائصه أنه يخفف الضغط على موزعي بعث الفيديو (serveurs vidéos) بمساهمة عقد الاستقبال (nœuds de réception) كمساعدين إضافيين لنشر البعث. يتطرء مشروعنا في هذه المذكرة على تصميم نظام محمول (système embarqué) متمثل في جهاز استقبال التليفزيون عبر شبكت الإنترنت, والذي يستعمل نظام ذو هندسة **P2P** بدلا من النظام ذو الهندسة client/server الأكثر استعمالا في شبكات **IPTV**. مما يسمح لأي عقدة استقبال من أن تقتني البعث من عقد أخرى داخل النظام دون أن تكون مضطرة لاستعمال الموزع الرئيسي. بهذه الطريقة يمكن لمؤسسات توزيع الإنترنت (FAI) من تخفيف عدد موزعي بعث الفيديو وتفادي تعطيلهم في حالة تجاوز ذروة العدد الأقصى لمستعملي النظام. النظام المحمول المستعمل في هذا المشروع يتكون من جهاز SBC (لوحة إلكترونية خاصة بالأنظمة المحمولة) لاستضافة البرنامج الذي يعالج الخوارزم DONet P2PTV.

الكلمات المفتاحية: نظام محمول, SBC, TV over IP, DONet, P2PTV, IPTV