Notice Bibliographique

Auteur : Abdellatif EL OUAZZANI

Année : 2010

Etablissement : Faculté des sciences et Techniques de Fès

Spécialité : Systèmes Electroniques et Télécommunications

(3^{éme} année du cycle d'ingénieur)

Etablissement d'accueil : THALES AIR SYSTEMS, locaux de l'INPT
Avenue Allal Al Fassi, Madinat Al Irfane, RABAT

Cadre du rapport : Stage de fin d'études

Titre du projet de stage:

« Etude & Développement de logiciels de supervision Radar sur Linux pour la nouvelle génération de DSP développé par Texas Instrument »

Encadrement professionnel :

M. Philippe LEGALL : Manager Equipe THALES AIR SYSTEMS

M. Jean-Philippe HALLAY: Ingénieur THALES AIR SYSTEMS

Encadrant pédagogique :

M. Ali AHAITOUF : Professeur de l'enseignement supérieur

DGE, FST de Fès, B.P. 2202 Fès MAROC

Date de début et de fin du stage:

Début : le 08 Février 2010

Fin : le 22 juillet 2010

Dédicaces

Je suis le plus embarrassé homme du monde, lorsqu'il me faut dédier mon travail.

J'espère que je n'oublierai personne dans ce périlleux exercice, et je m'excuse par avance auprès des malheureux écartés de cette liste, qui ne saurait de toute façon être exhaustive.

Au personne que j'ai tant aimé qu'elle assiste à ma soutenance : le regretté mon cher père qui n'a jamais cessé de me rappeler que la volonté fait toujours les grands hommes…

A mon adorable mère qui m'a beaucoup donné, et qui a toujours été là pour moi, et qui m'a donné un magnifique modèle de labeur et de persévérance. J'espère qu'elle est fière de moi,

A ma chère fiancée Ghizlane, pour sa tendresse débordée et son sacrifice perpétuel, avec elle je souhaite vivre les plus beaux jours de ma vie. J'espère qu'elle trouvera dans ce travail tout mon amour, même si je ne considère pas ce PFE comme une finalité, mais plutôt une étape

A mes chères sœurs Fatima, Naima, Souad, Najia, et leurs époux, Pour leur soutien morale et leurs sacrifices le long de ma formation.

A l'adorable petite sœur Charafa.

A mes chers frères Mohammed, Ismail, El Mahdi,…

A mes tantes et à mes oncles.

A mes chères nièces, et mes chers neveux.

A chaque cousin et cousine, et mes meilleurs amis :

À toutes les personnes qui sans eux, je n'aurai certainement pas pu autant affirmer, structurer et consolider ma pensée.

Je dédie ce mémoire.

Acceptez donc ici l'hommage de ma gratitude, qui, si grande qu'elle puisse être, ne sera jamais à la hauteur.

Remerciements

J'ai eu l'opportunité de pouvoir travailler pendant presque cinq mois entier en tant que stagiaire ingénieur (dédicace à ceux que la réforme de cycle d'ingénierie donne des boutons) au sein du soft center situé dans les locaux de l'Institut National des Postes et Télécommunications (INPT), chaque fois encadré par des gens différents et qui m'ont tous appris énormément. Bien sûr les difficultés ont été présentes, mais tous ont été surmontés avec persévérance et si l'opportunité se présente de poursuivre en thèse je sais d'avance que de la persévérance il en faudra !

Je veux tout d'abord remercier mes trois directeurs de stage M. Philippe LEGALL : Manager Equipe THALES AIR SYSTEMS , M. Jean-Philippe HALLAY: Ingénieur THALES AIR SYSTEMS et mon encadrant pédagogique : M. Ali AHAITOUF : Professeur de l'enseignement supérieur de m'avoir permis de faire un tout petit pas dans le monde des DSPs, systèmes embarqués, radars numériques, supervision… et de m'avoir fait confiance pour la démarche scientifique utilisée lors de mes manips.

Aux membres du jury de ma soutenance, qui par leur présence et leurs avis critiques, revalorisent le travail présenté. Je cite, outre mes encadrant, les professeurs A.MECHAQRANE, F.ABDI, Najia ES-SBAI, Hassane EL MARKHI, qui ont bien voulu assister, mes salutation les plus distingués et ma gratitude vous est destinée.

Ma gratitude va ensuite à tout le corps administratifs, et professoral du département Génie électrique de la Faculté de sciences et Techniques de Fès.
Mes plus vifs remerciement à M. M.LAHBABI responsable de la formation, qui a resté attentif à l'évolution de toute la promotion, espérons que ce modeste travail sera à la hauteur de ses espérances nous concernant.

Mes remerciements à tous mes chers professeurs qui nous ont offert disponibilité, et nous ont prêté savoir faire afin d'être apte à réaliser un sujet pareil.

Mon séjour à L'INPT m'a permis de découvrir des gens dont le côté humain et les compétences scientifiques m'ont beaucoup marqué. Merci à M. Charif CHEFCHAOUNI, le directeur de l'INPT, qui a bien voulu

m'accueillir au sein son établissement et à tous le personnel pour sa gentillesse et son soutien.

Merci aussi à tous ceux qui font le groupe de travail : AWADA Haidar (INSA Toulouse), EL OUESLETI Souhayeb (ESIEE Paris). EL khaiter Tarik (FSK), Amar Mançour Abderrahim (EMI), Anbar Hicham (EMI), toutes ces personnes ont contribué au bon déroulement de mon séjour à Rabat.

À tous mes camarades de la première promotion des élèves ingénieurs SET3 2010, avec qui j'ai partagé trois années intenses de formation.

Enfin, ce stage s'est déroulé comme je le convoitais et les gens qui composent l'équipe de ce projet, sont des gens dont l'humanisme m'a beaucoup marqué. Je suis ravi d'avoir pu effleurer un niveau aussi poussé de perfection dans le domaine des sciences. J'espère en profiter encore.

Table des matières

Liste des Tables

Liste des Figures

Introduction générale

Le monde a assisté à l'apparition de nouvelles formes de conflits armés. De nouvelles classes de menaces ont vu le jour (attentats terroristes, Missile balistique à longue portée...). Ces dernières se caractérisent par leur variété et leur capacité à engendrer un grand nombre de victimes surtout auprès des populations civiles.

L'industrie de l'armement et de la défense doit apporter une solution traitant cet éventail de menaces. C'est ainsi qu'aujourd'hui on se tourne vers des radars multi missions et pouvant subvenir aux besoins des missions de contrôle aérien, de la sécurité intérieure et d'observation météo. Les radars multi missions sont l'avenir incontournable de l'industrie des radars et seront exclusivement logiciels.

Dans ce contexte, on aura besoin de traiter des algorithmes de traitement de signal avec débit élevé par un réseau de microprocesseurs placés à l'antenne du Radar.

L'arrivée sur le marché des microprocesseurs DSP TMS320C6474 de Texas Instrument de 3GHz de fréquence va améliorer drastiquement les performances des unités de traitement et permet donc d'embarquer des algorithmes de traitement du signal massif qui s'exécutaient auparavant sur des DSP moins performants.

Vu l'important nombre de microprocesseurs intégrés dans un seul RADAR, la présence de microprocesseurs dans le réseau, dédiés au suivi et la supervision de l'ensemble des entités, est nécessaire. Le projet consiste dans un premier point, à porter l'ensemble des logiciels de supervision, tournant sur des processeurs ColdFire, sur la nouvelle carte, que Thales souhaite intégrer dans ses radars, soit TMS320C6474. Dans un deuxième point, le projet consiste à étudier la faisabilité de la communication inter cœurs et inter DSP via le protocole SRIO afin d'exploiter efficacement cette architecture et paralléliser les tâches.

Chapitre 1 : Contexte du stage

1 Contexte du stage

1.1 Présentation de Thales

Thales est leader mondial des systèmes d'information critiques sur les marchés de l'aéronautique et de l'espace, de la défense et de la sécurité. En maîtrisant les grands systèmes logiciels, Thales répond aux besoins de ses clients civils et militaires.

Pour répondre à la forte progression de la demande de sécurité et saisir les opportunités de croissance sur ses marchés, Thales mobilise des savoir-faire centrés sur les systèmes d'information critiques, les moyens de communication sécurisés, les systèmes de supervision et les équipements de détection. Le Groupe propose une gamme complète de solutions et de technologies permettant d'apporter des réponses adaptées aux besoins de ses clients gouvernementaux et institutionnels, avec lesquels il noue des relations de long terme, fondées sur la proximité et la confiance indispensables à la conduite de projets complexes dans le domaine de la sécurité : avec les forces de sécurité militaires et civiles en premier lieu, mais aussi avec les autres autorités publiques, ainsi que les grands opérateurs d'infrastructures sensibles et les grands avionneurs civils.

Avec une stratégie solidement ancrée dans sa présence sur l'ensemble de la « chaîne de valeur » (des équipements et systèmes à l'intégration des systèmes, en passant par la maîtrise d'œuvre et les services), un portefeuille équilibré de technologies et d'activités duales (civiles et militaires) et une présence multidomestique en tant qu'acteur local sur les marchés nationaux, le Groupe Thales a un brillant avenir devant lui !

1.1.1 Chiffres clés

Voici quelques chiffres et données concernant le Groupe Thales :

Création	1892
Siège social	Neuilly-sur-Seine France
Direction	Luc Vigneron- PDG
Activité	Aéronautique, Défense, Technologies de l'information
Effectif	67 028 effectifs gérés au 31/12/07
Capitalisation	7.5 Mds € (Septembre 2008)
Chiffre d'affaires	12.3 Mds €
Résultat net	887 M €

Tableau 1: Chiffres et données sur Thales

- **Actionnariat**

Jusqu'à 31 mai 2009, la répartition de l'actionnariat du groupe Thales est donnée par le graphe suivant:

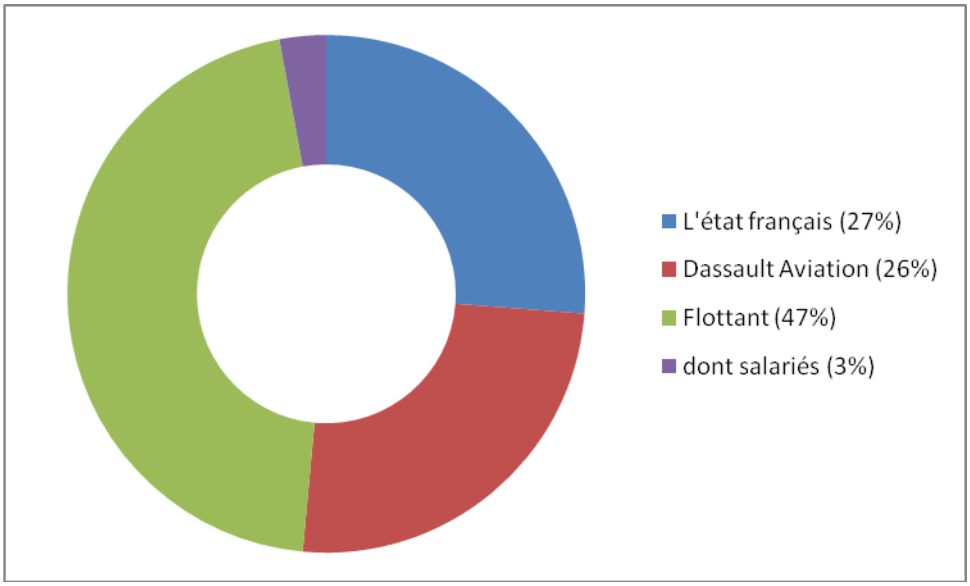


Figure 1: Répartition du capital de Thales au 31 mai 2009

1.2 Présentation de projet

Par la suite, sont exposés les points clés de projet de mise en place de la supervision Radar sous Linux sur le dernier DSP multi cœur TMS 320C6474 de Texas Instruments :

- Description technique
- Organisation
- Planification
- Environnement de travail

1.2.1 Description technique des travaux

1.2.1.1 Contexte

Dans le cadre du développement de radar numérique nouvelle génération, Thales souhaite:

- Installer et évaluer un LINUX déjà opérationnel sur la nouvelle génération de DSP développé par Texas Instrument
- Porter les logiciels de supervision appelés CCC Host (Sup appli, SNMP, DHCP, HTTP, FTP) s'exécutant dans un environnement Linux embarqué de la cible ColdFire sur le DSP TMS 320C6474
- Etudier les mécanismes de communication Inter-cœurs
- Etudier les Mécanismes de communication Serial rapid IO
- Etudier le benchmark applicatif
- Etudier les performances et Installer le DSP/BIOS

Le DSP TMS 320C6474 est un *System On Chip* (SOC) composé de 3 cœurs C64X+, de mémoire cache L1/L2, de plusieurs réseaux de communication à haut débit (Ethernet Gigabit, Serial RapidIO, liaison série 3Gb/sec à encodage 8b/10b).

La supervision s'exécute sur le kit d'évaluation EVM6474 qui sera considéré comme un abonné du réseau SUPLINK à la supervision centrale du radar numérique nouvelle génération en cours de développement. Cette supervision pilote et surveille des abonnés dont on simulera leur comportement.

Pour mettre au point cette application, on disposera d'un PC sous Linux mettant en œuvre les outils d'exploitation radar et des outils de cross compilation et debug pour cible TMS 320C6474.

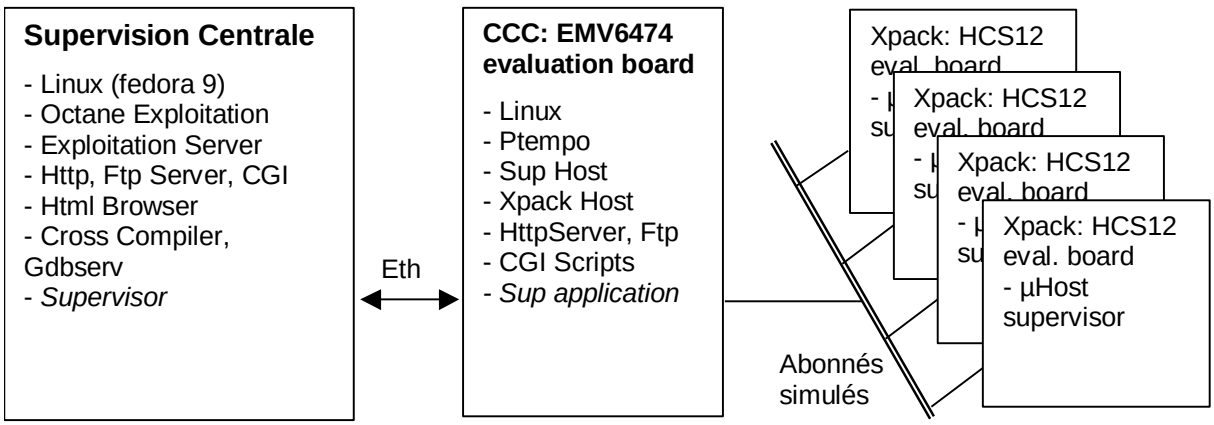


Figure 2: les logiciels de base de la supervision

1.2.1.2 Objet

Pour réaliser ce portage, une approche incrémentale est proposée en 5 itérations successives :

- Le 1er incrément consiste à mettre en place l'atelier de développement, la prise en main du DSP et du noyau Linux fourni par Texas,
- le 2ème incrément consiste à la prise en main de la couche de communication 'smart light', des protocoles TCP, UDP, DHCP et SNMP puis à mesurer les performances temps réel et robustesse de ces protocoles,
- le 3ème incrément consiste à porter les mécanismes de supervision, de CGI et des processus d'abonnés simulés (Xpack, DSP etc) et d'intégrer la supervision CCC EVM6474 dans la supervision centrale du radar numérique de nouvelle génération en cours de développement,
- le 4ème Incrément consiste à étudier les procédures et les mécanismes de communication Inter-cœurs et de la communication inter-DSP (serial rapid IO). Cela nous permettra d'étudier du benchmark applicatif. Autrement dit, étudier les briques de base de la supervision.
- Le 5ème Incrément consiste à étudier et à installer DSP/BIOS pour les autres cœurs dédiés aux applications de traitement du signal.

1.2.2 Organisation

Cette étude se décompose en 7 tâches :

1.2.2.1 Tâche 1 : Management du Programme

Cette tâche consiste à effectuer les différentes actions nécessaires au bon déroulement du PFE. On y retrouve les activités suivantes :

- gestion de configuration des documents, des logiciels et des jeux d'essais.

  Un PC de référence sera maintenu à **Limours** par le responsable technique. Le portage sera réalisé sur deux PCs à **Rungis** et un Pc à **Rabat**. Le code source nécessaire au portage sera livré par le responsable technique. A chaque fois qu'un composant de la supervision est porté, il sera livré à **Limours** pour validation et intégration.
- Rapport d'état d'avancement (avancement hebdomadaire des activités réalisées vers le responsable technique).

1.2.2.2 Tâche 2 : Mise en place de l'environnement

Cette tâche regroupe les actions nécessaires de l'incrément 1 et 2

On retrouve les activités suivantes :

- La mise en place de l'atelier (Linux, cross compilation, gdbserver, performance),
- l'étude de l'alignement des structures des messages de supervision sur l'EVM6474,
- les mesures de performances du noyau Linux,
- prise en main de la couche de communication 'smart light' et des protocoles TCP, UDP, DHCP et SNMP,
- et mesure de performance et de robustesse des protocoles de communication.

Donnée d'entrée :

- IGI,SDD CCC, Code source, fournis par Thales,
- Noyau linux, outils de cross compilation fourni par Texas Instrument,
- 3 Cartes d'évaluation et PC linux fournis par Thales

livrables à rédiger par les élèves ingénieurs :

- rapport de fabrication des logiciels,
- rapport de performance Linux et des moyens de communications
- rapport d'anomalies renseignées et transférées à Texas Instrument
- Mise en place d'un FAQ
- livraison des codes sources à Limours.

1.2.2.3 Tâche 3 : Etude du portage CCC

Cette tâche regroupe les actions nécessaires de l'incrément 3

On retrouve les activités suivantes :

- les mécanismes de supervision, des scripts CGI
- et des processus de abonnés simulés (Xpack, DSP etc)
- intégration de la supervision CCC EVM6474 avec la supervision centrale du radar numérique nouvelle génération.

livrables à rédiger par les élèves ingénieurs :

- SDD remis à jour
- Edition du rapport technique intégrant les résultats d'intégration.

1.2.2.4 Tâche 4 : Mécanismes de communication Inter-cœurs

Cette tâche consiste à étudier les différents mécanismes de la communication inter-cœurs en exploitant l'architecture de DSP TMS320C6474 et les modules sémaphores, EDMA (Enhanced Direct Memory Access) et IPC (Inter processus communication).

1.2.2.5 Tâche 5 : Mécanismes de communication Serial rapid IO

Cette tâche consiste à étudier les différents mécanismes de la communication inter-DSP via l'étude de protocole de communication à haut débit SRIO (Serial Rapid In Out) et les performances de ce dernier.

1.2.2.6 Tâche 6 : Etude du Benchmark applicatif

Cette tâche regroupe les actions nécessaires de l'incrément 4.

En effet la réalisation de cette tâche va porter sur l'intégration simultanée de la communication inter coeurs et inter DSP sur la carte EVM TMS320C6474. Par ailleurs, le cœur host (Master), chargé de gérer ces opérations, va commander à travers des interruptions les deux autres cœurs de calculs pour effectuer des traitements independants et retourner leurs traitements. Ainsi, le host va de nouveau intervenir en envoyant ces résultats à l'autre DSP de supervision via SRIO sous linux. Les résultats sont affichés sur une page http.

livrables à rédiger par les élèves ingénieurs :

- Edition du rapport technique.
- Rapport de performance

1.2.2.7 Tâche 7 : Etude Installation et performance de DSP/BIOS

Cette tâche regroupe les actions nécessaires de l'incrément 5.

livrables à rédiger par les élèves ingénieurs :

– Edition du rapport technique.
– Rapport de performance

1.2.3 Planning

La planification du projet doit bien englober et cerner toutes les étapes d'étude et réalisation. Vu que le projet est réparti sur plusieurs axes, on a bien été poussé à adopter une planification temporelle pour bien cerner les étapes de réalisations, ainsi que les livrables a produire. Cette planification est conforme aux phases du projet ainsi que les objectifs attendus, au cours de cette période, un ensemble de points de contrôle est mis en place pour bien établir un suivi du déroulement des étapes de projets, ainsi que la discussion et validation des livrables. La durée du développement est de 9 mois calendaires à partir de 11 janvier 2010. La figure suivante illustre cette planification.

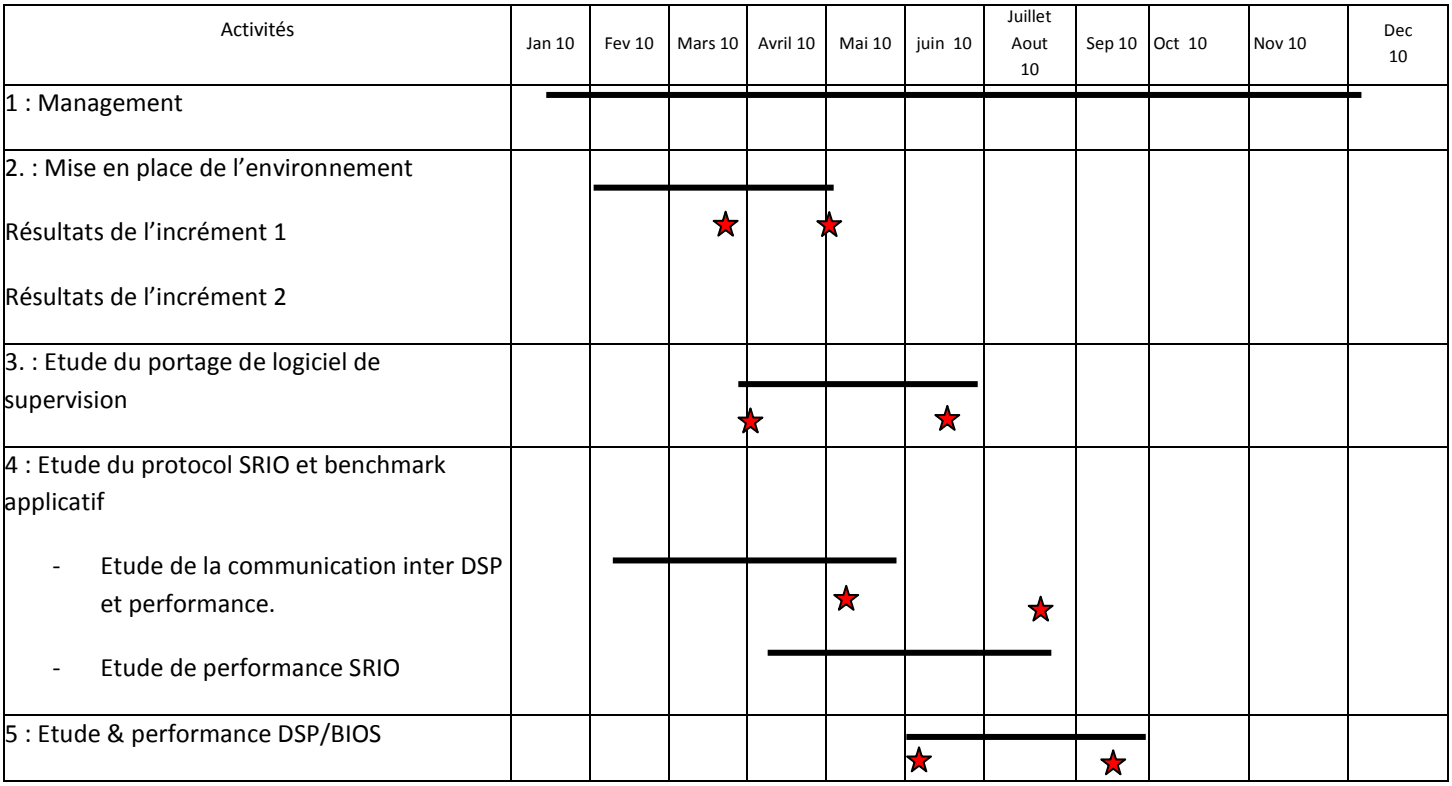


Figure 3: Planning du Projet

1.2.4 Environnement de travail

Trois étudiants stagiaires (PFE) sont affectés aux tâches 1 à 5 (répartis sur 2 site : 2 à Rungis et 1 à Rabat). Et Trois autres stagiaires vont s'atteler sur l'étude du protocole de communication inter DSP (SRIO). Ils seront pilotés par le responsable technique à raison d'un jour par semaine et encadrés par l'ingénieur chef de projet

On dispose d'un kit de développement composé de deux DSP TMS320C6474 et de l'outil de développement Code Composer Studio ainsi que le module d'évaluation de la carte. Nous disposons aussi de plusieurs documents de description du TMS320C6474 édités par Texas Instruments. Des résultats des travaux sur ce DSP effectués par d'autres équipes sont aussi accessibles. Deux ordinateurs sont nécessaires : l'un équipé de Windows XP, tandis que l'autre est équipé de Linux Fedora. A Rabat, un autre PC Intel E6550 équipé de Linux Fedora est disponible pour le test de performances.

Chapitre 2 : Le microprocesseur DSP TMS320C6474 et la supervision radar

2 Le microprocesseur DSP TMS320C6474 et la supervision radar

Le but de ce chapitre est de décrire d'une manière générale les processeurs de traitement numérique des signaux plus communément désignés par l'acronyme Anglais DSP (Digital Signal Processor). Et présenter spécialement celui de Texas Instruments le DSP TMS320C6474, leurs caractéristiques, type, architecture et domaine d'application. Dans ce chapitre, on présente dans un premier lieu et d'une manière générale les microprocesseurs de traitement de signal **[1].** Ensuit nous allons présenter le système de supervision radar où on va embarquer le TMS320C6474 au lieu de l'architecture ColdFire dans une carte de contrôle et de commande (CCC). Enfin on va présenter l'atelier logiciel dont nous disposons ainsi le noyau linux qu'on a configuré, compilé et exécuté pour notre DSP.

2.1 Le DSP

2.1.1 Introduction

Un DSP est un type particulier de microprocesseur apparu vers 1982. Il se caractérise par le fait qu'il intègre un ensemble de fonctions spéciales. Ces fonctions sont destinées à le rendre particulièrement performant dans le domaine du traitement numérique du signal**.**

Le traitement numérique du signal est une technique en plein essor. Cette technique s'appuie sur plusieurs disciplines, citons les principales :

- l'électronique analogique et numérique (générations et conditionnements des signaux, conversions numériques analogiques…),
- les microprocesseurs (classiques ou dédiés au traitement du signal),
- l'informatique (algorithmes, systèmes de développements, exploitations),
- les mathématiques du signal (traitements du signal).

Les domaines d'applications du traitement numérique du signal sont nombreux et variés (traitements du son, de l'image, synthèse et reconnaissance vocale, analyse, compression de données, télécommunication, automatisme, etc.), chacun de ces domaines nécessite un système de traitement adapté, pour un coût économique approprié.

Les microprocesseurs sont en perpétuelle évolution, chaque nouvelle génération est plus performante que l'ancienne, pour un coût moindre. Les DSPs, qui sont un type particulier de microprocesseur, n'échappent pas à cette évolution.

La figure suivante montre le rôle d'un DSP dans une chaîne de traitement numérique du signal.

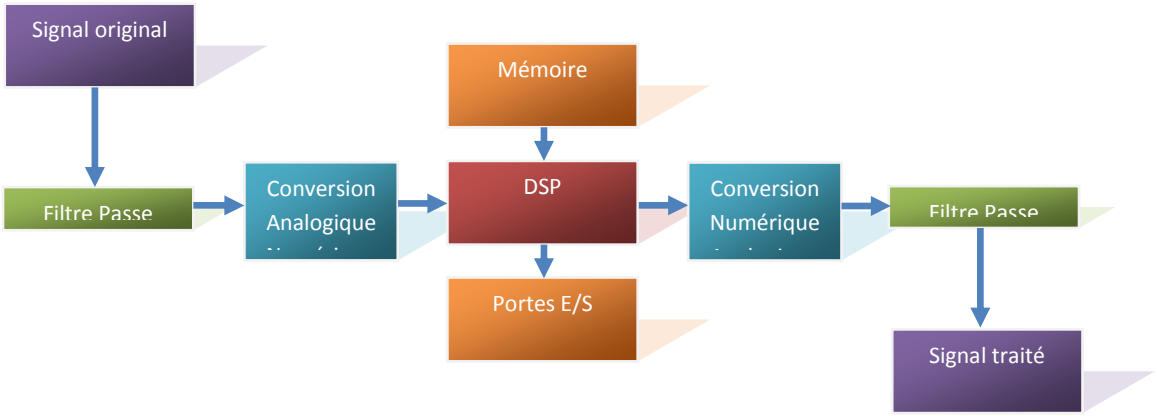


Figure 4: chaîne complète typique d'un système de traitement numérique de signal

Certains DSP de nouvelle génération intègrent tous ces composants en natif. C'est là que la rapidité devient de première importance. Généralement, les DSP sont prévus pour être installés dans un système autonome et c'est pourquoi ils intègrent presque toujours une RAM pour stocker les données temporaires, une ROM pour stocker par exemple le programme contenant les algorithmes.

On voit donc sur ce diagramme deux composants dont la qualité joue un rôle plus que fondamental : le Convertisseur Analogique Numérique (CAN) et le Convertisseur Numérique Analogique (CNA). Ces éléments influent directement sur la qualité du traitement par leur précision (précision de la référence interne, résolution sur 8, 10, 12, 14 bits ou plus). **[2]**

Tous les systèmes à bases de DSP bénéficient des avantages suivants :

- Souplesse de la programmation : un DSP est avant tout un processeur exécutant un programme de traitement du signal. Ceci signifie que le système bénéficie donc d'une grande souplesse de développement. Il supporte des programmes écrits en langage C, C++ et bénéficie des bibliothèques de calcul optimisées qui sont faites par le fournisseur de technologie).
- Implémentation d'algorithmes adaptatifs : une qualité issue de la souplesse, des programmes. il est possible d'adapter une fonction de traitement numérique en temps réel suivant certains critères d'évolution du signal (exemple : les filtres adaptatifs).
- Stabilité : en analogique, les composants sont toujours plus ou moins soumis à des variations de leurs caractéristiques en fonction de la température, de la tension d'alimentation, du vieillissement, etc. ce qui n'existe pas en numérique.

2.1.2 La Différence entre DSP et microprocesseurs ordinaires

Comme on a mentionné le DSP est un type particulier de microprocesseur, il diffère des microprocesseurs ordinaires par :

2.1.2.1 L'opération MULAC (Multiplication et Accumulation)

Le signal se présente sous la forme d'une suite de valeurs numériques discrètes. Cette suite de valeurs (ou échantillons) est prête à être stockée et traitée par un système informatique. Par nature, le traitement numérique du signal revient à effectuer essentiellement des opérations arithmétiques de base du type **A= (B x C) + D**
Un microprocesseur classique va nécessiter plusieurs cycles d'horloge pour effectuer un tel calcul, par exemple, un 68000 à besoin de :

- 10 cycles d'horloge pour effectuer une addition,
- 70 cycles d'horloge pour effectuer une multiplication.

Soit **80 cycles** pour seulement calculer **A.** Si ce temps est admissible dans des applications informatiques courantes, il n'est pas acceptable pour faire du traitement rapide du signal. Les DSP sont donc conçus pour optimiser ce temps de calcul, à cet effet, ils disposent de fonctions optimisées permettant de calculer **A** beaucoup plus rapidement.

Dans la pratique, la plupart des DSP ont un jeu d'instructions spécialisé permettant de lire en mémoire une donnée, d'effectuer une multiplication puis une addition, et enfin d'écrire en mémoire le résultat, le tout **en un seul cycle d'horloge**, c'est le cas pour le DSP TMS320C6474 objet de notre étude.

L'évolution des DSP avait comme principal objectif l'amélioration du temps de calcul d'un MULAC, car effectuer une opération MULAC en un seul cycle n'est malgré tout pas satisfaisant si le cycle d'horloge est trop « long ». **[3]**

2.1.2.2 L'accès à la mémoire

Outre l'opération MULAC, une autre caractéristique des DSP est leurs capacités à réaliser plusieurs accès mémoire en un seul cycle. Ceci permet à un DSP de chercher en mémoire une instruction et ses données réalisant un MULAC, et simultanément, d'y ranger le résultant du MULAC précédent. Le gain de temps est évident. Toutefois, sur certains DSP basiques, ce type d'opérations simultanées est généralement limité à des instructions spéciales. Ces instructions utilisent un mode d'adressage restreint, c'est à dire ne portant que sur de la mémoire vive intégrée au DSP.

Les modes d'adressages des données sont un point particulier des DSP. Un DSP peut posséder plusieurs unités logiques de génération d'adresse, travaillant en parallèle avec la logique du cœur du DSP. Une unité logique de génération d'adresse est paramétrée une seule fois via les registres appropriés. Elle génère alors toute seule, en parallèle avec l'exécution d'une opération arithmétique, les adresses nécessaires à l'accès des données.

Ceci permet non seulement de réaliser les accès mémoires simultanés en seul cycle, comme décrit plus haut, mais également d'incrémenter automatiquement les adresses générées. Ce mode d'adressage particulier, généralement appelé adressage indirect par registre avec post (ou pré) incrément, est très utilisé pour effectuer des calculs répétitifs sur une série de données rangées séquentiellement en mémoire. **[3]**

2.1.3 Autres Caractéristiques des DSP

Ce qui distingue deux DSP à part la virgule fixe ou flottante c'est :

- Sa vitesse, exprimée en MULAC/s: le nombre de cycles d'horloge par Multiplication addition. Nous pouvons consulter d'autres unités dans lesquelles la vitesse de calcul est exprimée dans le tableau en **Annexe 1.**
- Sa quantité de mémoire interne (DRAM/RAM/ROM/Flash…)
- Ses entrées /sorties (ports série, ports parallèles) et leurs vitesses respectives.
- Le choix d'un DSP pour une application particulière sera conditionné par le coût relatif du DSP par rapport au BOM (**B**ill **O**f **M**aterial : une liste quantitative des matériaux bruts) de l'application, et de la complexité des opérations à effectuer.

2.2 Le TMS320C6474

2.2.1 Introduction

La famille de processeurs la plus répandue actuellement est celle des DSP de Texas Instruments qui détient environ 70 % du marché, les 30 % restant étant partagé entre Motorola, Analog Devices, Lucent Technologies, Nec et Oki.

Les produits Texas instrument sont répartis en trois classes appelées plates formes contenant les différentes familles de DSP :

-TMS320C6000, formée des familles C62x, C67x et C64x;
-TMS320C5000, formée de la famille C54x et des C54xx;
-TMS320C2000, formée des familles C20x et C24x;

Le Tableau suivant résume les principales caractéristiques de ces trois classes :

Application		Type de DSP	Caractéristique
TMS30C6000 DSP hautes performances			
C64x	Application au traitement numérique de signal	32 bits virgule fixe	24000 MIPS -3,0 GHz
C62x	Applications exigeantes en vitesse: stations de base des réseaux de communications mobiles, équipement de radiodiffusion, réseaux informatiques	16 bits virgule fixe architecture VLIW	1200-2400 MIPS
C67x	Applications exigeantes en précision, dynamique et vitesse: Antennes adaptatives des stations de base, imagerie médicale, reconnaissance de parole, graphisme...	32 bits virgule flottante architecture VLIW	600MFLOPS-1GFLOPS
TMS320C5000 DSP optimisés en consommation			
C54x	Applications de télécommunications exigeantes en coût, consommation, vitesse: terminaux mobile, voix sur IP, alphapages...	16 bits virgule fixe	0.54 mWLMIPS 30-200 MIPS
TMS320C2000 DSP optimisé pour les applications de contrôle			
C20x	Applications de grand volume en téléphonie, électronique grand public, appareils photo numériques ou contrôleurs de disques durs...	16 bits virgule fixe	PLL, UART, timers, mémoire flash intégrée 20-40MIPS
C24x	Applications de contrôle moteur, automatisation, robotique, contrôle d'appareils électroménagers...Bon compromis prix/performance	16 bits virgule fixe	port série SCI, SPI et CAN, convertisseur Analogique/numérique 20MIPS

Tableau 2: Caractéristiques principales des familles de TI

La figure suivante représente l'évolution des produits DSP de la classe C6X de Texas Instrument et les futurs produits en développement.

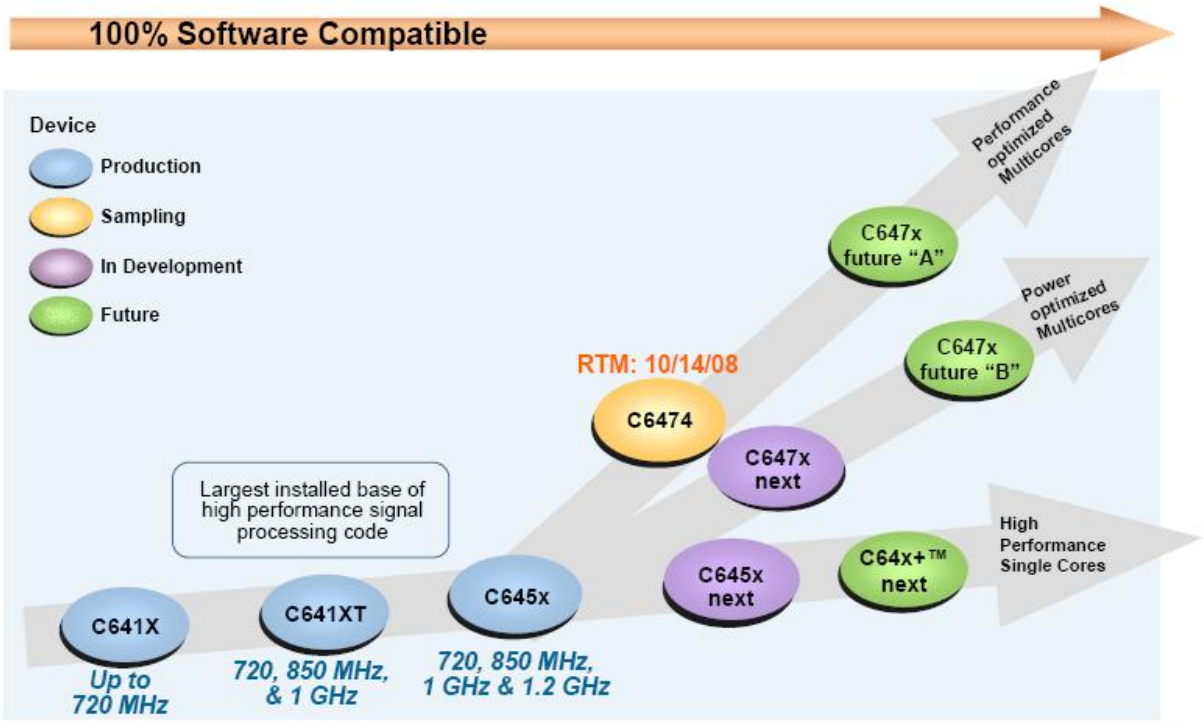


Figure 5: L'évolution des familles de la classe DSP TMSC6x selon les performances et les axes de recherche

À partir du C645x, les recherches sont lancées sur trois pistes principales :

- L'optimisation des performances des DSP Multi-cœurs.
- L'optimisation d'énergie consommée des DSP Multi-cœurs.
- Le développement des performances des DSP à cœur unique.

Le DSP sur lequel nous travaillons est le TMS320C6474, qui est le tout dernier DSP de Texas Instrument de type 32 bits et à virgule fixe.

La prochaine version du DSP sera disponible d'ici une année, elle prend en compte le flottant et la gestion de la mémoire MMU, et supporte tous les caractères du Risque.

La figure suivante représente la carte d'évaluation EVM6474 :

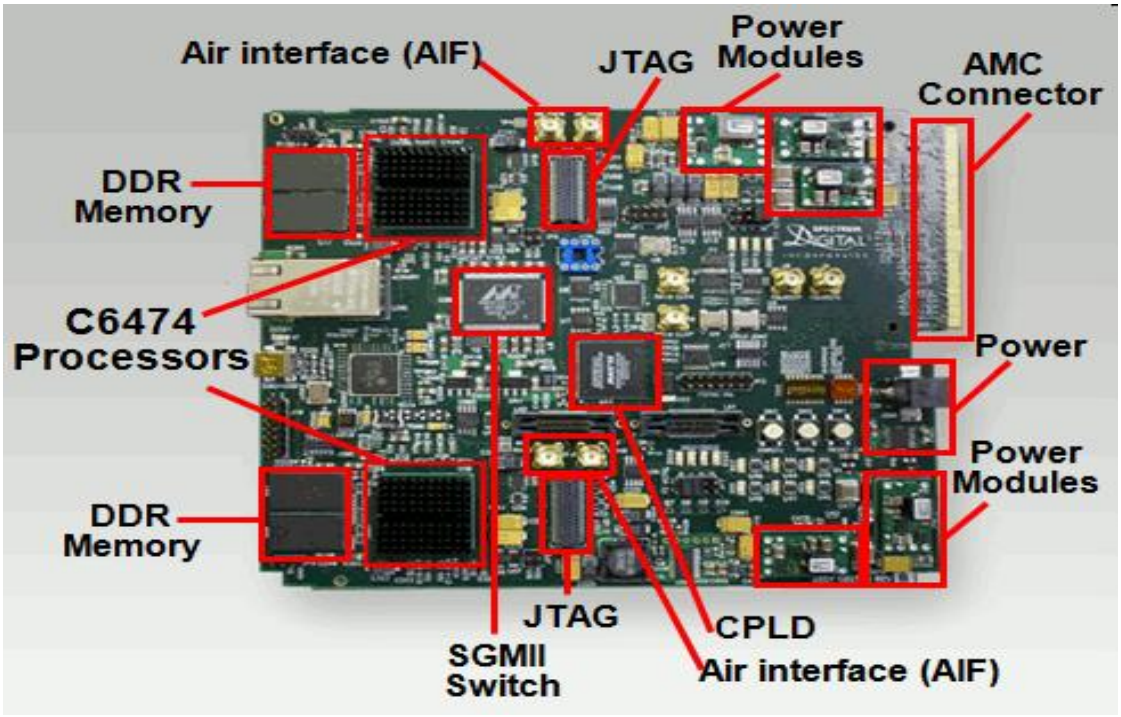


Figure 6: EVM6474

2.2.2 Caractéristiques

Le DSP TMS320C6474 de la sixième génération construite par Texas Instrument avec une technologie CMOS de 65 nm, est de la troisième génération de haute performance multi-cœurs qui est une architecture développée par Texas Instrument (TI)

- intégrant trois cœurs de C64x+ 1GHz sur une même puce, soit 3GHz de performances brutes le DSP peut faire 24000 millions instructions par seconde (24 instructions mips -par cycle).

Figure 7: Cœurs d'un GHz dans une seule architecture de C6474

- Caractérisé aussi par une consommation inférieure d'un tiers et un coût réduit de deux tiers par rapport aux solutions de traitement mettant en œuvre plusieurs DSP.

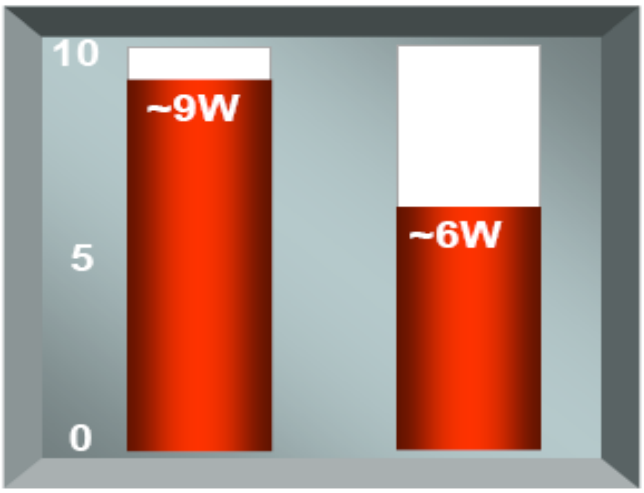


Figure 8: Consommation d'énergie du DSP TMS320C6474 par rapport au DSP TMS320C6455

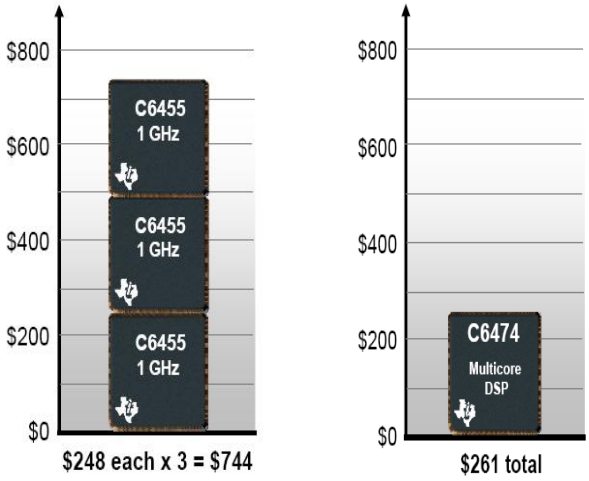


Figure 9: Le coût du C6474 par rapport au 3 coeurs du C6455

- Caractérisé par une mémoire interne divisée en deux caches L1 et L2, au contraire des anciens DSP qui ne contiennent qu'une seule Mémoire interne.

Suite à cet inventaire des caractéristiques intéressantes de DSP, une description de son architecture permettra de faire une idée générale sur le fonctionnement matériel, et sera une meilleure façon pour exploiter ses ressources.

2.2.3 Architecture

L'architecture du DSP TMS320C6474 est caractérisée par une diversité des composantes, permettant un traitement fluide des données, un adressage efficace, et des périphériques de communication supportant différents protocoles de communication (i.e. Serial RapidIO, Serial Link CPRI, 1 GigE). L'interface CPRI de type SERDES Sérialisation/Désérialisation est bien adaptée à la réception des signaux d'antenne à haut débit.

La figure suivante montre le schéma fonctionnel du cœur de DSP :

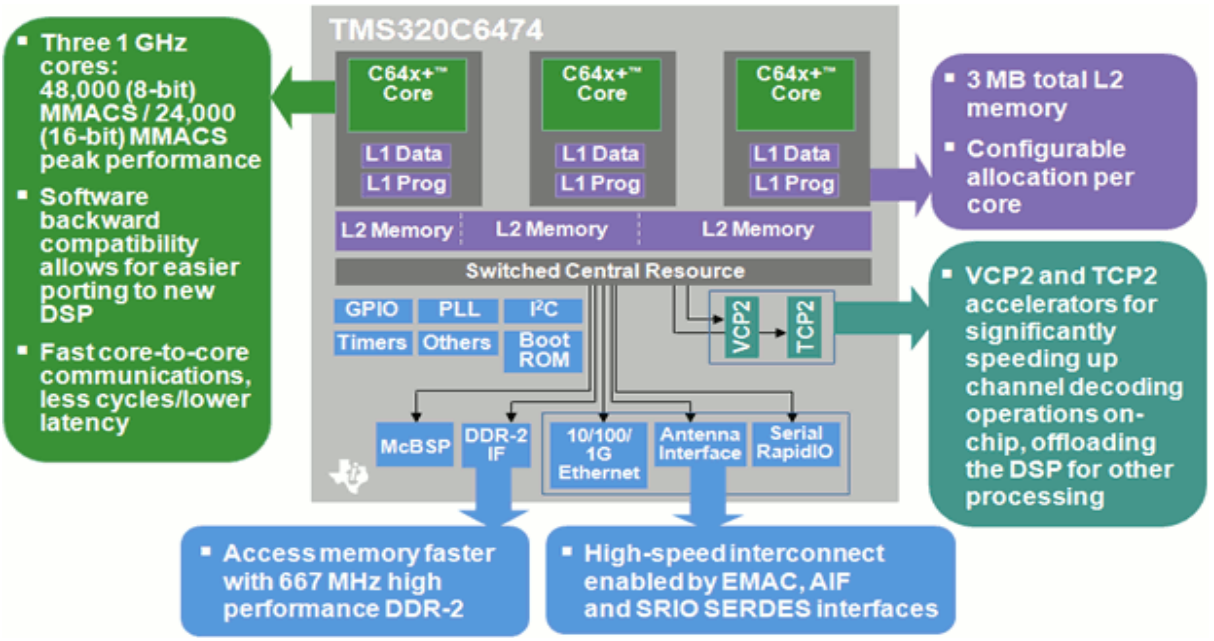


Figure 10: Schéma fonctionnel du cœur DSP [3]

Dans la partie suivante, on décrit les Blocs principaux de l'architecture interne de notre DSP, ainsi que les périphériques.

2.2.3.1 Architecture interne du DSP C64x+

La figure suivante représente le Bloc Diagramme de la famille TMS320C64x+. Les familles de la plate forme C6000 sont livrées avec la mémoire du programme qui, sur certains appareils, peut être utilisé comme un programme de cache. Les dispositifs ont également différentes tailles de la mémoire de données. Les Périphériques tels que le contrôleur d'accès direct à la mémoire (DMA), la logique power-down, et l'interface de la mémoire externe(EMIF) viennent habituellement avec le CPU, alors que d'autres périphériques tels que les ports série et les ports d'accueil ne sont qu'avec certains appareils **[4].**

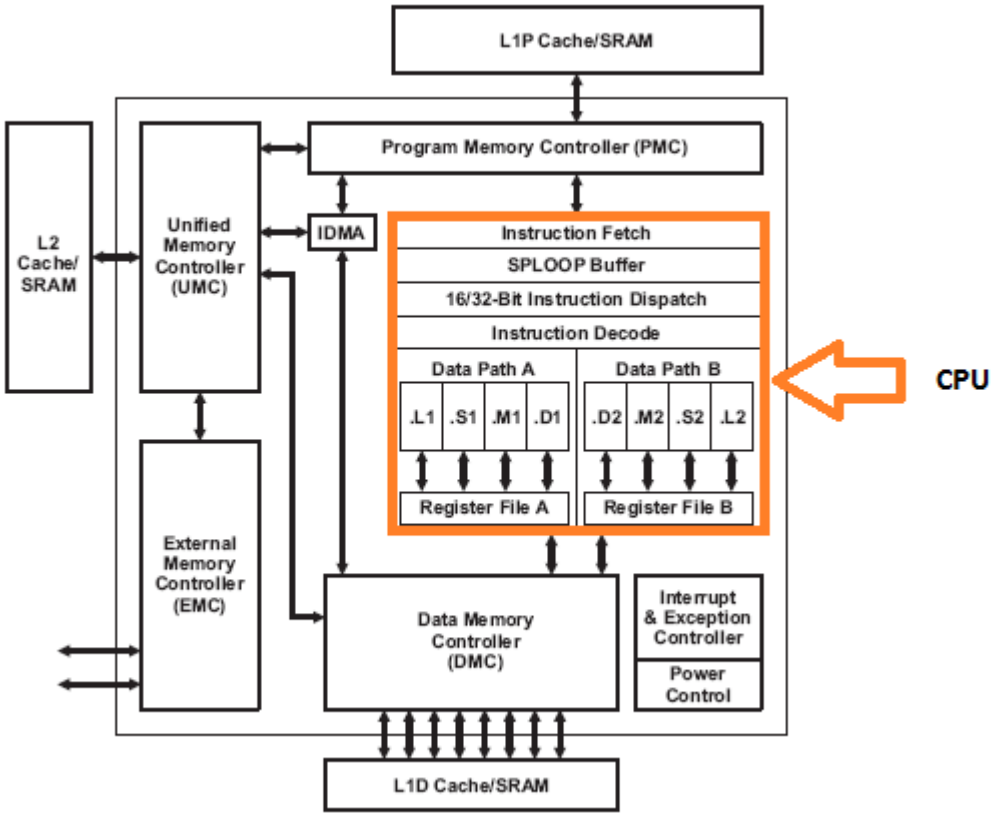


Figure 11: Diagramme en blocs de la famille TMS320C64x+ [4]

2.2.3.1.1 L'unité centrale de calcul (CPU)

L'unité centrale de calcul (CPU) se compose de huit unités fonctionnelles, deux registres, et de deux chemins de donnée. Les deux registres à usage général (A et B) 32-bit contiennent chacun 32 registres pour un total de 64 registres. L'objectif général des registres c'est qu'ils sont utilisés pour les données et peuvent être utilisés pour l'adressage des pointeurs de données, les types de données pris en charge sont de 8-bit, 32-bit, 40-bit, et 64-bit [**5].**

Les huit unités fonctionnelles (**.M1**, **.L1**, **.D1**, **.S1**, **.M2**, **.L2**, **.D2**, et **.S2**) et leurs rôles regroupés dans le tableau suivant :

Les Unités fonctionnelles	Leurs rôles
.M (multiply)	Effectue toutes les opérations de multiplication en un seul cycle d'horloge : - une multiplication 32 x 32 bit. - deux multiplications 16 x16 bit. - deux multiplications 16 x32 bit, 4 multiplications 8 x8 bit.
.L (arithmetic logical)	Effectue en parallèle les opérations +/- sur 32 bit ou 2 fois 16 bit
.S	Parallélise les instructions 8 bits/16bits et duales 16 bits. Il réalise aussi en parallèle des opérations MIN et MAX.
.D	Charge les données de la mémoire vers les registres et stocke les résultats des registres dans la mémoire.

Tableau 3: Les unités fonctionnelles du CPU et leurs rôles.

Beaucoup d'algorithmes de traitement de signal comme la FFT et le FIR complexe exigent la multiplication complexe, l'instruction (CMPY) prend quatre entrées de 16 bits, et produit une sortie réelle et imaginaire en 32 bits en un seul cycle. Il y a également des multiplications complexes avec une sortie sur 32 bits contenant 16 bits de réel et 16 bits d'imaginaire. Les instructions de multiplications 32 x 32 bit fournissent une précision efficace pour l'audio et d'autres algorithmes à haute précision pour une variété des données 32 bits signés ou non signés.

2.2.3.1.2 Mémoire interne

2.2.3.1.2.1 Le Cache L1

La Mémoire interne niveau 1 (figure 10 et 11) est organisée en deux espaces, l'un pour les données et l'autre pour les programmes, elle dispose de deux ports internes 64-bit pour accéder aux données internes de mémoire, et un seul port interne pour accéder à la mémoire interne programme **[5].**

2.2.3.1.2.2 Le Cache L2

Chaque cœur a une mémoire interne niveau 2, L2 RAM (figure 10 et 11). La capacité totale est de 3MB qui peut être configurée soit en 1MB pour chaque cœur ou 1.5MB/1MB/0.5MB respectivement pour chaque cœur, La SRAM L2 commence à la même adresse, quelle que soit la taille du cache de configuration **[5].**

2.2.3.1.2.3 La ROM L3

La ROM L3 (figure 10) est de taille 64KB, Le contenu de la ROM est divisé en deux partitions, La première est la ROM de démarrage (ou Boot) dont le but principal est contenir des logiciels au démarrage de périphérique, la seconde partition est la partie sécurisée du ROM, qui dispose d'un cœur nécessaire pour soutenir des dispositifs de sécurité sur le périphérique **[5].**

2.2.3.2 Périphérique du DSP C64x+

Le DSP TMS320C6474 intègre plusieurs périphériques annexes au microprocesseur C64x+. Ces périphériques sont en général soit dédiés à la communication, soit représentants des unités fonctionnelles témoignant de la particularité des exigences des applications de traitement de signal. **L'annexe 2** propose les définitions des différents périphériques qui décrivent le fonctionnement de chacun en apportant des détails sur ses caractéristiques et ses configurations.

2.2.4 Conclusion

Le C6474 pourrait offrir des solutions rentables de haute performance et posséder aussi la flexibilité opérationnelle de la grande vitesse ainsi qu'une architecture plus développée avec :

- Une vitesse de calcul de 24 MMULAC/s.
- Une architecture Multi-cœur de 3GHz de performance brute avec unes répartition du mémoire interne en deux caches L1 et L2 de 3Mo par rapport à la version ultérieure contenant un seul mémoire interne.

Les performances de ce DSP justifient l'intérêt qu'il suscite dans plusieurs entreprises, dont notamment Thales, qui mise sur cette nouvelle carte pour remplir ces objectifs. Avant de présenter les résultats des tests de performances que nous avons fait subir au Linux embarqué sur cette carte par rapport aux autres architectures, une description de système de supervision radar sera détaillée par la suite.

2.3 Le système de supervision Radar

2.3.1 Fonctionnalités de la Supervision Radar

Le Superviseur Radar Central (figure 12) supporte les principales fonctionnalités suivantes:

- ✓ Les fonctionnalités dites de Supervision (SUP)

- S'interfacer avec les supervisions locales de chaque abonnés SNS (**Suplink Network Subscriber**) pour gérer le « ON/OFF » des BB (building bloc) ou LRU (**line replaceable unit**),
- Accéder aux informations locales de BIT (built-in test) pour chaque SNS du réseau Supervision (**SUPLINK**),
- Recevoir les informations de BIT (filtrées ou non) de chaque BB (pour chaque LRU ou chaque fonction logicielle),
- Faire la corrélation de panne au niveau système, et fournir l'état du radar,
- Récupérer la configuration des SNS (configuration appliquée; configuration matérielle et logicielle),
- Récupérer les compteurs horaires des LRU des SNS si la fonction existe.
- Fournir l'historique des pannes,

✓ Les fonctionnalités dites de Services (Global Software Service) (GSS)

- Debug,
- Monitoring (trace, perfmeter, …),
- Recording,
- Mise à jour des logiciels (FPGA, …) de chaque carte des BB.

✓ Communiquer avec la LCC en version outillage afin d'avoir une IHM pour :

- Les actions opérateurs : ON/OFF….,
- Avoir une visualisation hiérarchique des états du radar,
- Visualiser les informations de BIT (BIT),
- Visualiser la configuration appliquée,
- Gérer les fonctionnalités de debug, d'enregistrement et de mise à jour des logiciels.

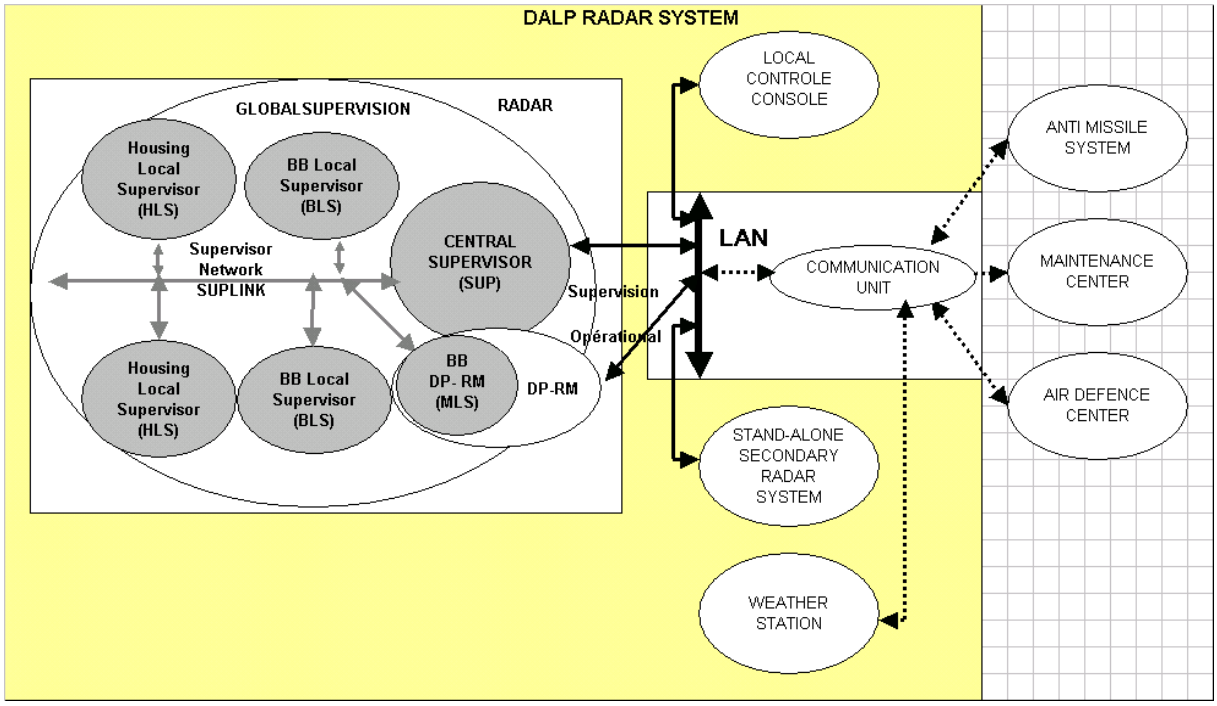


Figure 12: schéma de la supervision radar [6]

2.3.2 Système de Supervision

La supervision centrale est une architecture ouverte basée sur la politique Building Block. Dans notre cas le fonctionnement du superviseur est basé sur une architecture PC Linux qui est totalement compatible avec l'interface SUPLINK. Le schéma ci-dessous illustre un synoptique global de ce système :

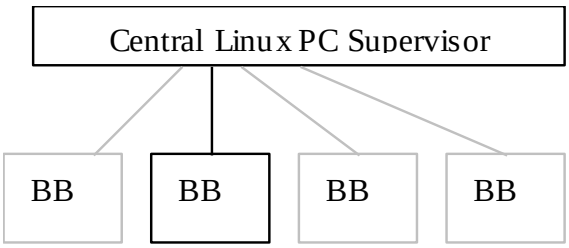


Figure 13: Architecture d'une centrale de supervision [7]

En effet, dans le système radar, un centre de PC sous Linux gère tous les radars: il peut être connecté à chaque BB (building bloc), à travers un hôte CCC (carte de commande et de contrôle). Tous les BB (building bloc) du centre PC superviseur constituent le réseau Ethernet de contrôle. Dans ce dernier on trouve plusieurs connexions de socket à chaque BB :

Pour **suplink** et pour **servlink** (HTTP, FTP, SNMP ou à l'exploitation et des clients spécifiques débogueur).

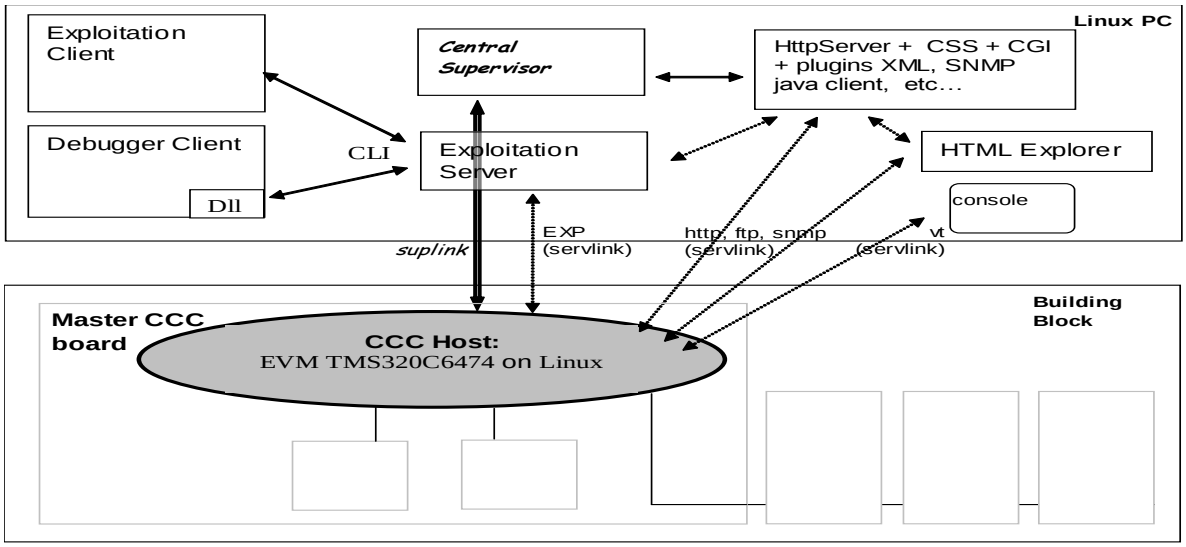


Figure 14: le superviseur central en lien avec CCC Host [7]

Le PC central superviseur a un lien direct avec toutes les cartes de commande et de contrôle (CCC) considérés comme des hôtes. Il la commande à travers une interface homme machine (IHM) sous html.

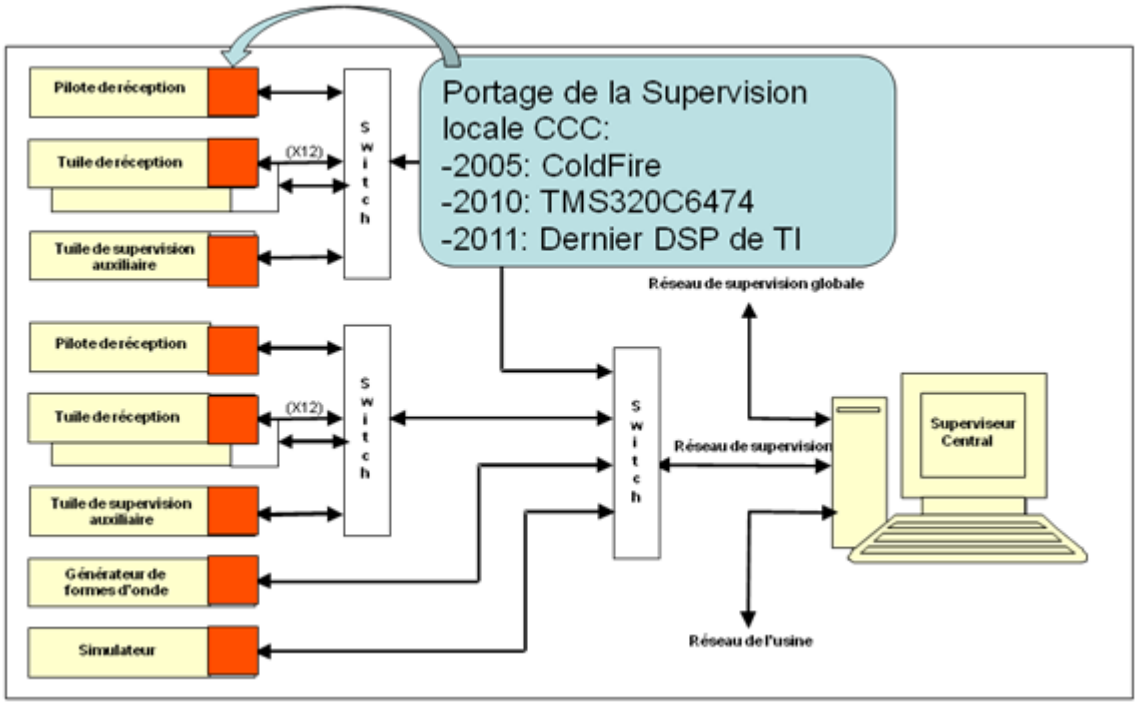


Figure 15: Architecture matérielle de la supervision Radar [6]

Un serveur http centrale sur PC Linux assumera la gestion des pages, la présentation de certaines données de téléchargement à partir du module carte de commande et de control (CCC) et les autres abonnés du réseau ou les fonctionnalités du BB (building bloc) du groupe.

La fonctionnalité «passerelle» du serveur d'exploitation entre le réseau de développement et le réseau cible pourrait être assumée par ce PC central, configuré en tant que passerelle HTTP. L'exploitation traditionnelle sera remplacée par l'exploitation html, utilisant des serveurs http.

Pour le débogage à travers Ethernet, le développement IDE (Visual DSP + +, Eclipse, Code-Warrior) sera connecté à travers des « Plugin » au processus tempo, ou à des superviseurs et hôtes Xpack. Un client de réseau SNMP, intégré dans le code HTML (applet Java) pourrait être utilisé à l'avenir pour accéder à la CCC SNMP MIB (l'information BB (building bloc) géré par le protocole SNMP sur les CCC d'accueil)

2.3.3 Presentation des BB (building bloc)

Dans un « Building Bloc » une Carte de commande et de contrôle (CCC) a des liens avec les DSP, Xpacks ou d'autres périphériques.

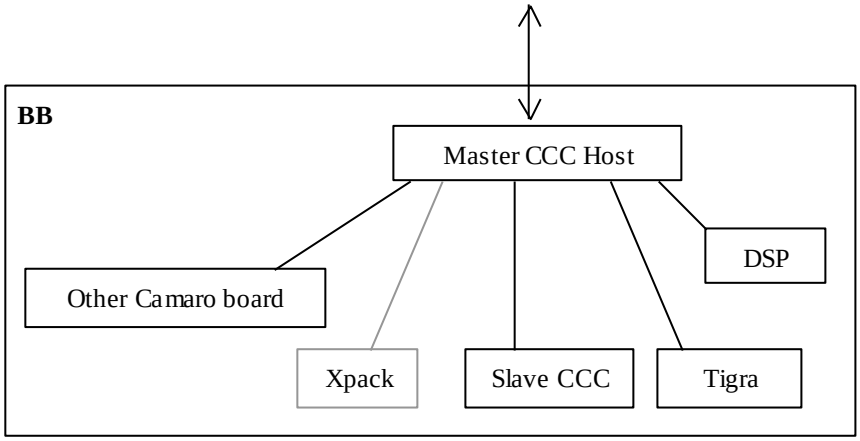


Figure 16: présentation des BB [7]

Le système superviseur attribut des adresses aux différents BB (building Bloc) pour permettre des transferts sur la ligne SUPLINK qui s'effectue à travers la carte de commande et de contrôle. Le but de l'adressage est :

- ✓ **D'identifier les BB (building bloc) en emplacement et la fonction.**
  - ➢ Pour donner une information externe sur le BB de l'identifier.
  - ➢ Pour configurer les fonctions par leur adresse logique dans un ensemble de billes qui ont la même fonction et le même logiciel.
  - ➢ D'identifier l'expéditeur et le récepteur d'un message sur le terrain Source ID, ID destination dans le bloc d'en-tête des messages applicatifs.

- ✓ **Pour réaliser les connexions entre les BB (building bloc)**

Exemple: les connexions TCP / IP pour les communications sur le réseau Ethernet

Sur Le système Linux, la carte de commande et de contrôle est implémentée par plusieurs applications:

- processus Tempo
- Processus Superviseur Host
- Processus Xpack Host
- Processus de Ftp, Http Server Process, CGI ...

Un système de fichiers, sur les disques de PC ou sur la mémoire flash de la cible, est utilisée pour le stockage de toutes les informations (binaire, CGI, pages HTML, application binaire, de vie des cartes, carte d'identité, copie de tables de correction des Xpack, les paramètres de sécurité, ptempo paramètres, les autres paramètres).

2.4 L'atelier Logiciel

Dans cette partie, est décrit l'ensemble de l'atelier logiciel permettant de développer des applications pour la carte EVM6474. L'atelier logiciel comprend deux ensembles. Le premier étant celui fournit avec la carte, le deuxième est l'atelier installé pour satisfaire les besoins du présent projet.

2.4.1 L'existant

Avec les cartes EVM6474, Texas Instruments fournit l'atelier **Code Composer Studio (CCS).** Cet atelier, destiné à être installé sur un ordinateur équipé de Windows XP, comprend les pilotes nécessaires à la carte. L'ordinateur étant relié à la carte par port USB, et d'un port JTAG du côté de la carte. CCS permet de rédiger le code en C ou bien en C++. Il permet aussi de compiler et d'optimiser les programmes rédigés. L'exécution des exécutables générés est possible dans un mode simulé ou bien directement sur la carte. Dans le dernier mode, le programme est chargé sur la carte via **JTAG**. **CCS** permet par la suite de déboguer un programme en exécution sur la carte. Plus de détails sont disponibles en **Annexe 3.**

2.4.2 Les compléments d'atelier

Les programmes de supervision tournent sur un Linux. Comme c'est le cas pour le processeur **ColdFire**, on devrait pouvoir exécuter un linux sur le TMS320C6474. D'autre part, la carte ne dispose pas d'espace suffisant pour contenir le système de fichiers de Linux, auquel s'ajouteront les différents modules de supervision. De plus, le **poste Windows**, sur lequel est installé **CCS**, ne présente pas une solution à ces contraintes. D'où l'introduction de l'environnement de Cross Compilation

2.4.2.1 Environnement de Cross Compilation

La solution technique proposée par Texas Instruments, consiste à stoker le système de fichiers de Linux sur un ordinateur **Linux Fedora**. Ce poste permettra de Configurer et de compiler un noyau Linux, et générera donc son système de fichier. Cet environnement de Cross Compilation permet donc de compiler des programmes pour le poste lui-même, ainsi que pour la carte. Ceci est possible car l'environnement est constitué de différents outils, dont en particulier une chaine de compilation de Texas Instruments pour le TMS320c6474. Il reste donc à définir la façon avec laquelle le noyau sera lancé sur la carte et comment il accèdera à son système de fichiers sur le **poste Fedora**.

2.4.2.2 Configuration, compilation et exécution du noyau Linux

Le noyau Linux sera exécuté sur l'EVM6474. Son système de fichiers, requis pour son exécution se trouvera sur le **PC Fedora**. L'accès du noyau à ce système de fichiers sera assuré sur le **Pc Fedora** par un serveur **NFS**. Le lien entre la carte et le PC est assuré par Ethernet. Un Switch est placé entre ces deux derniers. Sur le **PC Fedora** on configure l'adresse IP du noyau Linux sur la carte, le chemin d'accès au système de fichiers, et l'adresse IP du **PC Fedora** auquel il se connectera. En ce qui concerne le PC, on configure **NFS** et l'adresse IP de la carte réseau. L'exécutable du noyau linux configuré sera chargé sur la carte par **JTAG**, en utilisant **CCS** sur un **pc Windows**.

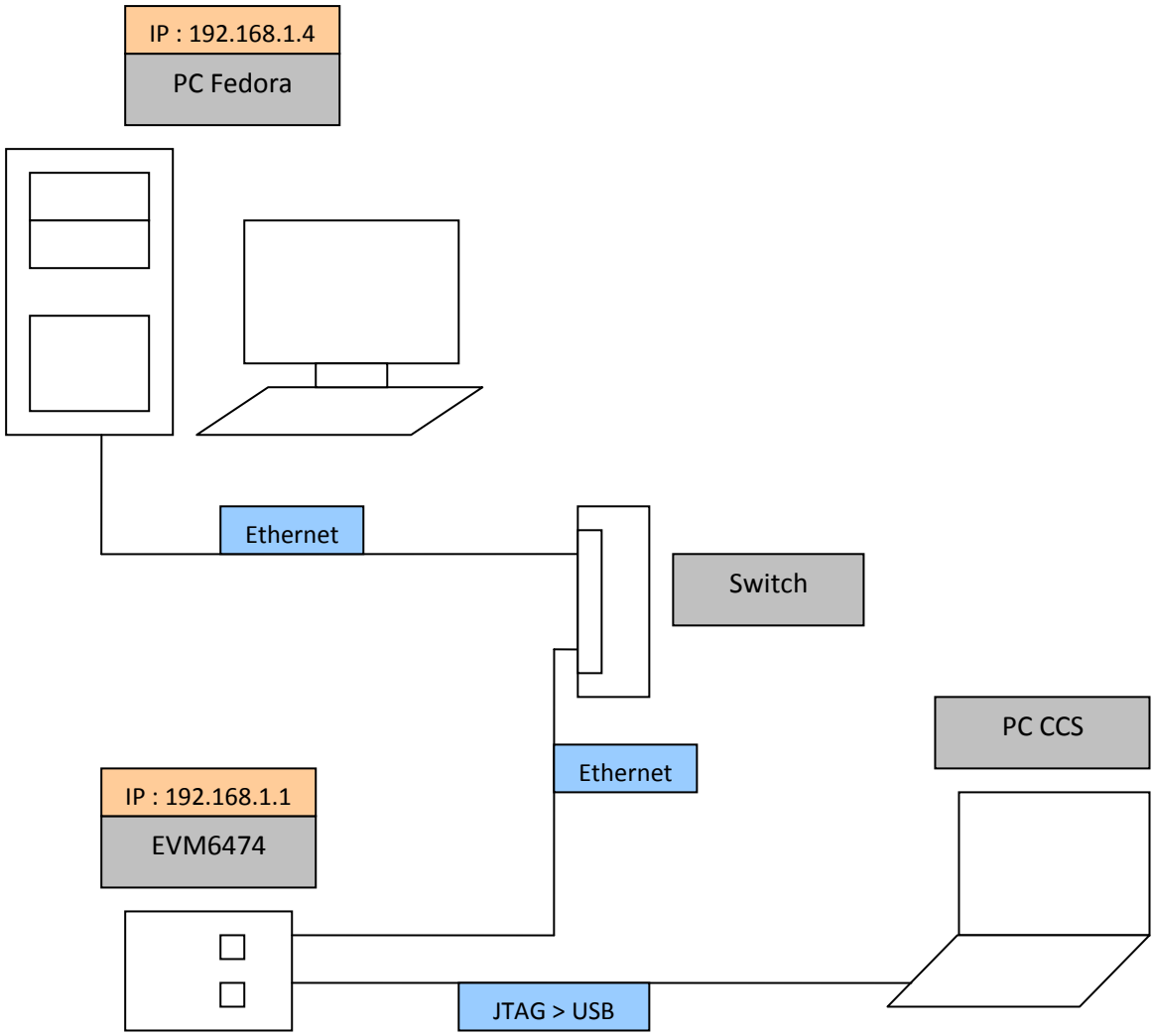


Figure 17: Dispositif de lancement de Linux sur EVM6474

Plus de détail sur la procédure sont disponibles en **Annexe 4.**

L'étape suivante, consiste à acquérir le savoir faire technique nécessaire pour pouvoir compiler des programmes pour la carte, en exploitant l'environnement de Cross Compilation, et pour les exécuter par-dessus de Linux déjà en exécution sur la carte.

2.4.2.3 Cross compilation et exécution de processus

Pour compiler des programmes pour la carte, il faut utiliser le compilateur de Texas Instruments au lieu de celui par défaut sur le poste Fedora. Donc dans la commande de compilation du programme, il faut remplacer GCC par le chemin complet de l'exécutable du compilateur. Pour générer des exécutables destinés à être exécutés par-dessus de Linux, il faut compiler en utilisant l'option **–Wl,-ar**. Ensuite, le binaire généré est placé dans le système de fichiers du noyau Linux. Finalement, à partir de la console Linux, il suffit de lancer le programme. Plus de détails sur cette procédure, appliquée à deux programmes de la supervision, est disponible en **annexe 5**.

Conclusion :

Maintenant qu'on maitrise les procédés nécessaires pour faire tourner des applications par-dessus de Linux embarqué sur la carte EVM6474, il va falloir déboguer ces applications. La mise en place d'un système de débogage adapté à notre dispositif est réalisée par l'autre groupe à Limours.

Chapitre 3 : Mesure des performances du noyau linux sur le DSP tms320c6474

3 Mesure des performances du noyau linux sur le DSP TMS320C6474

3.1 Objectifs

A ce niveau, on dispose d'un Linux qui tourne sur la carte EVM6474. La mesure des performances de ce dernier, permettra de l'évaluer, et de le valider. Les résultats des tests de performances permettraient de déceler de probables défauts d'embarquement de ce dernier par l'entreprise qui l'a fournit. De plus, sur l'actuel système de supervision, un linux tourne sur le processeur **ColdFire**. En faisant des tests de performances à Linux sur les deux plateformes, on va pouvoir apprécier le gain de performance que permet le TMS320C6474. D'un autre côté, Les radars actuels embarquent des processeurs **Intel Core 2 Duo**. Par conséquent, ces tests de performances exécutés sur les trois plateformes permettent de positionner le **TMS320C6474** (présentés parfois par Faraday comme le cas dans le Tableau 4) par rapport aux deux autres. Nous allons plus loin dans le positionnement du **TMS320C6474** en le comparant aussi au **PowerQuicc** (de la société **Freescale**). D'autre part, nous disposons des résultats de tests de l'entreprise qui a embarqué Linux sur la carte dont nous disposons, appliqués à Linux embarqué sur une plateforme intégrant le même microprocesseur que celui sur notre carte. On pourra donc s'assurer que nous obtenons les mêmes résultats.

3.2 Les architectures cibles

Hardware		ColdFire	PowerQUICC	Faraday	Core2Duo
Processor	Core Name	V4e	e500	C64x+	E6550
	Number of Cores Available	1	1	3	2
	Frequency	200 MHz	1GHz	1 GHz	2,33 GHz
	Lithography			65 nm	65 nm
	TDP	1,5 W per chip	7 W per chip	2 W per core	65 W per chip
	FPU	Yes	Yes	No	Yes
	MMU	Yes	Yes	No	Yes
External Memory	Type	DDR SDRAM	DDR SDRAM	DDR2 SDRAM	DDR2 SDRAM
	Clock Rate	66-133 MHz		667 MHz	667 MHz
	Organisation			Mono Bank 1x32 bits	Dual Bank 2x64 bits
Internal Memory	L1	2x32 KB	2x32 KB	2x32 KB	64 KB
	L2		256KB per chip	1 MB per core	4 MB per chip
Software platform	Operating System	Linux		Linux	Fedora 10
	Run Mode			on one core	on the entire chip
	Kernel Version	2.6.13		2.6.13	2.6.27
	Dev Environment	Cross Environment		Cross Environment	Red Hat 4.3.2-7
	GCC Version	4.2.3		3.2.2	4.3.2

Tableau 4: Descriptif des architectures cibles

3.3 Les outils de benchmarks

Les modules testés sont : Les appels système, la puissance de calcul, la gestion de la mémoire et la communication via Ethernet. Les logiciels de tests effectués sont les suivants :

- LMBench
- NBench
- RAMspeed
- CacheBench
- TTCP
- IPERF
- Dhrystone

Parmi ces logiciels de tests, **LMBench**, **TTCP**, **IPERF** ainsi que **Dhrystone** sont fournit par **Virtual Logix**. Par contre, les autres sont des logiciels Open Source téléchargés sur Internet.

Pour intégrer ceux fournit par **Virtual Logix** au système de fichiers du noyau, il a fallut juste les sélectionner à partir du fichier de configuration des modules et de relancer la génération du système de fichiers, ils y sont installé automatiquement. Par contre, pour installer les autres logiciels de test de performance, il a fallu modifier leurs fichiers de compilation. Les modifications apportées consistent principalement à reconfigurer le compilateur utilisé, et de spécifier celui dont nous disposons dans l'environnement de Cross Compilation. Ainsi, les binaires générés pouvaient êtres exécutés par-dessus du Linux embarqué. En ce qui concerne l'exécution des tests sur l'ordinateur équipé d'un processeur **Intel Core 2 Duo**, il n'y avait pas de procédure spécifique. En ce qui concerne le **ColdFire**, les tests n'ont pas encore été effectués car on ne dispose pas encore du support nécessaire pour le faire, et l'action n'est pas encore organisée.

3.4 Dhrystone

3.4.1 Description

Dhrystone est un logiciel de test de performances standard écrit en C ANSI. Ce logiciel de test mono-tâche est exécuté sur Linux. La version utilisée pour les tests est la **version 2.1 [8].** Ce test a été compilé pour Faraday en utilisant l'environnement de Cross Compilation de **VLX** (la société **Virtual Logix**). Ce logiciel de test n'a pas été rédigé de façon à présenter particulièrement des optimisations pour **Faraday**. Il a été compilé avec l'option **–O5** pour le compilateur GCC. Les résultats de ce test sont représentés en DMIPS, pour **Dhrystone** MIPS, selon la référence **VAX** (un processeur de référence) 11/780 (pour avoir la valeur en DMIPS, on divise par 1757 la valeur trouvé en MIPS : Million d'instructions par seconde).

3.4.2 Résultats et analyse

Processor	ColdFire (1 core)	PowerQUICC (1 core)	TCI6488 (3 cores) Test for one core	Faraday (3 cores) Test for one core			Core2Duo (2 cores)
MHz	200	1000	1000	1000	1200	1400	2327
W per Core	1,5	7	2	2	2	2	32,5
DMIPS per Core	308	2312	711	671	805	939	5735
DMIPS/W	205	330	346	335	402	469	176

Tableau 5: Résultats de Dhrystone

On remarque d'abords que la performance que nous avons mesurée pour **Faraday** est proche de celle publiée par **VLX**. Ce qui valide nos résultats. Sachant que la carte Faraday dont nous disposons tourne a une fréquence de 1000 MHz. Comme la fréquence de fonctionnement progresse toujours, afin d'avoir une idée sur l'évolution des résultats de **Dhrystone** sur de probables prochaines architectures, les résultats pour Faraday tournant à 1000 MHz ont été linéairement extrapolés pour donner une approximation des résultats correspondant aux fréquences 1200 MHz et 1400 MHz. En ce qui concerne les résultats pour le **ColdFire [9]** ainsi que le **PowerQUICC [10],** ils ont été publiés sur le site officiel de **Freescale** et ils sont repris dans ce tableau. Par contre, les résultats obtenus pour **Dhrystone** sur l**e Core 2 Duo** ne sont pas cohérents, approximativement 50% supérieurs (**3907 VAX MIPS**), avec les résultats publiés par des tierces parties sur internet [**11**]. Notons que pour Faraday 1400 MHz, les résultats de **Dhrystone** seraient à peu près 3 fois supérieurs aux résultats du **ColdFire**. On remarque aussi que les résultats pour le **Core 2 Duo** et le **PowerQUICC** sont supérieurs au **Faraday**. Ceci peut être expliqué par le fait que le test de **Dhrystone** est un test généraliste et est donc destiné à des architectures généralistes. **Dhrystone** manipule par exemple des chaines de caractères, alors que le Faraday n'est pas, bien évidemment, conçu pour faire des traitements sur des chaines de caractères. Ceci expliquerait donc, entre autres, les écarts de performance en question. Il est à noter aussi que le test a été compilé pour le **Faraday** en utilisant le compilateur croisé de **VLX**, dont on doit s'assurer de l'efficacité. Ce qui pourrait bien interférer dans les résultats de **Dhrystone** sur le **Faraday**.

En ce qui concerne la performance des cartes du point de vue énergétique, nous avons calculé le rapport de la performance de **Dhrystone** en **DMIPS**, par la dissipation thermique en Watt. Il s'en ressort que **Faraday** donne le meilleur rapport performance/énergie dissipée et donc la meilleure efficacité énergétique. Le **Core 2 Duo** est le moins efficace, ce qui est dû à la stratégie qu'Intel à mise en œuvre pour faire évoluer ses processeurs, qui constitue en l'augmentation de la fréquence sans assez optimiser les circuits, ce qui amène à des pertes énergétiques et par conséquent une réduction de l'efficacité énergétique du circuit.

3.5 TTCP/IPERF

3.5.1 Description

Afin de mesurer la performance de l'Ethernet, on va combiner **TTCP** [**12**] et **IPERF** [**13**]. En fait, les tests sont effectués dans les deux modes: TCP et UDP. Pour chaque mode, deux tests sont effectués : l'envoi (**send**) et la réception (**receive**). Tous les tests sont effectués en utilisant **TTCP**, à l'exception du **UDP receive**, qui est effectué en utilisant **IPERF**. Comme le module Ethernet est l'objectif de ce test, et que par conséquent on mesure l'envoi et la réception de données, il est nécessaire d'avoir une deuxième machine pour échanger les données de tests avec le module Ethernet objet des tests. Ainsi, disposant d'un client et d'un serveur de l'outil de test, le client est exécuté sur la machine cible tandis que le serveur est exécuté sur un PC. Le PC utilisé est le mêmes pour toutes les architectures cibles. Comme un lien Ethernet doit être établi entre la machine cible et le PC, un Switch Ethernet 10/100 est

utilisé pour lier les deux. Pour mesurer la vitesse d'échange de données, les tests sont exécutés sur des durées suffisamment longues pour produire des résultats pertinents. De plus, différentes tailles de paquets ont été testées. Notons que les résultats sont en Mbit/s.

3.5.2 Résultats et analyse

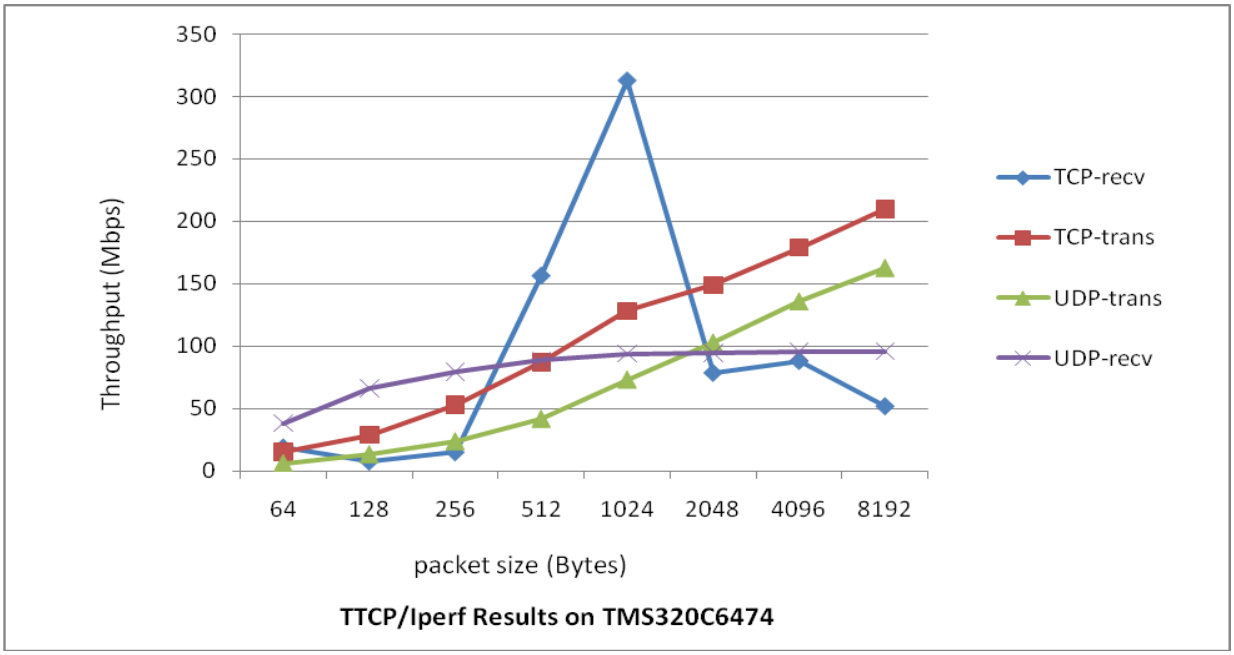


Figure 18: résultats de test de TTCP/IPERF pour TMS320C6474.

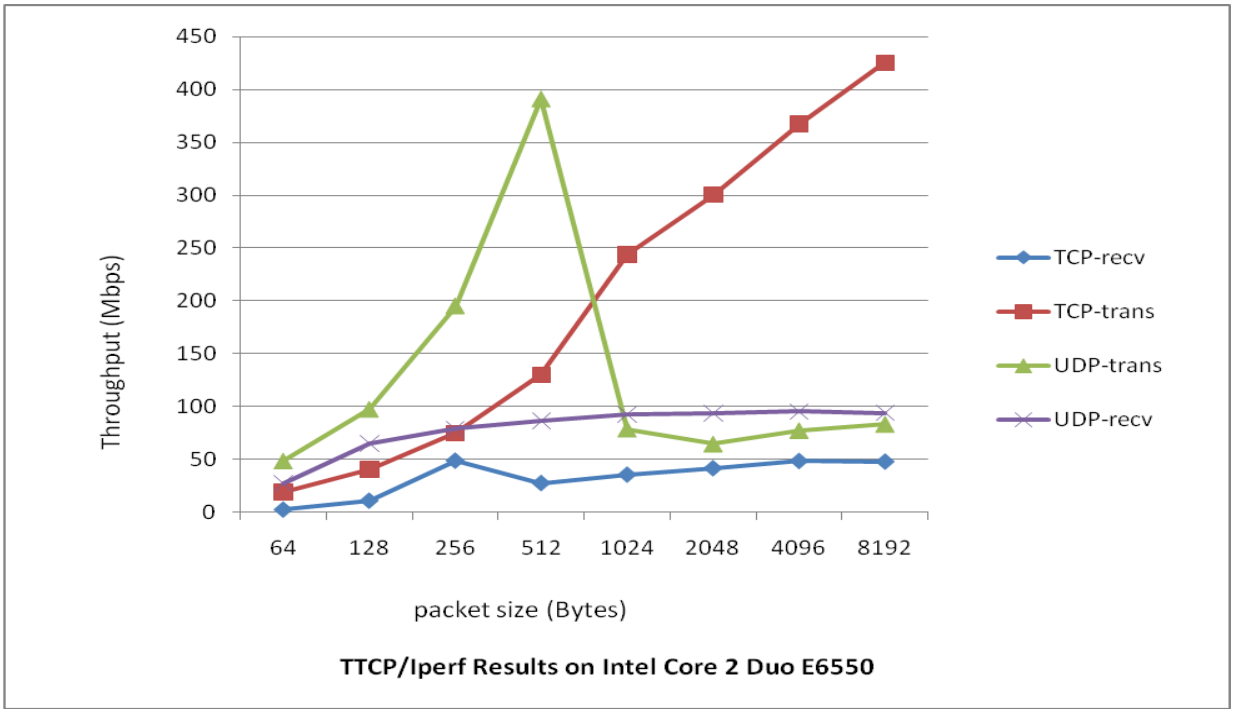


Figure 19: résultats de test de TTCP/IPERF pour PC dual cœur.

On remarque les pics (on peut dire que ces valeurs sont plus probablement aberrantes) de la bande passante en réception TCP sur **Faraday** et en transmission UDP sur le **Core 2 Duo**. Ainsi, si on se permet de les ignorer, on retrouve la même allure que pour les autres: augmentation et puis stabilisation sur une valeur proche du seuil de 100 Mbit/s, en décroissant dans le cas de la réception en mode TCP. Le débit en réception UDP est borné par ce même seuil. Par contre, la transmission en mode TCP pour les deux architectures croît jusqu'à dépasser légèrement 150 Mbps sur le **Faraday** et 400 Mbit/s sur le **core 2 duo**. La réception TCP sur le **Core 2 Duo** croît sans dépasser 50 Mbps.

La figure suivante présente les tests de **Virtual Logix** pour le TCI6488 avec un Switch Giga Ethernet :

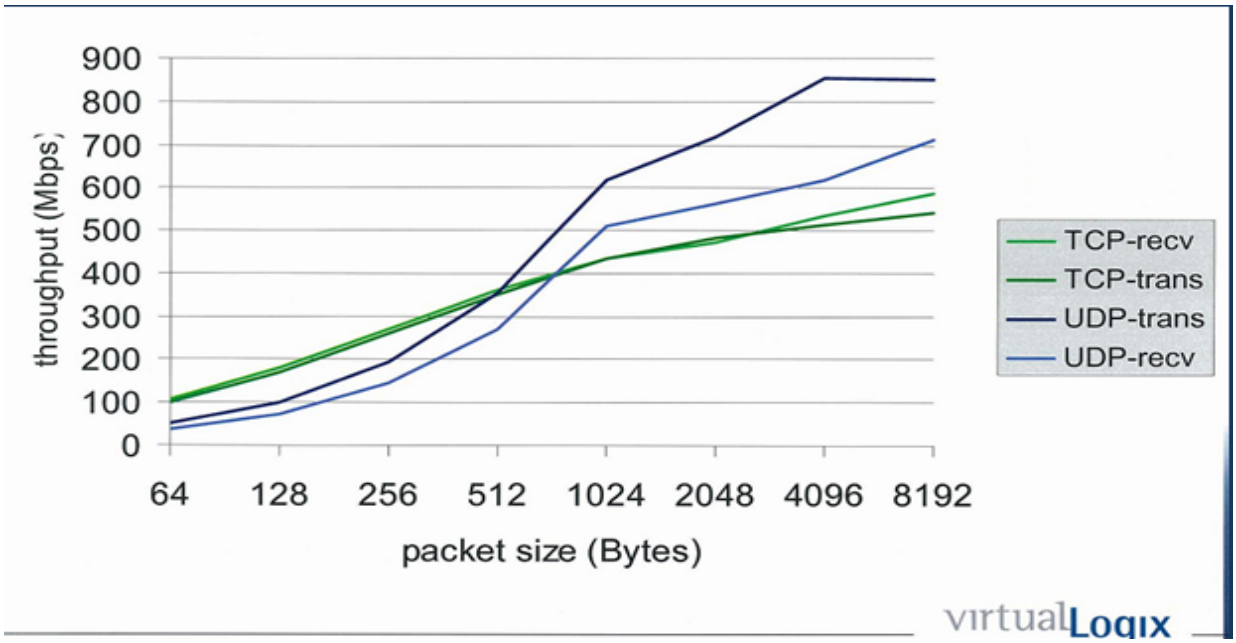


Figure 20: résultats de test de TTCP/IPERF pour le TCI6488

Le débit croît en fonction de la taille de paquets et se stabilise autour de 900 Mbit/s. ce qu'on doit avoir normalement pour le TMS320C6474 avec un Switch Giga Ethernet.

Les résultats d'Iperf tous seuls sont présentés dans les figures suivantes :

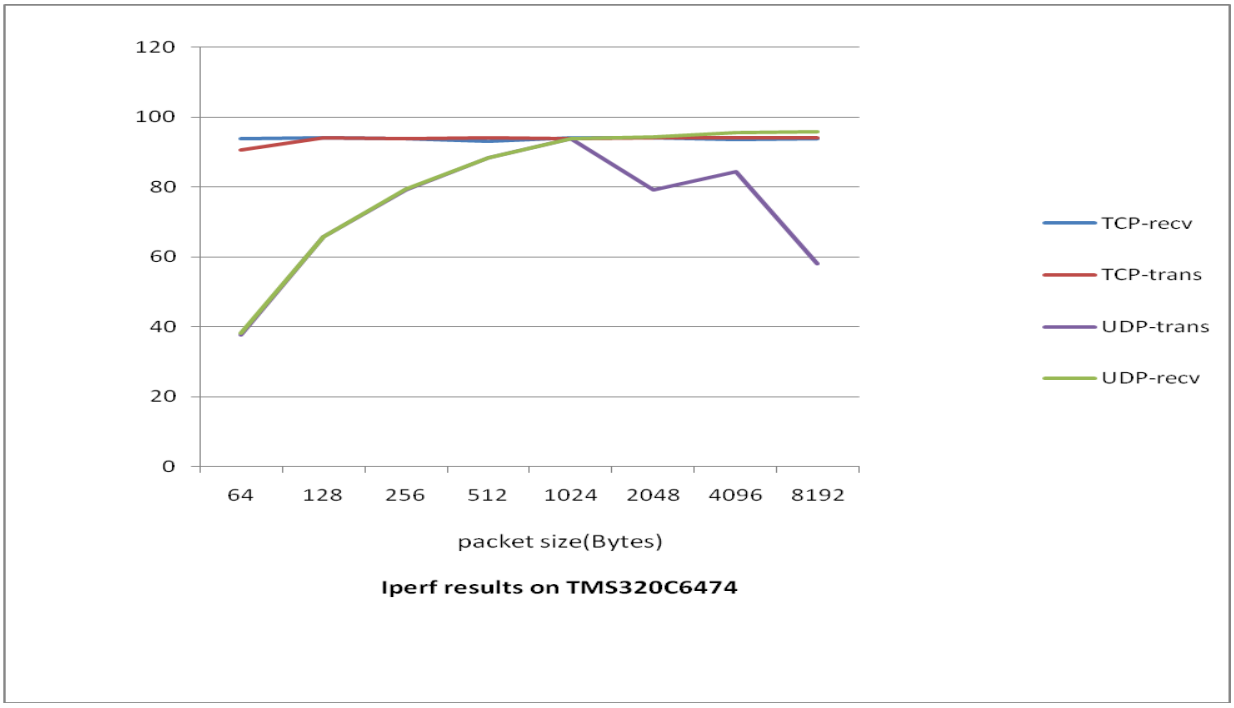


Figure 21: résultats de test d'IPERF pour le TMS320C6474

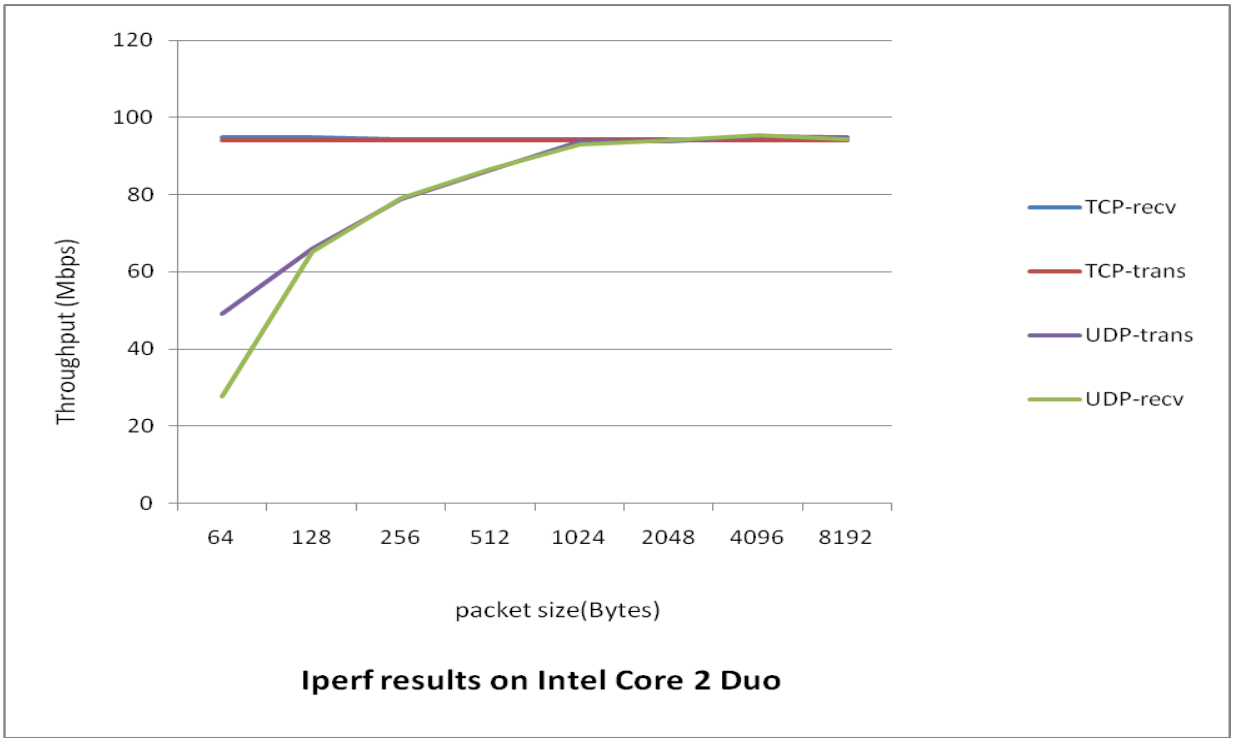


Figure 22: résultats de test de IPERF pour le core 2 duo

On retient que, globalement, il y a une limitation du débit à 100 Mbit/s, ceci est dû au type de Switch Ethernet utilisé, 10/100, qui lui ne supporte pas un débit supérieur. Ce test doit être donc relancé en utilisant un Switch Ethernet Gigabit. On constate aussi que la performance se stabilise au dessus du seuil de 1024 KO. Par conséquent, on favorisera l'échange de données de taille supérieure à la valeur de ce seuil.

3.6 RAMspeed

3.6.1 Description

RAMspeed est un test de performance du cache et de la mémoire **[14]**. Il est rédigé en C ANSI, ne comportant des optimisations particulières pour le DSP. **RAMspeed** est constitué d'une suite de tests comportant des tests en entiers et en virgule flottante. Du fait que **Faraday** n'intègre pas une unité de calculs flottants, seuls les résultats en entier sont présentés. **RAMspeed** a été compilé dans l'environnement de compilation croisée de **VLX**, et exécuté sur Linux tournant sur un seul cœur dans le cas de **Faraday**. Dans le but de mettre à l'épreuve la gestion de la mémoire et du cache par Linux et le compilateur croisé de **VLX**, **RAMspeed** a été compilé dans **CCS** en le configurant pour utiliser toute la mémoire cache disponible, et exécuté en **stand-alone** sur un cœur de **Faraday**.

3.6.2 Résultats et analyse

Test	Faraday	Faraday CCS	Core2Duo
Copy	366	717	3175
Scale	308	546	3219
Add	429	681	3946
Triad	369	582	3944
Average	**368**	**632**	**3571**

Tableau 6: Résultats de RAMspeed

Le test **copy :** copier une zone mémoire **A** dans une autre zone mémoire B **(B=A)** en retournant la bande passante en Mbit/s.

Le test **scale** : avant de copier la zone mémoire **A**, on la multiplie par un scalaire **m** et on copie les résultats dans une autre zone mémoire **B (B=m*A)** en retournant la bande passante en Mbit/s.

Le test **Add** : additionner deux zones mémoires **A** et **B** et mettre le résultat dans une autre zone mémoire **C** (**C=B+A**) en retournant la bande passante en Mbit/s.

Le test **Triad** : réaliser les trois opérations au même temps **C= m*A+ B** (**A, B,C** sont des zones mémoires et **m** un scalaire) en retournant la bande passante en Mbit/s.

Le test **Average :** retourner la bande passante moyenne des tests **copy**, **scale**, **Add** et **Triad**.

On remarque (les deux premières colonnes de **Tableau 6**) qu'un rapport de 1,7 en moyenne se présente entre les performances mesurés des tests compilés dans l'environnement croisé de **VLX** et exécuté sur Linux tournant sur un cœur du Faraday, et les tests compilés dans **CCS** et exécuté en **stand-alone** sur un cœur de **Faraday** exploitant toute la mémoire cache disponible. Ce rapport met en question l'environnement de compilation croisée de **VLX** et rend à l'évidence les lacunes qu'il présente au niveau de la gestion des registres de configuration de la mémoire cache. Notons aussi que l'importance du dernier constat est légèrement minimisée du fait que Linux consomme aussi de la mémoire cache, ce qui n'est pas le cas quand le test est exécuté en **stand-alone**. D'un autre côté, un rapport de 5,6 en moyenne se présente entre les performances de **RAMspeed** sur le **Core 2 Duo** et de **RAMspeed,** compilé dans **CCS**, et exécuté sur le **Faraday** (les deux dernières colonnes de **Tableau 6**). Ce rapport est dû à la différence qui existe au niveau du bus de données, reliant la mémoire externe et le CPU, entre **Faraday** et le **Core 2 Duo**. En fait, sur le **Core 2 Duo**, ce bus est double banc mémoire de 64 bits de large chacun, alors qu'il y a un seul banc de 32 bits de large sur le **Faraday**. Cette différence contribue d'un facteur 4 (2 par 2) dans les résultats.

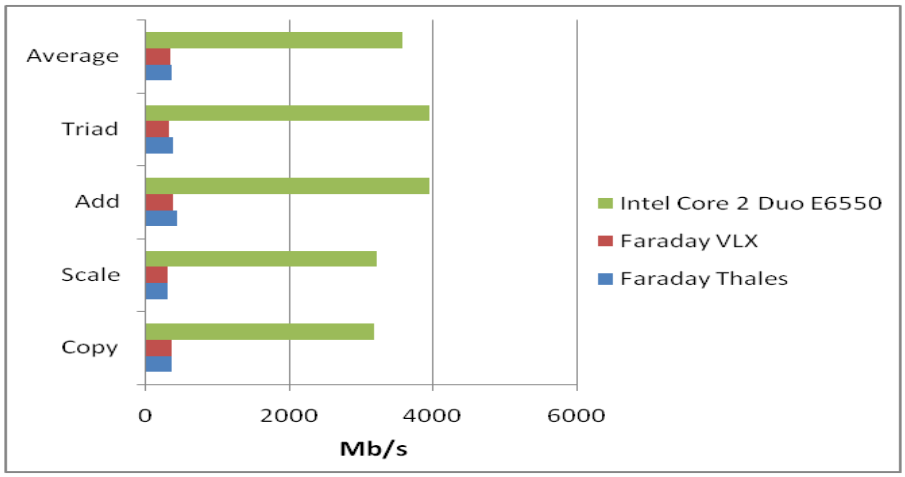


Tableau 7: Résultats Thales et VLX de RAMspeed sur Faraday

On remarque que les résultats des mesures que nous avons effectuées sont conformes avec celles fournies par **VLX**.

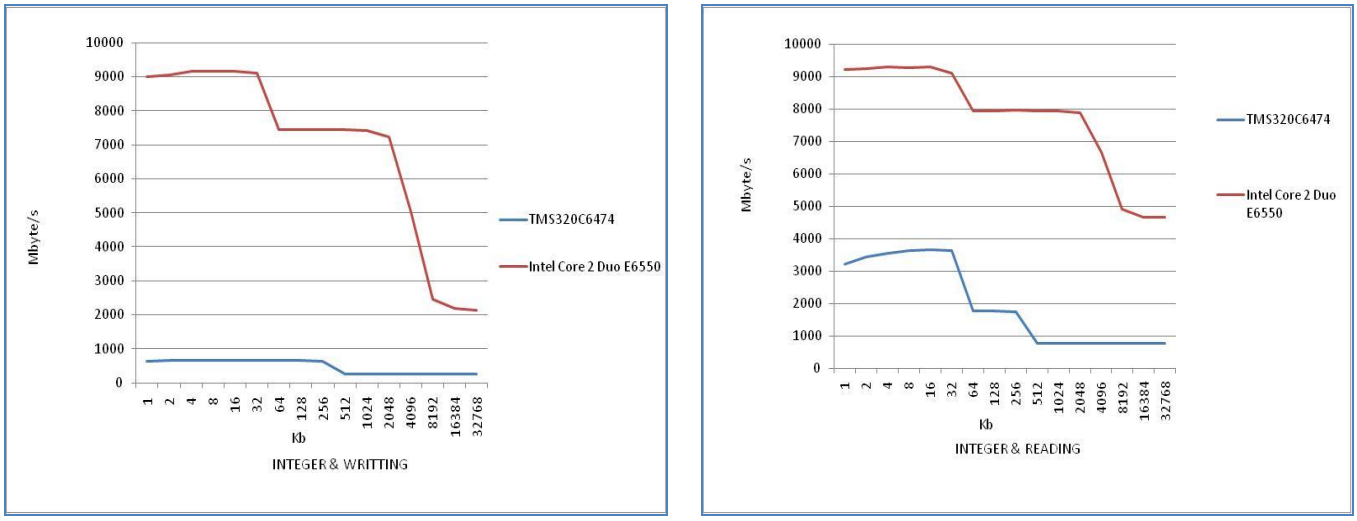


Figure 23: Résultats de Ramspeed lecture et écriture pour Faraday et le Core 2 Duo

3.7 CacheBench

3.7.1 Description

CacheBench est un test de performance élaboré dans le but d'évaluer la performance de la hiérarchie de la mémoire [**15].** Le test prend en compte dans la mesure de performances le paramètre indiquant la multiplicité des niveaux que présenterait la mémoire. Le but de ce test de performance est d'établir un pic du débit de calcul, étant donné une réutilisation optimum du cache et de vérifier l'effectivité des optimisations haut niveau du compilateur. **CacheBench** incorpore 8 différents tests de performance. Chacun effectue des accès répétés à des éléments de données de longueur vectorielle variable, tout en rassemblant le temps pris par les itérations et tout en calculant la bande passante de la mémoire en MB/s. On s'intéressera aux tests suivants :

- **Cache Read** : fournit la bande passante en lecture de vecteurs de taille variable dans une boucle optimisée par le compilateur.
- **Cache Write** : fournit la bande passante en écriture de vecteurs de taille variable dans une boucle optimisée par le compilateur.
- **Cache Read/Modify/Write** : fournit la bande passante en lecture modification et écriture de vecteurs de taille variable dans une boucle optimisée par le compilateur.
- **Memset()** : fournit la performance de la fonction C **memset()** (remplie une zone mémoire avec un octet donné).

- **Memcpy()** : fournit la performance de la fonction C **memcpy()** (copie n octets depuis la zone mémoire de source vers la zone mémoire de destination. Les deux zones ne doivent pas se chevaucher).

3.7.2 Résultats et analyse

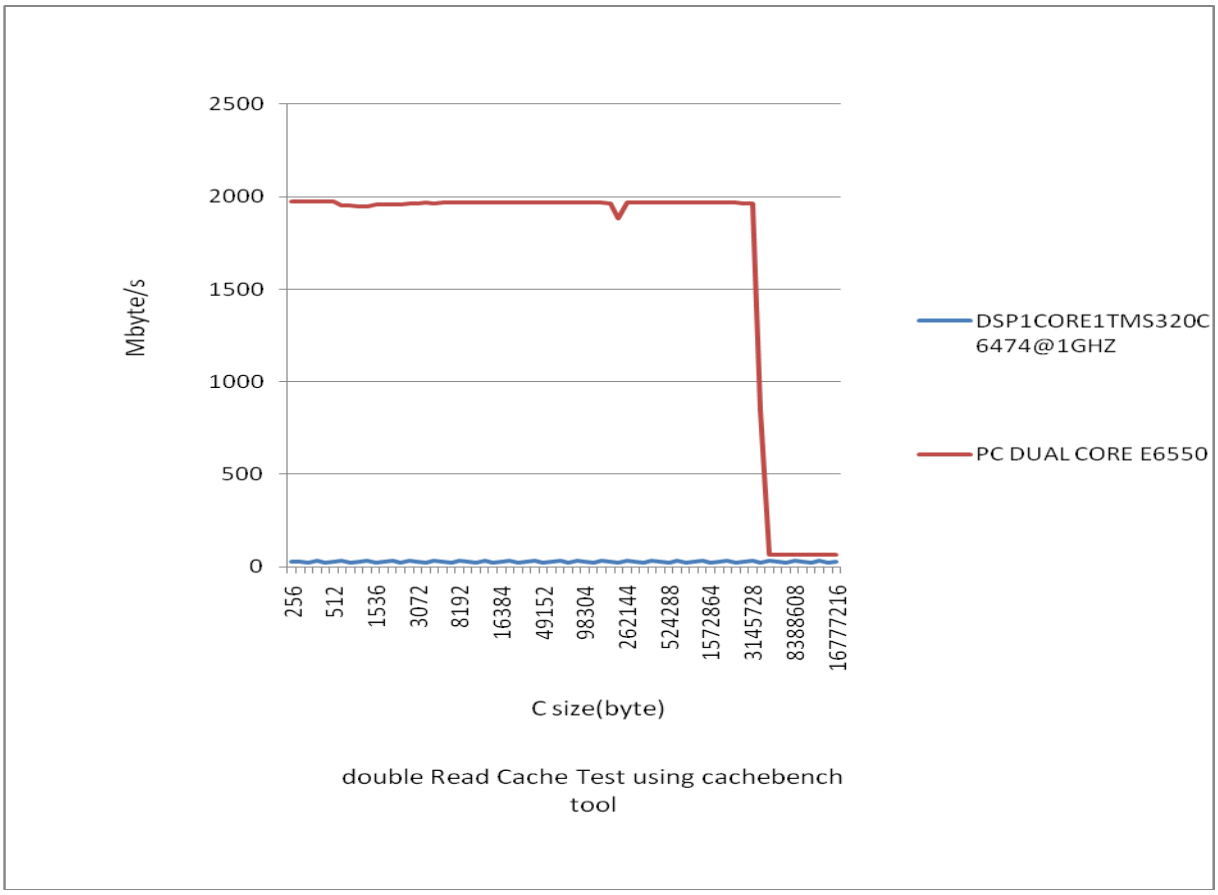


Figure 24: Résultats de CacheBench-Read pour Faraday et le Core 2 Duo

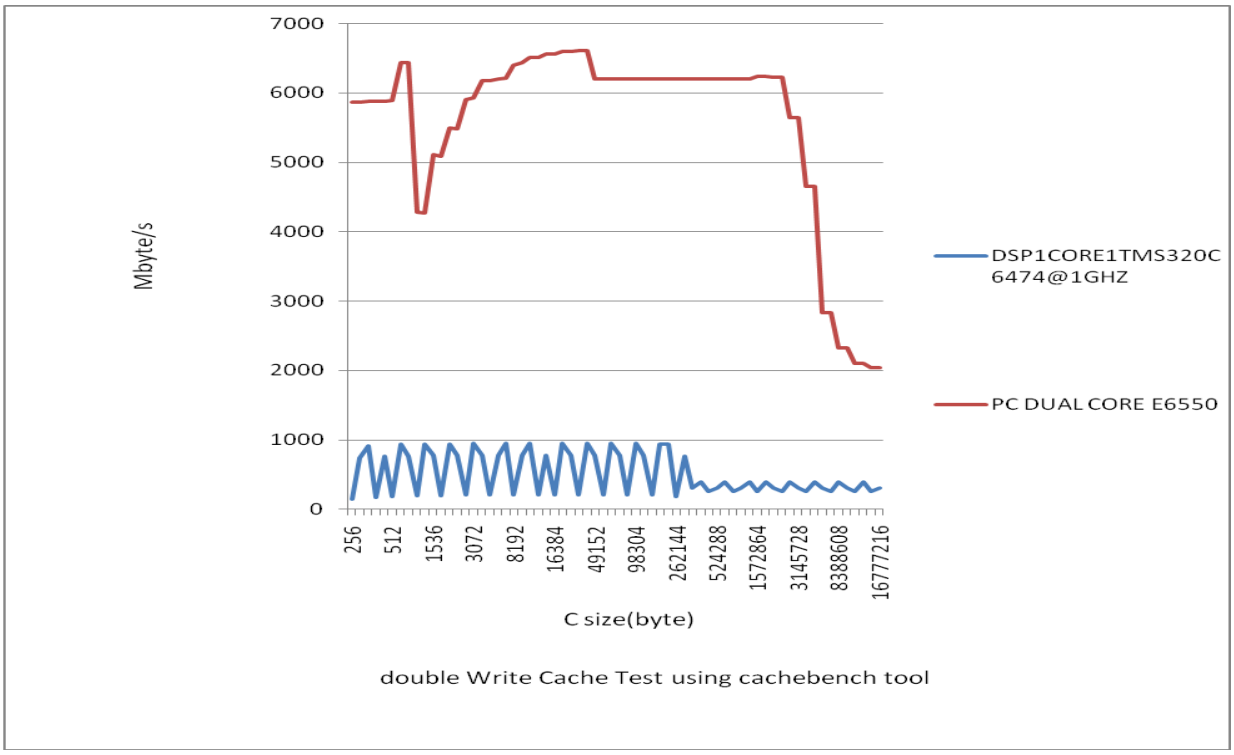


Figure 25: Résultats de CacheBench-Write pour Faraday et le Core 2 Duo

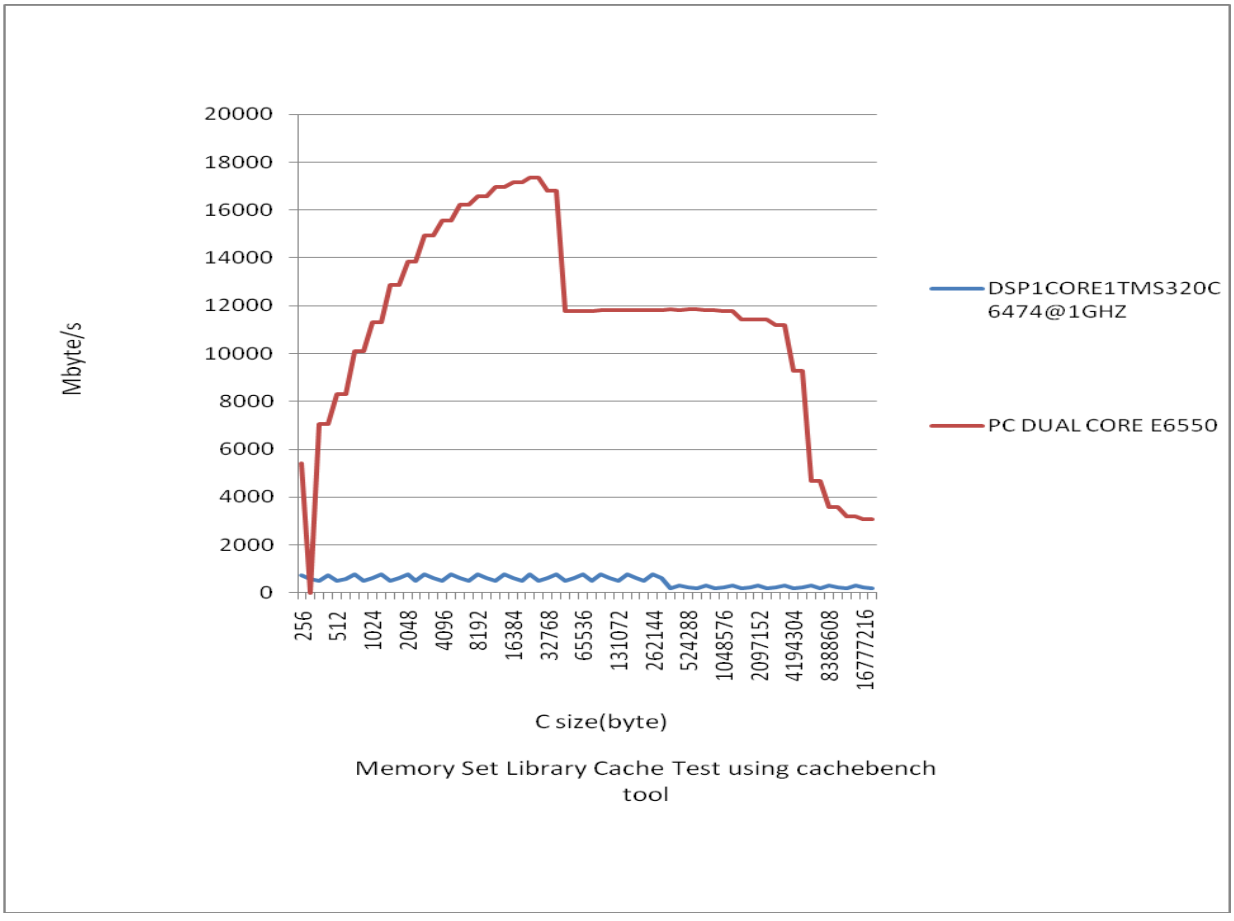


Figure 26: Résultats de CacheBench-memset pour Faraday et le Core 2 Duo

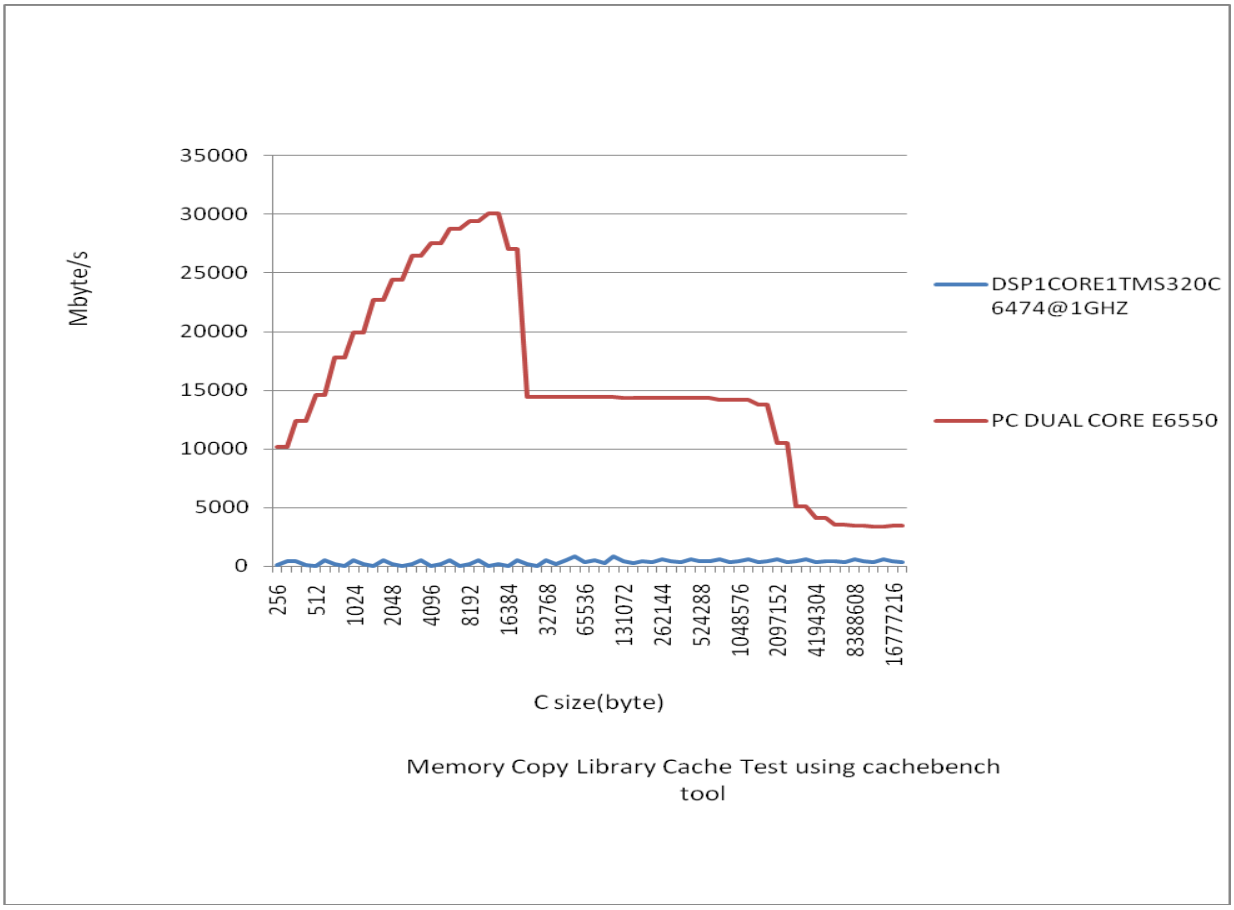


Figure 27: Résultats de CacheBench-memcpy pour Faraday et le Core 2 Duo

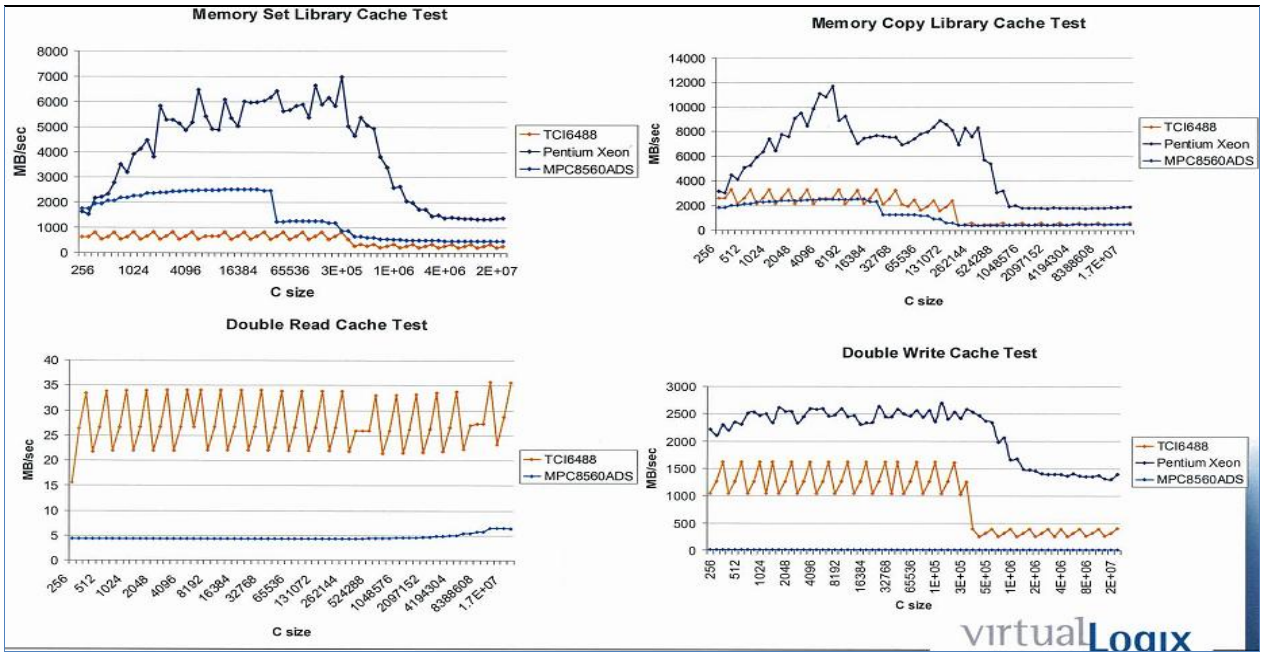


Figure 28: Résultats de CacheBench-memcpy pour Faraday VLX

Les figures 24, 25, 26 et 27 montrent qu'un écart considérable existe entre les résultats de **cachebench Read , write, memory set** et **memory copy** trouvés pour le **core 2 duo** et ceux d'un cœur de TMS320C6474 (**Faraday** testé par Thales).

La mesure des performances de la fonction **memset()** ainsi que **CacheBench Read**, effectuée sur **Faraday**, donne des résultats similaires à ceux publiés par VLX (figure 28). Par contre, ils sont inférieurs dans les cas de la fonction **memcpy()** ainsi que celui de **CacheBench Write**. Toutefois, pour ce dernier, les résultats présentent la même allure dans les deux cas.

Par rapport à **CacheBench**, le fait que **RAMspeed** teste aussi la mémoire du système et mesure la bande passante, pourrait impliquer que les deux tests doivent fournir les mêmes résultats. Ce qui n'est pas le cas car les deux tests n'opèrent pas sur le même domaine de tailles de données utilisées. **RAMspeed** concerne les données de petits multiples de la taille du cache, alors que **CacheBench** manipule aussi des blocs de données relativement volumineux. Cette différence ne manque pas de perturber les performances. Il est à rappeler que les conclusions tirées de **RAMspeed** sont de rigueur pour **CacheBench**, et notamment concernant le rapport de performance 4 entre le **Core 2 Duo** et **Faraday** qui découle de la différence matérielle qu'il y a concernant le bus de données des deux architectures.

Remarque : on a l'impression que les résultats de test **Cachebench** de TMS320C6474 sont nuls du fait de la représentation en échelle linéaire, pour cela on ajouté une représentation de test **Cachebench** en échelle semi logarithmique dont on a appliqué le logarithme décimal sur l'axe des ordonnées après transformation où l'unité de débit est en bit/s.

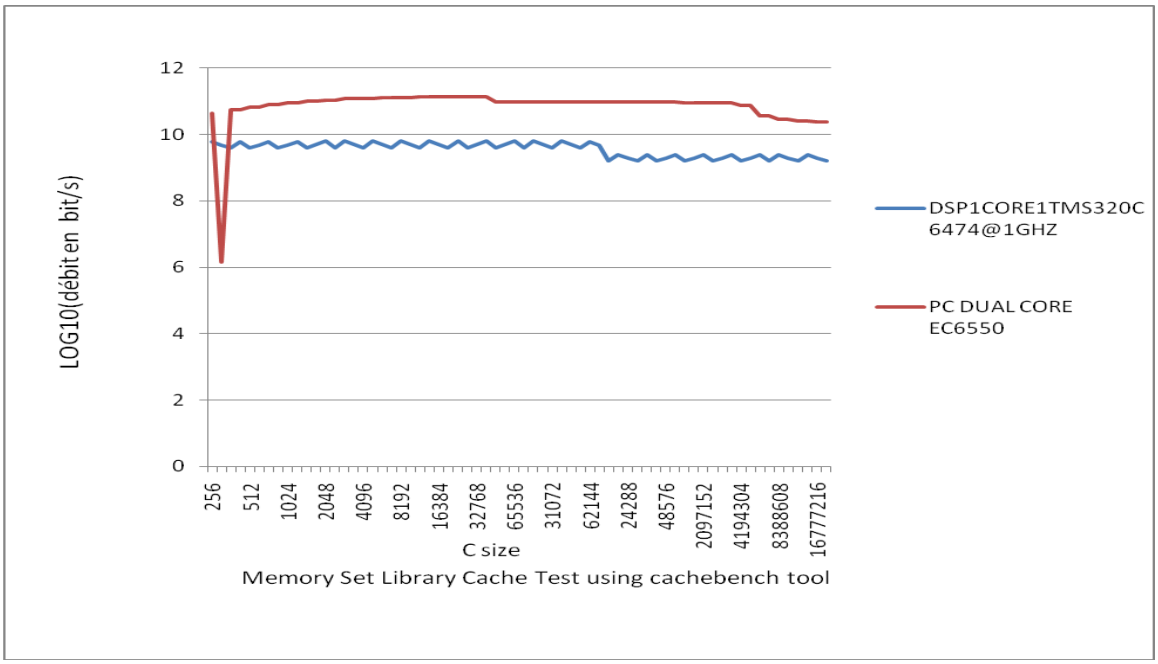


Figure 29: Résultats de CacheBench-memset pour Faraday et le Core 2 Duo en échelle semi logarithmique

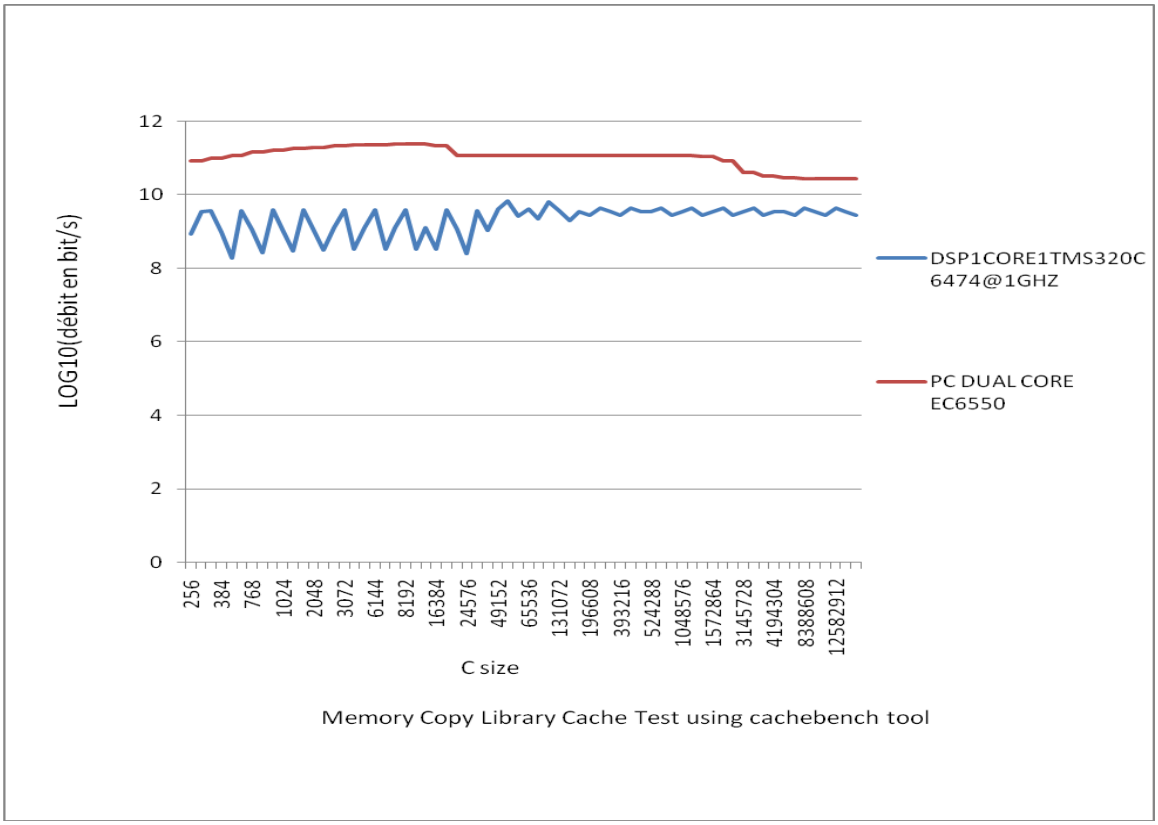


Figure 30: Résultats de CacheBench-memcpy pour Faraday et le Core 2 Duo en échelle semi logarithmique

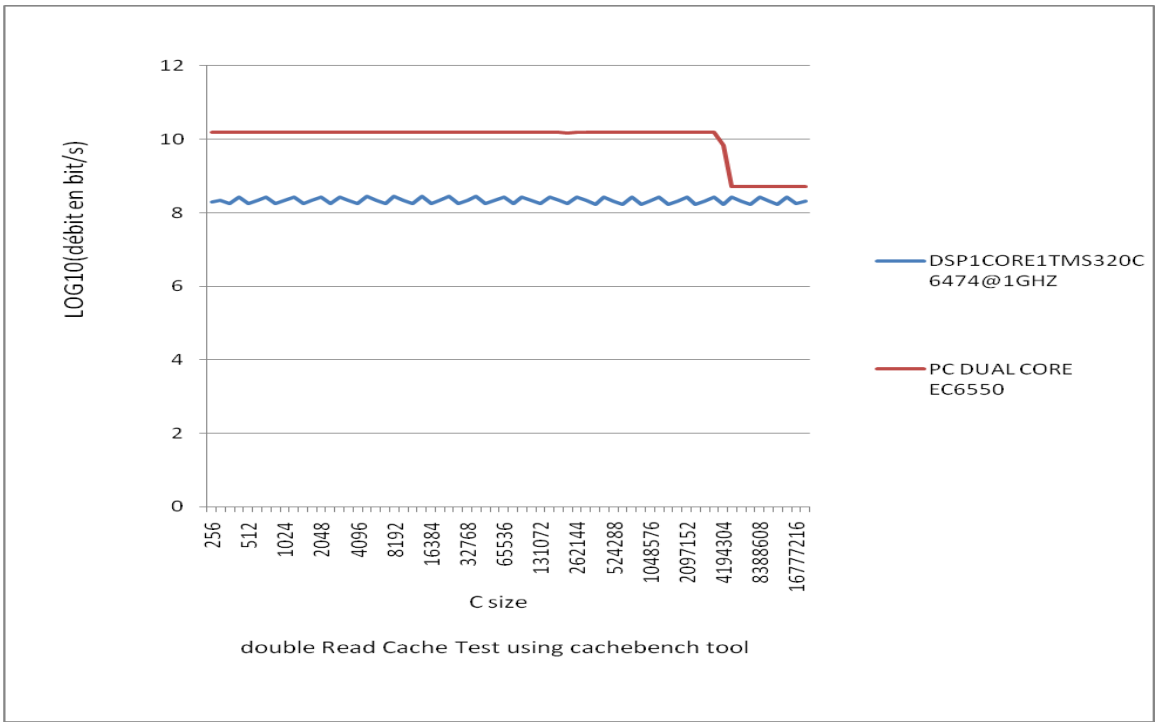


Figure 31: Résultats de CacheBench-Read pour Faraday et le Core 2 Duo en échelle semi logarithmique

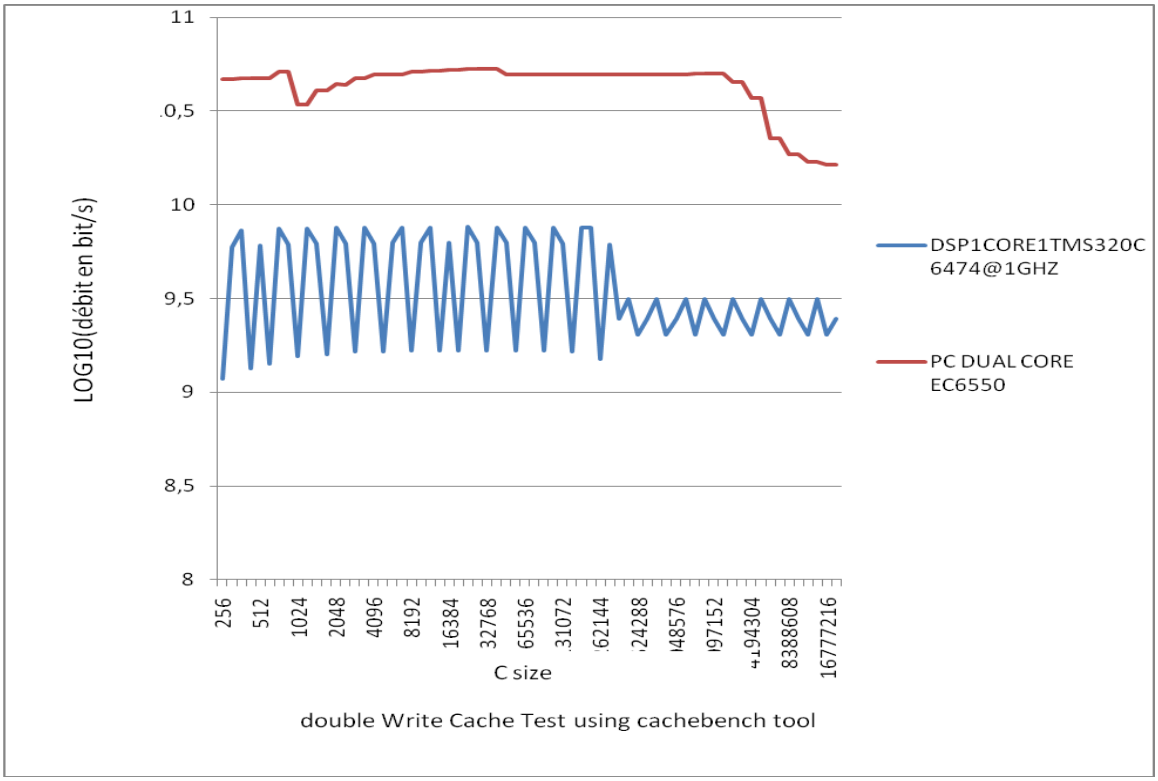


Figure 32: Résultats de CacheBench-Write pour Faraday et le Core 2 Duo en échelle semi logarithmique

3.8 NBench

3.8.1 Description

NBench est un test de performance standard du CPU, FPU (**Floating point Unit**) et la mémoire système **[16]**. Rédigé en C ANSI, il utilise des algorithmes, tels que l'algorithme de **Huffman**, afin d'établir les évaluations de l'architecture cible.

3.8.2 Résultats et analyse

Processor	Faraday	Core 2 Duo
MHz	1000	2331
W per core	2	32,5
MEMORY	5,7	17,5
INTEGER	7,3	16,3
FLOAT	0,07	30,5

Tableau 8: Résultats de Nbench en indice

TEST	Faraday	Core 2 Duo
NUMERIC SORT	3,65	8,27
STRING SORT	3	10
BITFIELD	5	15,4
FP EMULATION	20	16
ASSIGNMENT	12	35
IDEA	15	29
HUFFMAN	2,5	18,5
FOURIER (*)	0,06	17
NEURAL NET (*)	0,07	28
LU DECOMPOSITION(*)	0,11	58

Tableau 9 : Résultats de Nbench par algorithme en indice

On peut déduire à partir de ces résultats que le **Core 2 Duo** équivaut 3 **Faradays**. Remarquons que ce rapport n'est pas de rigueur dans le cas des algorithmes annotés de (*). En fait, il y a une chute des performances de **Faraday** pour ces algorithmes. En regardant de prés le code source de ces algorithmes, on trouve que l'implémentation de ces algorithmes implique des calculs en virgule flottante. Or, **Faraday** n'intègre pas une unité de calculs flottants, ce qui explique les pauvres performances de **Faraday** pour ces tests.

Noter qu'à la sortie de **Nbench**, les résultats sont présentés sous trois formes: itération/seconde, ancien index et nouvel index. L'index correspond à la division des résultats en itération/seconde obtenus par ceux d'une autre machine. En l'occurrence, Pentium 90 pour l'ancien index et **AMD K6/233** pour l'autre. Dans cette étude, les résultats sont indexés par rapport à **AMD K6/233**.

3.9 LMBench

3.9.1 Description

LMBench est un logiciel de test de la performance du noyau Linux **[17].** Il effectue les mesures suivantes :

Mesure de la bande passante

- Lecture de fichiers dans la mémoire cache
- Copie de la mémoire (**bcopy**)
- Lecture de la mémoire

- Ecriture dans la mémoire
- Pipe
- TCP

Mesure de latence

- Changement de contexte
- Réseau: établissement de connexion, pipe, TCP, UDP, et RPC
- Création et suppression dans le système de fichiers
- Création de processus
- Gestion des signaux
- Latence des Appels système
- Latence de la lecture de la mémoire

3.9.2 Résultats et analyse

Host	OS	intgr bit	intgr add	intgr mul	intgr div	intgr mod
Faraday	Linux 2.6.13-jaluna	**1,08**	**0,98**	**3,81**	**37,19**	**28,36**
Core 2 Duo	Linux kernel 2.6.27.19-170.2.35.fc10.i686	**0,43**	**0,38**	**0,18**	**15,9**	**7,99**

Tableau 10: Résultats de LMbench pour les opérations arithmétique en nanosecondes

- **intgr bit** est une opération de bits évalue deux valeurs intégrales sous forme binaire (base 2). Elles comparent les bits à des positions correspondantes, puis assignent des valeurs en fonction de la comparaison (integer x=3 and 5 =1).
- **Intgr add** est une opération d'addition de deux entiers
- **Intgr mul** est une opération de multiplication de deux entiers

- Intgr **div** est une opération de division de deux entiers
- Intgr **mod** est une opération de modulo de deux entiers

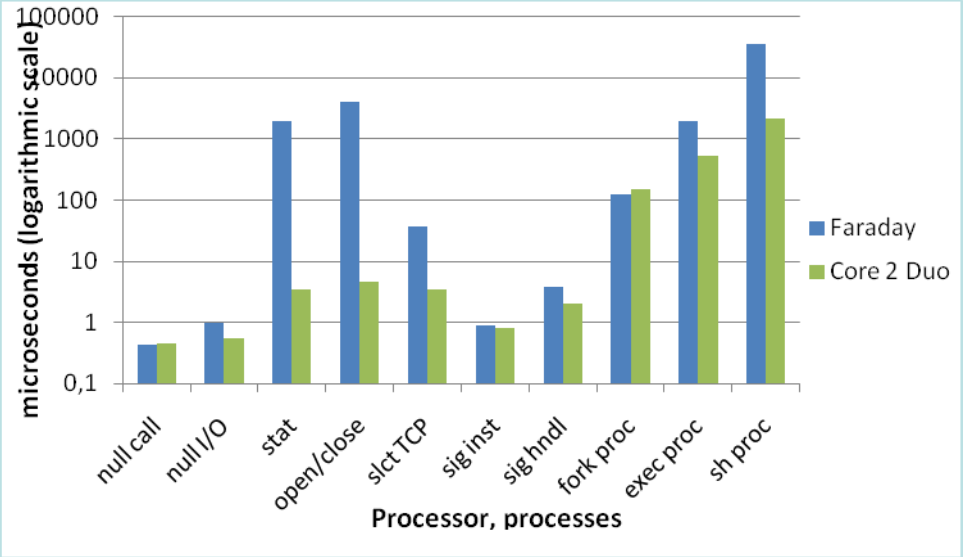


Figure 33: Résultats de LMbench pour les tests du système

- Le test de performance **stat** mesure le temps d'invoquer l'appel de l'état système sur un fichier temporaire.
- Le test **open/close** mesure le temps pour ouvrir un fichier temporaire pour la lecture et le ferme immédiatement.
- Les valeurs pour les tests de **open/close** et **stat** changent en fonction de répertoire où se trouve le fichier à ouvrir. Ainsi ces résultats peuvent être mieux que ce qui est présenté dans la figure en dessus.
- Le test **null I/O** mesure la moyenne en temps pour la lecture et l'écriture d'un octet de /à **/dev/zero**.
- **Sig inst** et **sig hndl** mesurent le temps pour installer et cacher un signal respectivement.
- Les tests **fork**, **execve**, et **sh** mesurent trois formes de plus en plus complexes en termes de la création d'un processus:

  - **Processus fork+exit**

    Le temps qu'il faut pour couper un processus en deux (ou presque) des copies

identiques et avoir une sortie. C'est ainsi que de nouveaux processus sont créés, mais n'est pas très utile puisque les deux processus sont fait la même chose.

- **Processus fork+execve**

Le temps qu'il faut pour créer un nouveau processus et que ce nouveau processus a lancé un nouveau programme. Il s'agit de la boucle interne de tous les Shells (interpréteurs de commandes).

- **Processus fork+/bin/sh -c**

Le temps qu'il faut pour créer un nouveau processus et que ce nouveau processus a lancé un nouveau programme en demandant le shell du système pour trouver ce programme et l'exécuter. C'est ainsi que le système de l'interface de la bibliothèque C appelé est mis en œuvre. Il est le plus général et le plus complexe.

Les résultats de **LMbench** confirment la tendance selon laquelle le **Core 2 Duo** serait plus performant que **Faraday**. Par contre, il faut noter que certains tests sont élaborés pour tourner sur des stations de travail, donc nécessitant la présence du système de fichiers sur un disque, alors que **Faraday** est exploité par un Linux dont le système de fichiers est monté par **NFS**. De plus, certains tests n'étaient pas en mesure de fonctionner, ou bien donnaient lieu à des résultats aberrants, du fait qu'ils se basent sur la présence d'une **MMU (**unité de gestion mémoire; responsable de l'accès à la mémoire demandée par le processeur), alors qu'elle n'est pas présente. Ces tests devraient être donc modifiés pour les adapter à **Faraday**.

3.10 Conclusion

En considérant les résultats des différents tests effectués, on peut déduire que :

- Globalement, les résultats des mesures effectuées sur **Faraday** sont conformes avec ceux annoncés par **VLX** sur la même carte.
- Le processeur d'Intel, le **Core 2 Duo E6550**, est plus performant que **Faraday**.
- La présence d'une **FPU** sur le **Core 2 Duo**, une unité que **Faraday** n'a pas, explique l'infériorité des résultats de cette dernière dans les tests mettant en œuvre des calculs en virgule flottantes.
- Le bus de données entre le processeur et la mémoire externe, étant 4 fois plus rapide sur le **Core 2 Duo** que sur **Faraday**, introduit un facteur 4 dans les performances entre ces deux derniers.

- Le compilateur intégré dans **CCS** est approximativement 2 fois plus efficace que le compilateur croisé.
- La mémoire cache n'est pas proprement gérée par Linux par-dessus de **Faraday**
- La bande passante à travers Ethernet est limitée à 100 Mbit/s par le switch 10/100 utilisé. Le test devrait être repassé en utilisant un switch Gigabit.
- Le système de fichiers de Linux étant monté par **NFS** à travers l'Ethernet, ce qui perturbe la mesure de la bande passante Ethernet.
- Les tests du système ont mis en avant des faiblesses du Linux sur Faraday (Système de fichier monté par **NFS**, absence de **MMU**, etc.)

Donc pour conclure :

- **Faraday** est 3 fois plus performant que **ColdFire**, et 3 fois moins performant que le **Core 2 Duo**

- Des évolutions futures sont attendues de la part de TI : une mémoire **DDR3**, une **FPU**, et un Linux intégré à **CCS**

Chapitre 4 : Communication inter cœurs et inter DSPs

4 Communication inter cœurs et inter DSPs

Dans ce chapitre nous allons réaliser la communication inter cœur et inter DSP via les mécanismes qu'on va introduire dans les parties suivantes. Ensuite nous allons présenter quelques applications à partir des exemples de Texas adaptés et compilés pour le TMS320C6474. Avec ces mécanismes et en exploitant le protocole Serial Rapid IO de la communication à haut débit entre les DSPs, nous allons réaliser approximativement à travers une application, les briques de base de la supervision radar numérique nouvelle génération.

4.1 La Communication inter cœurs

La communication inter cœurs est l'échange de l'information entre les cœurs dans un multi-cœur. Dans cette partie, nous allons présenter brièvement les modules de base pour communiquer entre les différents cœurs du DSP TMS320C6474 tels que les modules **sémaphores** et **IPC**. Ensuite nous allons voir deux exemples de Texas qui utilisent ces mécanismes de communication inter cœurs **[18].**

4.1.1 Le module sémaphore [19]

Dans le DSP TMS320C6474, plusieurs processeurs tentent d'accéder simultanément à des ressources partagées. Pour éviter tout conflit de ressources, le module sémaphore est utilisé pour accéder à des ressources partagées en respectant l'exclusion mutuelle. D'autre part, le module sémaphore est atomique afin de satisfaire la lecture-modification-écriture avec succès.

Le module sémaphore a une interruption unique pour chaque cœur de DSP (maîtres) afin de déterminer quand ce maître a acquis la ressource demandée. De même, il y a des interruptions d'erreur propres à chaque cœur de DSP quand une tentative de maître particulier pour accéder à la ressource sémaphore qui est déjà verrouillée par le même ou autre maître. Pour chaque périphérique de sémaphore, Il existe trois différents mécanismes pour acquérir la ressource: direct, indirect ou combiné.

- **Accès direct** : Un cœur accède directement à une ressource de sémaphore. Si libre, le sémaphore sera accordé. Sinon, le sémaphore n'est pas accordé.

- **Accès indirect :** Un cœur accède indirectement à une ressource sémaphore en écrivant à un des trois registres : **DIRECTx**, **INDIRECTx** ou **QUERYx [20]**(x est le numéro de périphérique de sémaphores). Une fois qu'il est libre, une interruption informe le processeur concerné (qui a fait la requête) qu'il est disponible.

- **Accès combiné :** dans ce mode, une ressource est accédée en lisant le registre **INDIRECTx** (x est le numéro de périphérique de sémaphores). Si la ressource est libre, l'accès est immédiatement accordé au cœur qu'a fait la requête. Si la ressource n'est pas disponible, la requête est ajoutée à la file d'attente des requêtes assignée aux ressources sémaphores. Une fois libre, une interruption informe le processeur concerné (qui a fait la requête) que la ressource est disponible.

  L'exemple suivant montre deux CPUs tentent à accéder au même périphérique A,

  Durant l'initialisation, le périphérique A a été assigné au sémaphore numéro 2

  1. le cœur 1 tente à accéder au sémaphore 2
  2. le cœur 2 tente à accéder au sémaphore 2
  3. le cœur 1 accède au sémaphore 2
  4. le cœur 2 mis en attente pour accéder au sémaphore 2
  5. le cœur 1 accède au périphérique A
  6. le cœur 1 libère le sémaphore 2
  7. le cœur 2 accède au sémaphore 2
  8. le cœur 2 accède au périphérique A
  9. le cœur 2 libère le sémaphore 2

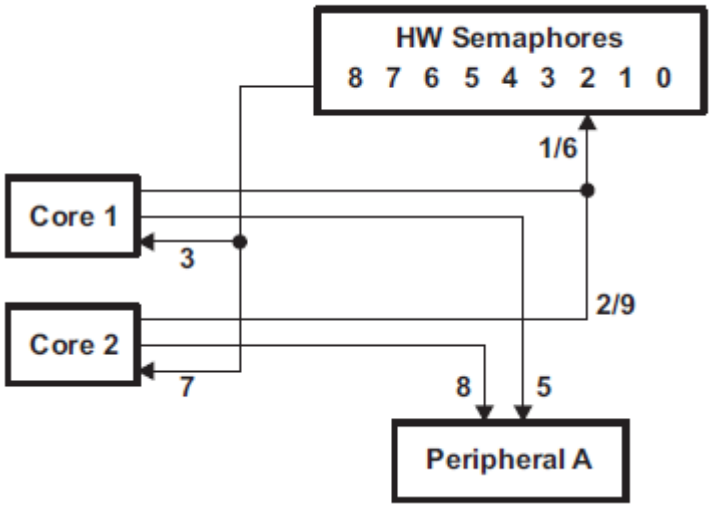


Figure 34: exemple de deux cœurs tentent à accéder au même périphérique A. [21]

Le deuxième exemple illustre la gestion de l'accès à la même ressource en utilisant les modules sémaphores :

CPU A	CPU B
1. Tente à acquérir le verrou	1. Tente à acquérir le verrou
-> Passe : verrouillage réussi	-> verrouillage échoué
2. Modifier la ressource	2. Répéter 1 jusqu'à ce qu'il passe
3. Libérer la ressource	-> Passe : verrouillage réussi
	3. Modifier la ressource
	4. Libérer la ressource

Tableau 11: exemple de deux cœurs tentent à verrouiller même ressource

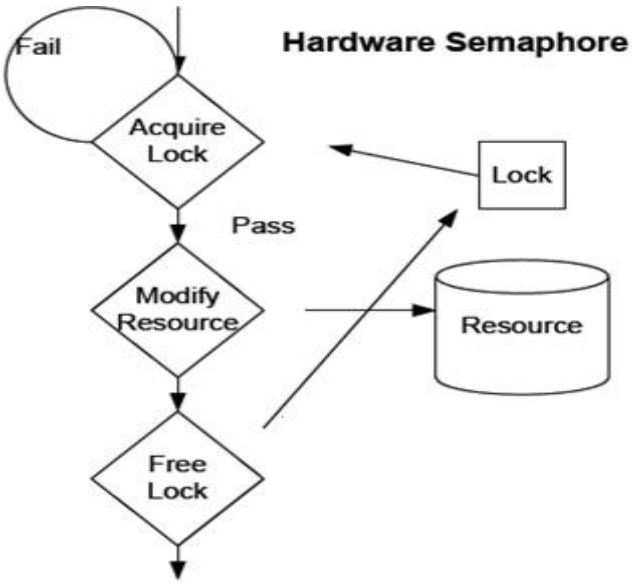


Figure 35: exemple de deux cœurs tentent à verrouiller même ressource. [22]

4.1.2 Le module IPC (Inter processus communication)

Les registres IPCGRn (n=0,1,2) et les registres IPCARn (n=0,1,2) facilitent les interruptions inter-cœurs. Ceci peut être utilisé par les hôtes externes ou C64x+ méga modules pour générer des interruptions aux autres cœurs. L'écriture d'un **1** dans le champ IPCG de registre IPCGRn génère une impulsion d'interruption au cœur correspondant. Ce registre permet d'avoir la source de l'interruption pour lequel jusqu'à 28 sources différents d'interruptions peuvent être identifiées **[23].**

- **IPC Generation Registers (IPCGR0 – IPCGR2) [Tableau 12]:**

Les registres IPCGR (IPC **Generation Registers**) sont écrits par le cœur source pour générer une interruption, éventuellement avec des flags, au cœur de destination. Il y a trois entrées, une par cœur de destination: IPCGR0/1/2 pour générer une interruption pour le cœur 0/1/2, respectivement **[23].**

A noter que tous les cœurs génèrent la même interruption; certains flags SRCS peuvent être utilisés pour déterminer celui qui a envoyé cette interruption.

31	30	29	28	27 … 6	7	6	5	4	3 … 1	0
SRCS27	SRCS26	SRCS25	SRCS24	…	SRCS3	SRCS2	SRCS1	SRCS0	Reserved	IPCG
RW+0	RW+0	RW+0	RW+0	…	RW+0	RW+0	RW+0	RW+0	RW+000	RW+0

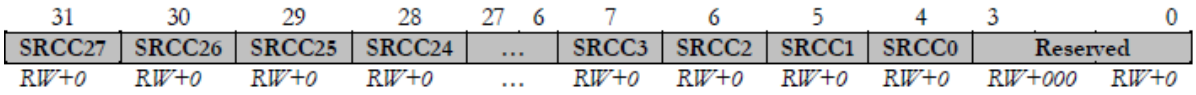
Bit	champ	description
31 :4	SRCS[27 :0]	L'ecriture : 0 Pas d'effet 1 Modification des bits de registre Lecture Retourne la valeur courante de bit de registre
3 :1	Reserved	Reservés
0	IPCG	L'ecriture : 0 Pas d'effet 1 Création d'une impulsion d'interruption vers le destinataire (C64X+ mega modules0 pour IPCGR0…) Lecture Retourne 0, pas d'effet

Tableau 12: Description des champs du registre IPCGRn

- **IPC Acknowledgement Registers (IPCAR0 – IPCAR2)[Tableau 13]**

IPCAR (IPC Acknowledge Registers) sont utilisés par les cœurs de destination pour supprimer les bits SRCS. Il y a trois de ces registres: IPCAR0/1/2 qui sont utilisés par les cœurs 0/1/2, respectivement.

Le cœur source modifie le flag en écrivant 1 aux bits SRCS dans le registre IPCGR le cœur de destination va les effacer en écrivant 1 aux bits SRCC dans le registre IPCAR. Le cœur source ne peut pas effacer les bits de SRCS. **[23]**



31	30	29	28	27 6	7	6	5	4	3 0	
SRCC27	SRCC26	SRCC25	SRCC24	…	SRCC3	SRCC2	SRCC1	SRCC0	Reserved	
RW+0	RW+0	RW+0	RW+0	…	RW+0	RW+0	RW+0	RW+0	RW+000	RW+0

Bit	champs	descriptions
31 :4	SRCC[27 :0]	L'écriture 0 Pas d'effet 1 Effacer les bits de registres de cœur source Lecture Retourne la valeur courante de bits d e registre
3 :1	Reserved	Reservés

Tableau 13: Description des champs du registre IPCGRn

4.1.3 Exemples de programmes pour la communication inter-coeurs

L'annexe 6 présente la compilation, l'exécution et l'encapsulation de deux exemples de Texas de la communication inter-cœurs qui utilisent les modules vus précédemment.

Le premier exemple **[23]** montre la capacité du DSP TMS320C6474 d'exécuter du code identique sur les trois cœurs. Ce code effectue des transferts EDMA d'un cœur à l'autre, à partir de cœur0. Il envoie ensuite une interruption IPC au cœur de réception pour l'informer que des données sont arrivées. Quand un cœur reçoit un IPC, il éditera ces données et commencera un nouveau transfert EDMA pour l'envoyer au cœur suivant, dans l'ordre. Lorsque le transfert est complet, il enverra un IPC au cœur de réception pour qu'il fasse la même chose.

Le deuxième exemple **[24]** est un programme qui s'exécute sur tous les trois cœurs. Chaque cœur envoie des messages via les registres IPC vers les deux autres cœurs. Le message est envoyé périodiquement en se basant sur les événements de **timer** de la carte EVM6474. Lorsque le cœur qui génère le message (le cœur source) écrit dans le registre IPCGR qui correspond au cœur de réception (le cœur de destination), le cœur de destination reçoit une interruption. Cela correspond à la routine de service d'interruption (**ISR : Interruption Service Routine**), qui s'exécute sur le cœur de destination, et qui détermine le cœur qui a été la source du message et écrit le sémaphore logiciel correspondant, il y a trois sémaphores logiciels au total, un par cœur. L'écriture (**Posting**) du sémaphore logiciel produit un changement de tâche qui était en attente (**pending)** sur le sémaphore. Il y a trois tâches et

chaque processus de messages est l'origine d'un cœur particulier. A noter que sur un cœur donné, uniquement deux tâches sont actives puisque, dans cet exemple, un cœur n'envoie pas un message à lui-même.

4.1.4 Mise en place de librairies logicielles A partir des exemples de Texas

La programmation sur **code composer studio** fait appelle souvent à des librairies complexes comme la librairie CSL (**Chip Support Library**) pour gérer les différents matériels, canaux et modules de DSP comme la gestion de l'EDMA, des modules sémaphores, IPC ou SRIO. Cela rend l'application longue illisible voire incompréhensible par l'utilisateur qui peut ne pas avoir les connaissances techniques sur ces détails. La mise en place d'une solution de l'encapsulation de ces programmes pour qu'ils soient plus lisibles et compréhensibles est indispensable. Pour avoir une encapsulation globale de programme sur CCS, on est amené à créer des nouvelles librairies qu'on appelle dans le programme principal.

Dans cette partie, nous allons créer une bibliothèque à partir d'un exemple de Texas instrument téléchargé à partir de son site internet, dont le nom du projet est **C6474_Edma_IPC_BIOS.pjt (annexe 6)** où Le programme **main.c** est surchargé par les fonctions avant encapsulation. L'encapsulation ici consiste à déclarer les différentes fonctions dans un en-tête (**all_functions.h**) ensuite les définir dans un autre fichier C **(all_function.c**) et enfin, appeler et utiliser ces fonctions dans un nouveau programme principal qu'on appelle aussi **main.c**. Notre bibliothèque va contenir donc les programmes **all_function.c** et d'autres programmes qui n'ont pas été utilisés directement dans le programme principal à savoir **EdmaIntDispatcher.c**, **Edma_IPC_BIOScfg_c.c** et **IpcIntDispatcher.C** (qu'existaient à l'extérieur de l'ancien programme **main.c** avec des fichiers sources). Le détail sur la création de cette bibliothèque qu'on l'a appelé **c6474_edma_ipc_bios_lib.lib** se trouve dans **l'annexe 7.**

4.2 La communication inter- DSP

La communication entre deux DSPs est assurée par le protocole de communication à haut débit, le **Serial Rapid IO (SRIO).** Son principe repose sur la commutation de paquets. Il est destiné principalement à être une interface intra-système entre puce à puce et carte-à-bord, il permet des niveaux de transfert allant aux gigaoctet par seconde. Ses domaines d'utilisations sont diverses et variés, il peut être trouvé dans l'interconnexion des microprocesseurs, la mémoire, la mémoire mappée entrée/sortie dans un réseau, sous-systèmes de mémoire, et dans l'informatique à usage général. Dans l'application qui va suivre, ce mécanisme sera utilisé pour assurer la communication entre le cœur 0 de DSP0 (DSP de calcul) et cœur 0 de DSP1 (DSP de supervision).

4.3 Application

Afin de mettre en œuvre la communication inter-cœurs et inter-DSP via le Protocole RapidIO, on a conçu et implémenté des algorithmes qu'on a développé et chargé dans les cœurs du premiers DSP et on a transféré, après calcul, les données vers l'autre DSP via SRIO

Dans notre étude on va prendre le cahier de charge suivant :

DSP1 : Ce DSP abrite les 3 cœurs où on va implémenter les algorithmes de communication inter- cœurs pilotés par un maître.

Cœur 0 : ce cœur va jouer le rôle du maître qui va manager le travail des cœurs 1 et 2. En effet, après avoir rempli deux zones mémoire dans la DDR ; ce dernier va ordonner aux cœurs 1 et 2 d'effectuer des calculs sur ces données. Par ailleurs, en notifiant la fin des calculs à l'hôte (host), ce dernier va entamer un transfert inter-DSP via SRIO des résultats.

Coeur1 : en tant que esclave, ce cœur va recevoir la commande du host (sous forme d'interruption), pour effectuer des calculs et stocker les résultats dans une zone mémoire dans la DDR (out1).

Cœur2 : il va jouer le rôle d'un esclave qui va recevoir la commande du hôte (host), effectuera les calculs et stockera les résultats dans une zone mémoire dans la DDR (out2)

Les commandes entre l'hôte et les deux esclaves seront gérées par des interruptions dont la gestion est effectuée par les registres IPC.

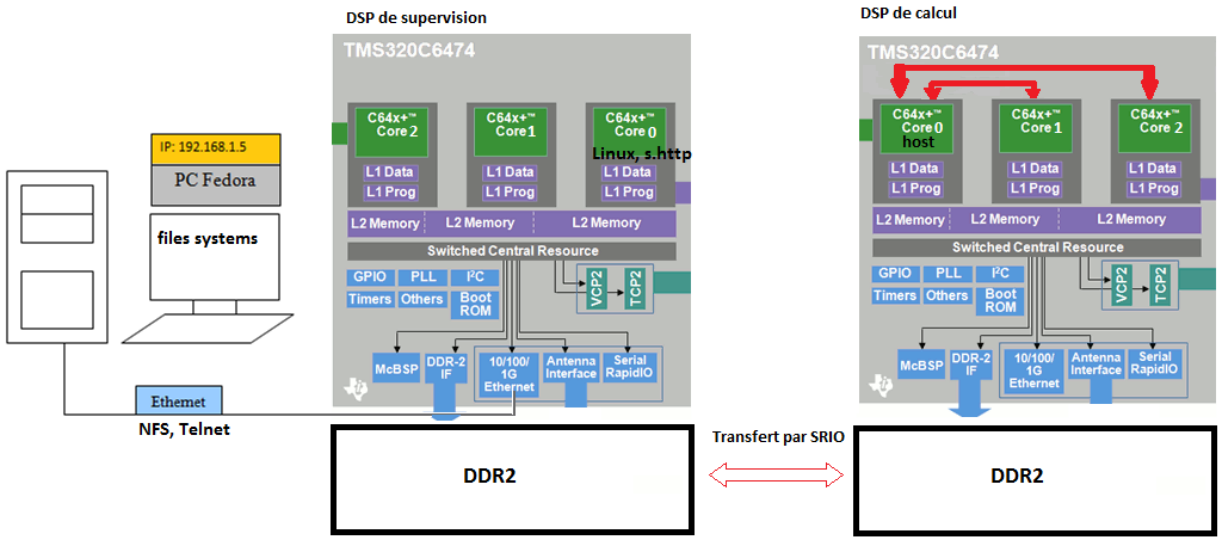


Figure 36: schéma fonctionnel de l'application

Les algorithmes de Cette application sont mises en œuvre en **annexe 8**

Notons que DSP/BIOS est un noyau spécifique aux processeurs de traitement de signal de Texas Instruments. Il existe des versions pour les familles TMS320C2x, TMS320C5x et TMS320C6x. Ce noyau fait partie intégrante de «Code Composer Studio ». Il utilise très peu de mémoire et de ressources sur la cible. C'est un noyau configurable : seules les primitives utilisées sont chargées dans la cible.

Cette application montre la capacité de TMS320C6474 d'exécuter des programmes et de réaliser la communication inter-cœurs via les mécanismes IPC, sémaphore, DSP / BIOS et SRIO. Un serveur http sera utilisé pour afficher les résultats de la supervision dans le centre de supervision (PC sous Fedora) à partir du DSP de supervision via le protocole NFS.

4.4 Serveur http

Une des principales fonctionnalités de la supervision Radar est la collecte de données sur les différentes cartes incorporées dans le Radar. Les données collectées sont stockées dans des fichiers qui seront envoyés vers la supervision centrale pour traitement. Cela implique que les DSPs doivent exécuter et vérifier tous les protocoles réseaux. Cette partie a pour but le portage de serveurs http, un des logiciels de base de la supervision. Les serveurs FTP, SNMP et DHCP seront portés à Rungis (France).

Un serveur HTTP ou serveur Web, est un logiciel servant des requêtes respectant le protocole de communication client-serveur **Hypertext Transfer Protocol** (HTTP), qui a été développé pour le **World Wide Web**.

Un ordinateur sur lequel fonctionne un serveur HTTP est appelé serveur Web. Le terme « serveur Web » peut aussi désigner le serveur HTTP (le logiciel) lui-même. Les deux termes sont utilisés pour le logiciel car le protocole HTTP a été développé pour le Web et les pages Web sont en pratique toujours servies avec ce protocole. D'autres ressources du Web comme les fichiers à télécharger ou les flux audio ou vidéo sont en revanche fréquemment servis avec d'autres protocoles.

Dans notre projet, les sources de serveur http sont données par Thales avec les outils de produit Octane. Donc ces sources fonctionnent déjà pour l'architecture de ColdFire et qui ont été légèrement modifiés pour être compilé pour le processeur *C64x+*. La compilation et l'exécution de ce serveur sont réussies. Son test l'est également. **L'annexe 9** présente les détails techniques pour le portage, la compilation, l'exécution et un test de fonctionnement de serveur http.

Conclusion :

Dans ce chapitre, nous avons pu réaliser la communication entre les cœurs de même DSP, la communication entre les cœurs des deux DSP. Ensuite nous avons pu afficher les pages html se trouvant avec les fichiers systèmes d'un DSP dans un central superviseur ou tout autre hôte en réseau. Maintenant il va falloir afficher les résultats du traitement à partir de DSP de supervision dans le centre de supervision en utilisant le serveur http et l'application présentée dans ce chapitre ainsi on accomplira les briques de base de la supervision des Radars numériques nouvelles générations.

Conclusion générale

Dans ce rapport, j'ai commencé d'abord par découvrir les Microprocesseurs de traitement Numérique de signal, spécialement le DSP TMS320C6474 sur lequel j'ai entamé mes études. La suite a été de maîtriser l'outil de travail « Ccstudio » qui sert à compiler, charger et exécuter les programmes en C/C++.

En essayant de répondre aux différents points de cahier de charges, nous avons commencé par l'étude des performances de noyau linux sur le DSP TMS320C6474 par rapport aux autres architectures en portant les différents outils de benchmarks. Les résultats de ces tests de performances ont montré qu'un cœur de TMS320C6474 est trois fois plus performant que de l'architecture ColdFire actuellement utilisée. Mais aussi des évolutions futures sont attendues de la part de TI : une mémoire DDR3, une FPU, et un Linux intégré à CCS. Ensuite le portage des logiciels de supervisions tels que http, SNMP, FTP et DHCP est considéré comme une étape nécessaire pour la supervision des radars numériques. Une dernière partie a été consacrée à étudier la communication inter cœurs et inter DSP afin de réaliser la couche logiciel pour les briques de base de la supervision.

Pour la plupart, ces concepts m'étaient encore inconnus, et leur utilisation a certes constitué un défi, mais a surtout été d'un grand apport.

Ce projet constitue un double expérience, technique et humaine, au sein d'une société qui porte une attention particulière aux compétences de ses acteurs, THALES. En outre l'interaction avec les membres de l'équipe THALES AIR SYSTEM a été fructueuse et m'a énormément facilité la tâche.

Le travail en groupe constitue certainement une composante essentielle de la vie professionnelle, et ce projet nous permettra sans aucun doute d'aborder le monde du travail de manière plus sereine, et nous donner un aperçu de notre futur métier d'ingénieur.

La formation acquise au sein de la faculté des sciences et Techniques de Fès a constitué notre rampe de décollage. Les connaissances techniques de base nous y ont été inculquées. Ce stage nous a permis d'affronter le monde réel et nous a rendus prêt pour le monde professionnel.

Documents Annexes

Annexe 1:Unités de mesure des performances des processeurs

Les unités les plus courantes des mesures des performances sont regroupées dans le tableau suivant :

Acronyme Anglais	Définition
MFLOPS	Million Floating-Point Operations Per Second
	Mesure le nombre d'opérations à virgule flottante (multiplications, additions, soustractions, etc.) que le DSP à virgule flottante peut réaliser en une seconde
MOPS	Million Operations Per Second
	Mesure le nombre total d'opérations que le DSP peut effectuer en une seconde. Par opérations, il faut comprendre non seulement le traitement des données, mais également les accès DMA, les transferts de données, les opérations d'E/S, etc. Cette définition mesure donc les performances globales d'un DSP plutôt que ses seules capacités de calcul.
MIPS	Million Instructions Per Second
	Mesure le nombre de codes machines (instructions) que le DSP peut effectuer en une seconde. Bien que cette mesure s'applique à tous les types de DSP, le MFLOPS est préféré dans le cas d'un DSP à virgule flottante.
MBPS	Mega-Bytes Per Second
	Mesure la largeur de bande d'un bus particulier ou d'un dispositif d'E/S, c'est à dire son taux de transfert.

Tableau 1: les définitions des unités les plus courantes de mesures des performances des DSP **[25].**

Annexe 2:Les Périphériques du DSP TMS320C6474

Dans ce document, on détaille la description des périphériques du TMS320C6474. On reprend le schéma fonctionnel du cœur du DSP, le C64x+ contenant les périphériques :

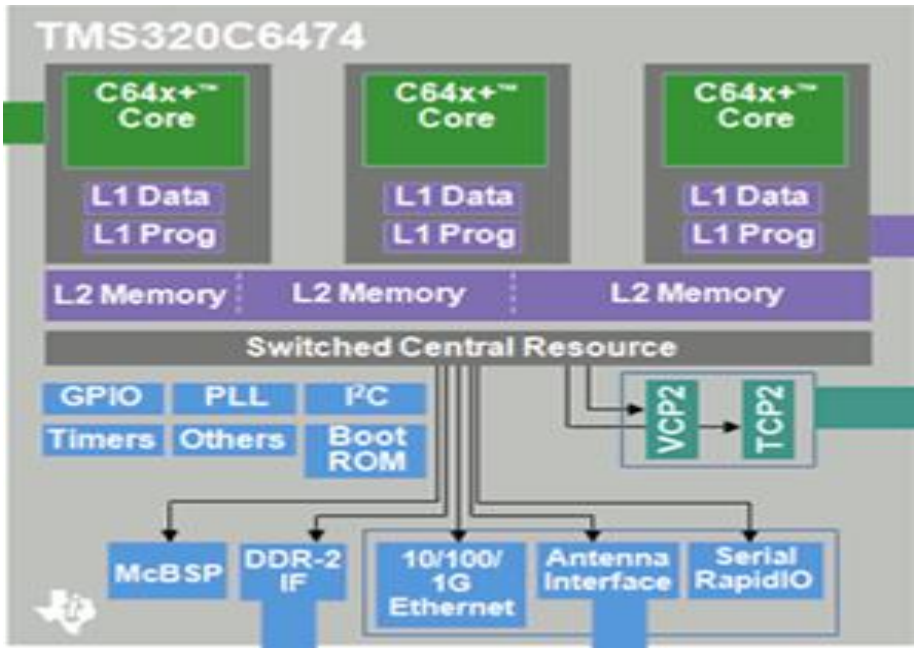


Figure 1: schéma fonctionnel du cœur DSP [3]

1. Mémoire DDR2

Le contrôleur de mémoire DDR2 est la principale interface externe que l'on peut utiliser pour les programmes et le stockage des données. Le processeur peut accéder au mémoire DDR2 à travers le centre de ressources commuté (SCR).

La mémoire DDR2 supporte les fonctionnalités suivantes:

- 512M octet d'espace mémoire
- Largeur de bus de données est 16,32bits
- Burst de type séquentiel
- SDRAM auto initialisation
- Mode auto-refresh
- rafraîchissement par ordre de priorité
- Little Endian et Big Endian transfers **[26].**

2. Le Timer

Le TIMER peut être configuré en trois modes soit :

- TIMER 64-bits à usage général : dans ce mode il fonctionne comme un compteur 64-bit et commence à augmenter de 1 à chaque cycle d'horloge d'entrée.
- TIMER 32-bit double : lorsqu'il est configuré comme TIMER double 32-bit, chaque moitié peut fonctionner indépendamment.
- TIMER de surveillance : Le TIMER a une broche d'entrée (TINPL) et une broche de sortie (TOUTL), ils sont contrôlés par Le TIMER Register contrôle (TCR).

3. Serial Rapid IO

Le Serial Rapid IO est utilisé pour la réception des signaux d'antenne à haut débit. Il transmet 10 bits pour 8 bits d'information, ce périphérique est conçu pour être un module esclave externe capable d'agir comme maître dans le système DSP, cela signifie que ce périphérique externe peut pousser des données sur la DSP, selon les besoins, sans avoir à générer une interruption, ceci a plusieurs avantages : Il réduit le nombre total d'interruptions, il réduit le Handshaking (latence) associé aux périphériques en lecture seul, et elle libère le EDMA pour d'autres tâches.

Les Caractéristiques principales du RapidIO comprennent:

- Une flexible architecture permettant la communication paire -to- pair.
- Une robuste communication avec détection d'erreur.
- Une interconnexion à haut débit.
- Une faible puissance **[26].**

4. Antenna Interface

L'Antenna Interface du TMS320C6474 est un périphérique d'interface série à haute vitesse qui prend en charge les transferts de données IQ (In-phase and quadrature data) avec une liaison montante et descendante entre les processeurs DSP. **[27]**

5. Software-Programmable Phase-Locked Loop Controller (PLL)

Le contrôleur PLL offre la modification de signal d'horloge avec facilité de configuration par logiciel. Les sorties de l'horloge sont transmises à la DSP de base, au périphériques et à d'autres modules à l'intérieur de la DSP.

L'entrée de référence de l'horloge de la PLL:

- CLKIN: signal d'entrée de l'oscillateur externe.

La résultante de sortie des horloges de la PLL:

- AUXCLK: horloge interne du signal sortie directement de CLKIN.
- SYSCLK1 à SYSCLK16: chaque sortie a son propre diviseur. **[28]**

6. Ethernet Media Access Controller/Management Data Input/output module

Le module d'EMAC est utilisé sur les dispositifs TMS320C6474 pour déplacer les données entre un dispositif et un centre serveur différent reliés au même réseau, conformément au protocole d'Ethernet.

Les EMAC et les modules de MDIO connectés au DSP par une interface faite sur commande, qui permet la transmission et la réception des données efficaces. Cette interface faite est mentionnée comme le module de commande d'EMAC, et considérée intégrale au périphérique d'EMAC/MDIO.

Les différentes fonctionnalités de l'EMAC sont :

- Des opérations synchrones de 10/100/1000 Mbit.
- Les Little et Big endian.
- Les Contrôles de flux de matériel.
- L'opération duplex de gigabit. **[29]**

7. General-Purpose Input/output (GPIO)

Le périphérique (GPIO) d'usage général d'entrée sortie fournit et consacre des broches d'usage général qui peut être configurées en tant que des entrées ou sorties. Une fois configuré comme sortie, on peut écrire sur un registre interne ; ou configuré comme entrée, on peut détecter l'état de l'entrée en lisant l'état d'un registre interne.

En outre, le périphérique de GPIO peut produire des événements d'interruptions d'unité centrale de traitement et de synchronisation de l'EDMA dans différents modes de génération interrupt/event. **[30]**

8. Inter-Integrated Circuit Module (I2C)

Le module d'I2C fournit une interface entre le dispositif C6474 et d'autres dispositifs et reliés par un I2C-bus. Les composants externes fixés à ce bus à 2 fils peuvent transmettre et recevoir jusqu'aux 8 bits de données.

Le module d'I2C a les caractéristiques suivantes :

- Transfert de format de 2 à 7 bits.
- Données de format libre.
- On lit et on écrit l'événement de DMA qui peut être employé par cette dernière.
- Sept interruptions qui peuvent être employées par l'unité centrale de traitement. **[31]**

9. Bootloader

Le Bootloader est le code de DSP qui transfère le code d'application à partir d'une source extérieure dans la mémoire interne ou externe de programme après que le DSP soit pris hors de la reset. Le Bootloader est stocké de manière permanente dans la ROM du DSP,

commençant à l'adresse de byte 0x3C000000, le Bootloader offre une série de méthodes pour transférer l'application dans la mémoire de DSP.

Le dispositif C6474 contient trois noyaux de C64x+. Pendant le démarrage, le core0 commence à s'exécuter et maintient le core1 et le core2 dans la reset. Après le reset, le core0 commence à s'exécuter de l'adresse de base de ROM L3 qui est responsable d'effectuer le processus de démarrage, après le démarrage du core0 apporte les autres cores de C64x+ hors de la reset **[32].**

10. Turbo Decoder Coprocessor2 (TCP2)

Le décodage des voies de transmission de données de débit binaire élevées trouvées dans la (3G) des normes cellulaires, exige le décodage des données turbo-codées. Le coprocesseur de turbo-décodeur (TCP2) a été conçu pour effectuer cette opération pour des normes de la radio IS2000 et 3GPP.

Le TCP2 fournit :

- Une haute performance :
  - ✓ le traitement du retard peut être encore réduit en permettant des critères d'arrêt d'algorithme tout en réalisant l'exécution optimale de JUJUBES.
  - ✓ TCP2 et le DSP peuvent courir à toute vitesse en parallèle.
- Une optimisation du coût de système :
  - ✓ La communication entre le DSP et le TCP2 est exécutée par une haute performance, DMA augmenté (EDMA3).
  - ✓ TCP2 emploie ses propres mémoires optimisées et rapporte une exécution globale plus élevée.
- Une amélioration au-dessus de TCP :
  - ✓ Réduction de Prolog.
  - ✓ Contrôle par redondance cyclique.
  - ✓ Commande programmable de décision pour MSB ou LSB **[33].**

11. Viterbi-Decoder Coprocessor 2 (VCP2)

Le décodage du canal de la voie de transmission des données binaires trouvées dans les normes cellulaires telles que 2.5G, 3G, et WiMAX exige le décodage des données. Le coprocesseur2 VCP2 de Viterbe les décode pour des normes de la radio IS2000 et 3GPP, fait également la correction d'erreur pour les systèmes 2G et 3G sans fil, et offre une solution très rentable.

Le VCP2 fournit :

- Une flexibilité élevée :
  - ✓ Longueur variable, K = 5, 6, 7, 8, ou 9
  - ✓ Coefficients écrits par l'utilisateur

- ✓ Codez les taux (1/2, 1/3, ou 1/4)

- Une optimisation de système :
  - ✓ Le VCP2 libère des ressources de DSP pour d'autres traitements.
  - ✓ La communication entre le DSP et le VCP2 est exécutée par le moteur EDMA3 à rendement élevé.
  - ✓ Utilisations optimisées de ses propres mémoires temporaires.
  - ✓ Des bibliothèques sont données pour l'élaboration d'un temps réduit **[34].**

12. Multichannel Buffered Serial Port (McBSP)

En se reliant avec les dispositifs externes, le McBSP assure avec les sept bornes la transmission et la réception des données en communicant aux dispositifs externes.

Le dispositif DSP C6474 communique au McBSP en utilisant deux bus séparés. La commande est accédée par l'unité centrale de traitement, le bus de configuration enregistre les données et l'EDMA utilise son bus consacré pour accéder aux registres **[35].**

Annexe 3:Préparation de l'environnement de travail

Cette annexe est en guise de guide d'installation, du lancement et de configuration du Code composer studio.

1. Utilitaires d'installation du Code Composer Studio

Cette section fournit des informations sur la façon de définir et mettre en place la configuration de votre cible à la fois unique et les configurations multi-processeur, et la façon de personnaliser plusieurs options générales IDE.

a. Ajout d'une configuration existante

Vous devez sélectionner une configuration dans le programme d'installation avant de lancer le Code Composer Studio IDE. Vous pouvez créer une configuration utilisant les fichiers de configuration standard, ou de créer une configuration en utilisant vos propres fichiers de configuration.
Pour créer un système de configuration en utilisant un fichier de configuration:

1. Double-cliquez sur l'icône Setup CC Studio. Le système de configuration de boîte de dialogue apparaît.

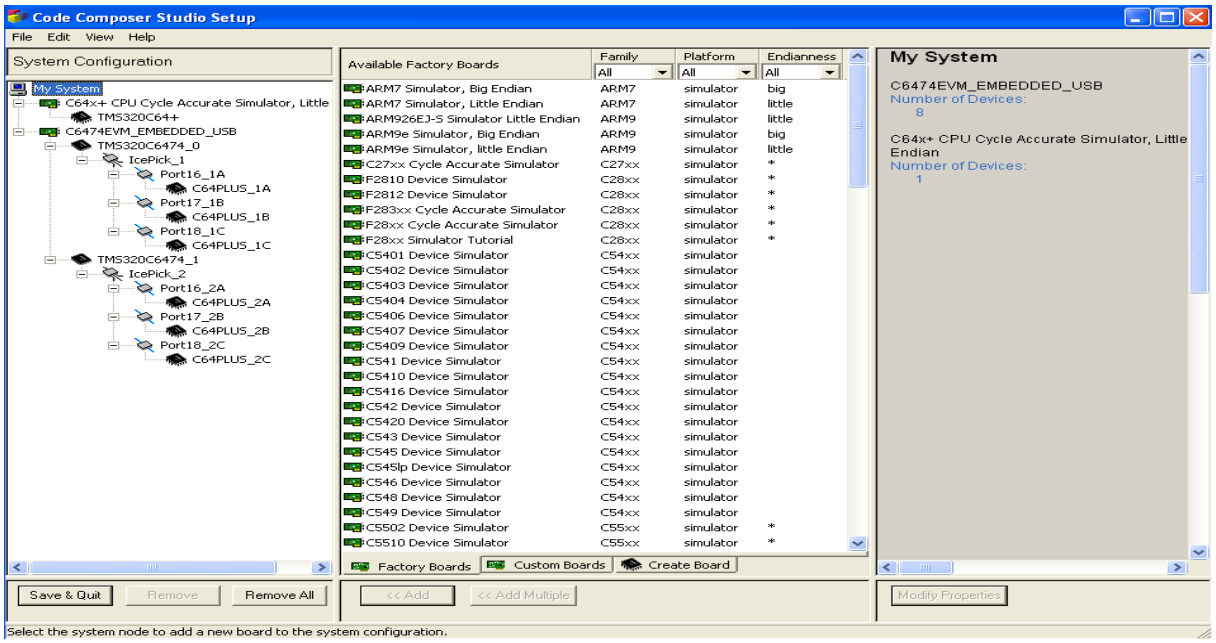


Figure 2: Setup CCS

2. Dans la liste *Available Factory Boards*, sélectionnez la configuration standard, qui correspond à votre système, déterminez si l'une des configurations disponibles correspondant à votre système, si aucun n'est suffisant, vous pouvez créer une configuration personnalisée.

3. Cliquez sur le bouton Add pour faire l'importation de la configuration du système en cours de création. La configuration que vous avez sélectionnée s'affiche maintenant sous Mon My System icon dans le panneau de configuration système de la fenêtre d'installation.

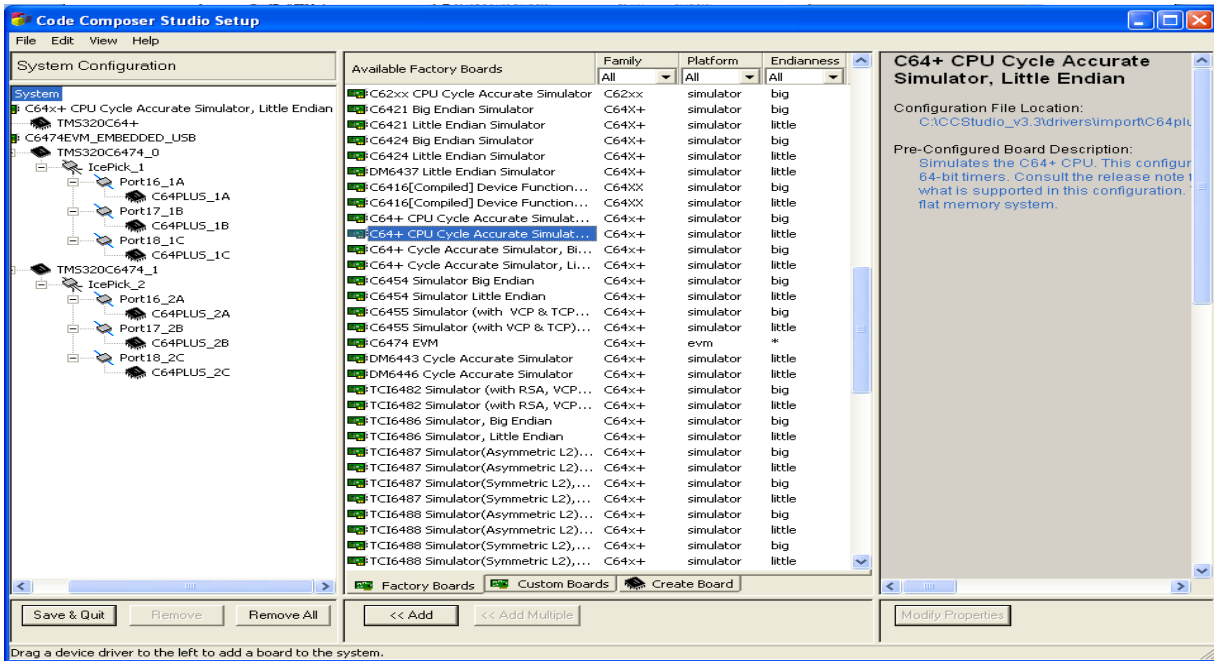


Figure 3: Le choix d'une configuration existante en *Available Factory Boards*

4. Cliquez sur le bouton Enregistrer et Quitter pour enregistrer la configuration.

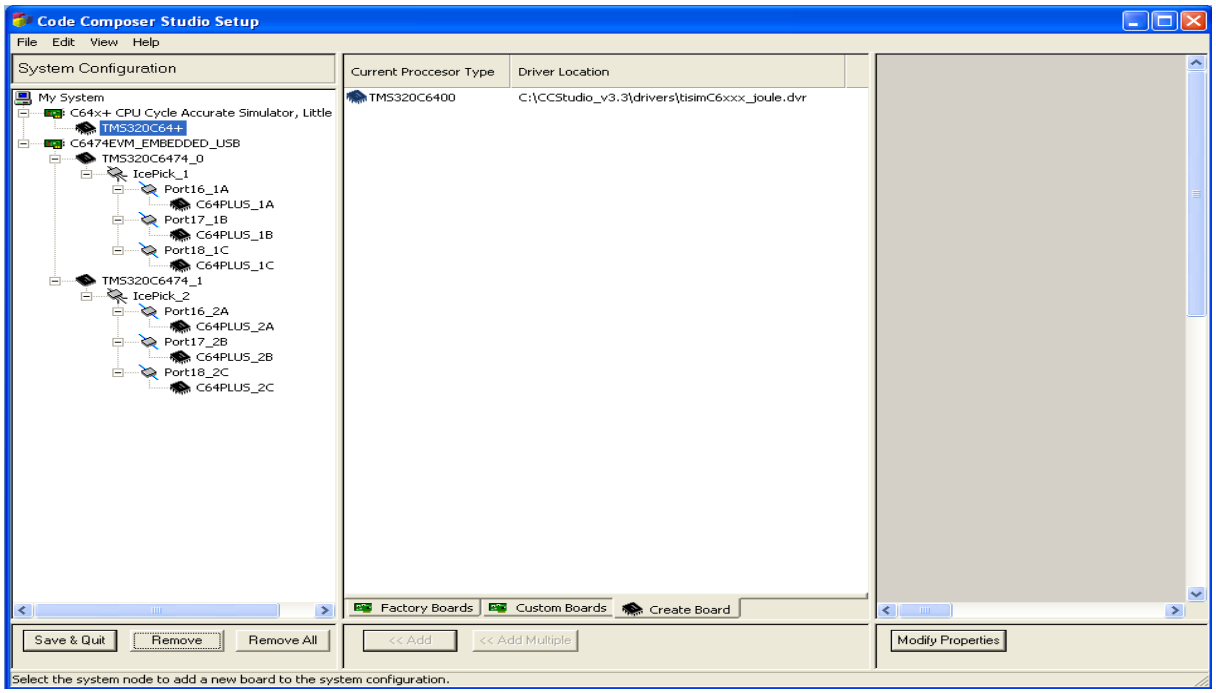


Figure 4: Le regroupement du choix de configuration

5. Cliquez sur le bouton oui pour démarrer le Code Composer Studio IDE avec votre configuration. Vous pouvez maintenant commencer sur un projet.

b. Parallel Debug Manager

Dans les configurations multiprocesseurs, en invoquant Code Composer Studio lance un contrôle spécial, connu sous le nom de Parallel Debug Manager Plus (PDM +).

Le parallèle Debug Manager vous permet d'ouvrir un Code Composer Studio IDE, pour chaque session périphérique cible. Activité sur les appareils spécifiés peut être contrôlée en parallèle à l'aide du contrôle de PDM.

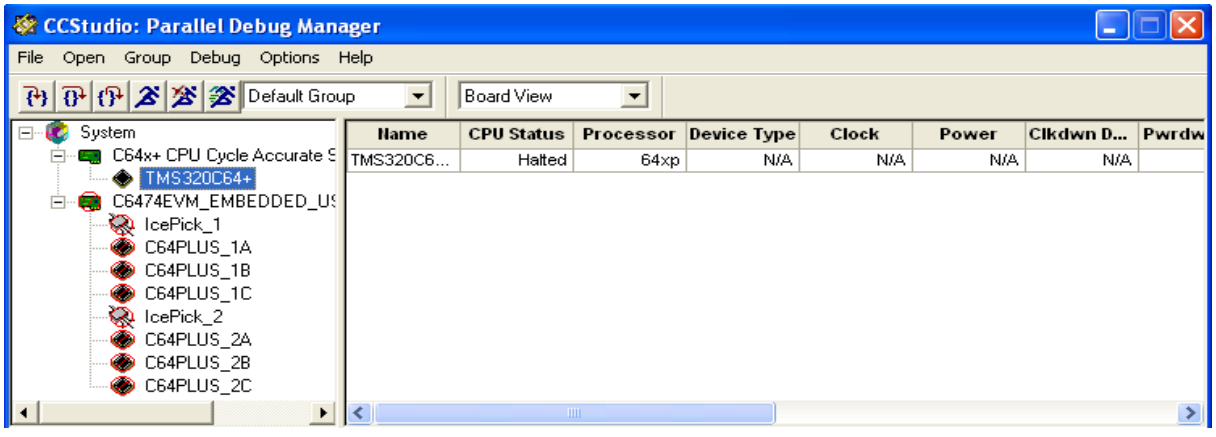


Figure 5: Parallel Debug Manager de TCI6474 EVM CC Studio v3.3

Cette version de Parallel Debug Manager (PDM +) inclut les caractéristiques suivantes:

Vous pouvez connecter ou déconnecter de cibles à la volée par un clic droit sur le processeur sur le panneau droit.

L'interface permet une vue élargie de processeurs, avec plusieurs filtres de liste déroulante pour afficher une liste de groupe, par le CPU ou par le conseil.

L'icône rouge sur le processeur (sur le volet de gauche) indique que le processeur n'est pas connecté au système, ou qu'il a mis à jour les informations d'état.

c. Connexion / Déconnexion

Code Composer Studio IDE, il est facile à connecter et déconnecter dynamiquement avec la cible à l'aide de l'anglet Connect/Disconnect. Ainsi vous permet de déconnecter votre matériel objectif et même de restaurer l'état précédent de débogage lors de la reconnexion.

Par défaut, Code Composer Studio IDE ne tentera pas de se connecter à la cible lors de la fenêtre de contrôle est ouvert. Lorsque vous êtes connecté, la barre d'état indique également si l'objectif est connecté ou non.

Après une connexion à la cible, une option de menu intitulé Restore Debug State sera disponible sous le menu Debug.

Si le parallèle Debug Manager est ouvert, vous pouvez vous connecter à une cible par un clic droit sur la cellule correspondant en dessous de la colonne Name.

2. Introduction à la plateforme Code Composer Studio de TI

Figure 6: Fenêtre de démarrage du CCS

a. Lancement du Code Composer Studio

Pour lancer Code Composer Studio IDE pour une première fois, cliquez sur l'icône qui est sur votre bureau. Un simulateur est automatiquement configuré par défaut se lance.

La fenêtre suivante s'ouvre si la carte de développement n'est pas branchée. Cliquer sur Ignorer. Cela fait apparaître la fenêtre « Code Composer Studio» pour faire les préparations, mais c'est important que la carte soit branchée.

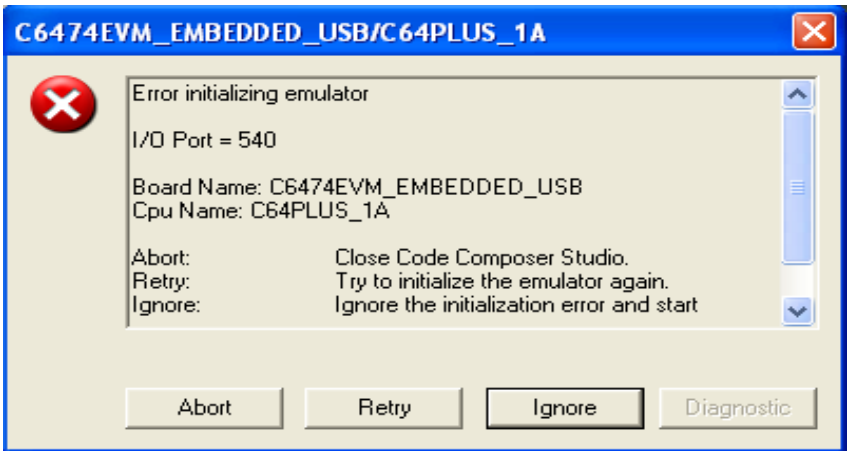


Figure 7: Message d'erreur d'initialisation de l'émulateur

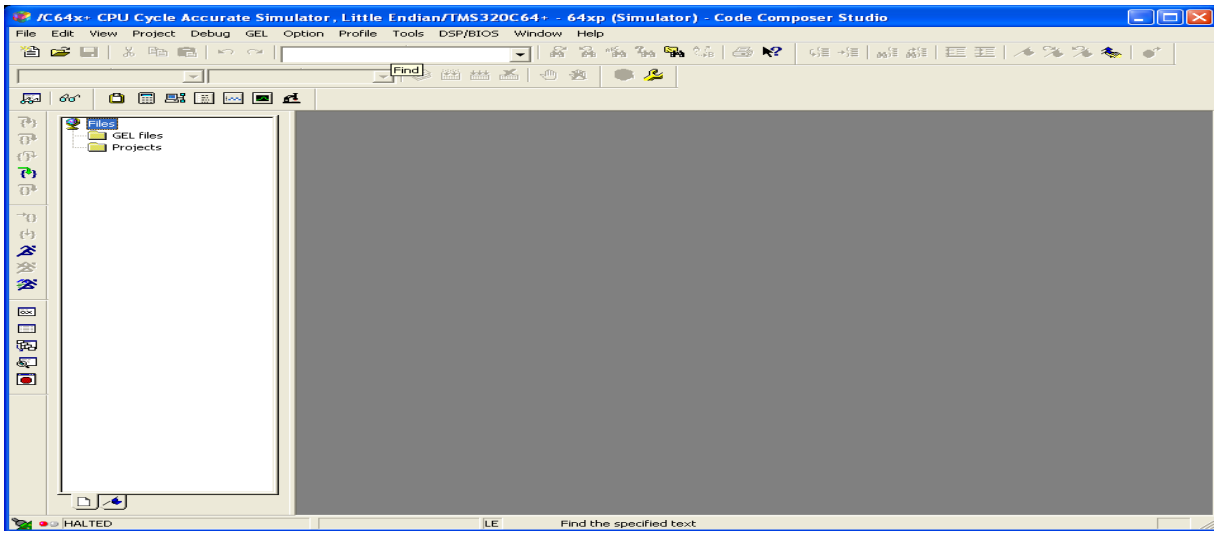


Figure 8: Interface graphique de CCS

b. Création et sauvegarde d'un nouveau fichier

b.1. Création d'un nouveau fichier

CCS fournit un éditeur intégré qui permet la création des fichiers sources. Pour créer un fichier, choisir l'élément de «File --> New--> Source File» de la barre d'outils. Cela fait apparaître la fenêtre de dialogue « Untitled1 » comme illustré dans la Figure 9.

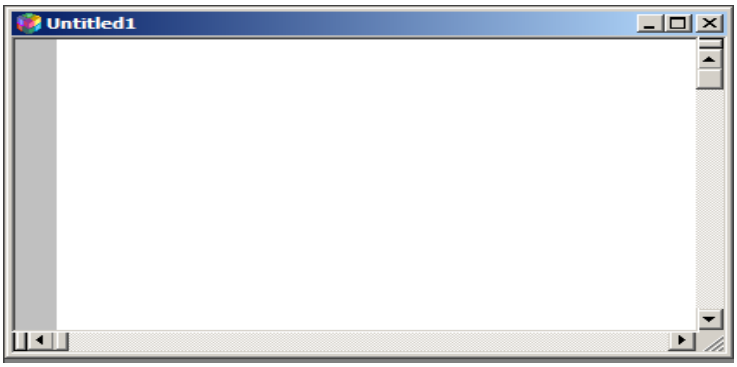


Figure 9 : nouveau fichier source

b.2. Sauvegarde d'un fichier

Vous sauvegardez le fichier source créé en choisissant l'élément de menu « File→Save » ce qui fait apparaître la fenêtre de dialogue « Enregistrer sous » comme illustré dans Figure 6. Dans cette fenêtre, allez à la zone « Type » et choisissez le type de fichier à partir de la liste déroulante. Ensuite, allez à la zone « Nom du fichier » et écrivez le nom du programme. Finalement, en cliquant sur « Enregistrer », le code est sauvegardé dans le fichier source spécifié. Dans le répertoire «C:\ti\examples\exemple».

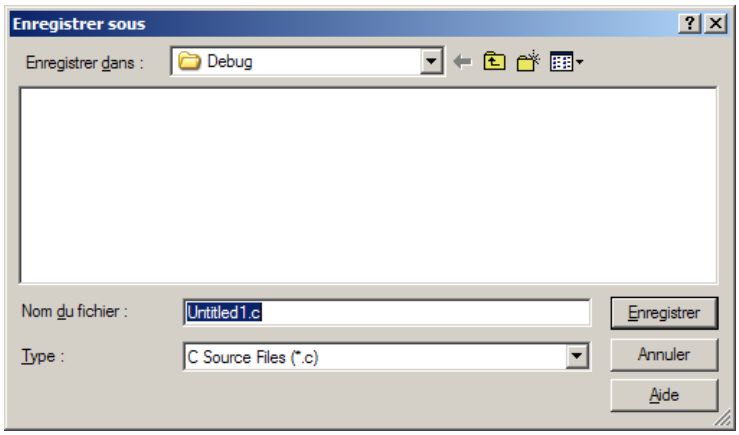


Figure 10: La fenêtre de dialogue « Enregistrer sous »

b.3. Création d'un nouveau projet

Pour créer un projet, suivez ces étapes:

1. Si vous avez installé Code Composer Studio dans C: \ CCStudio_v3.3, créez un dossier nommé dans la pratique C: \ CCStudio_v3.3 \ myprojects dossier.
2. Copiez le contenu de C: \ CCStudio_v3.3 \ tutorial \ target \
3 .Lancement Code Composer Studio.
4. À partir du menu Projet, choisissez Nouveau.
5. Dans le champ Nom du projet, tapez le nom du projet.

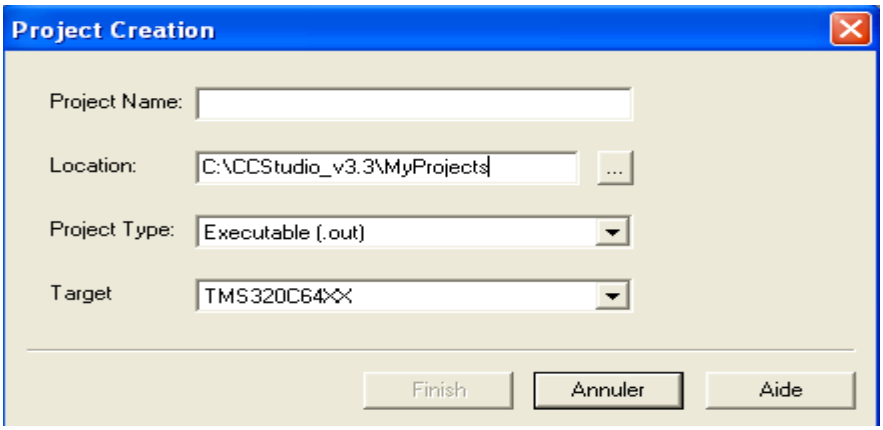


Figure 11: Menu de création d'un nouveau projet

6. Dans le champ Emplacement, tapez ou recherchez le dossier que vous avez créé à l'étape 1.
7. Cliquez sur Terminer. Code Composer Studio crée un fichier de projet appelé practice.pjt. Ce fichier stocke vos paramètres du projet et des références des différents fichiers utilisés par votre projet.
8. Ajouter des fichiers sur le projet en choisissant Ajouter des fichiers de projet dans le menu Projet. Vous pouvez aussi cliquez avec le bouton droit sur la fenêtre de projet sur la gauche, puis sélectionnez ajouter des fichiers au projet.

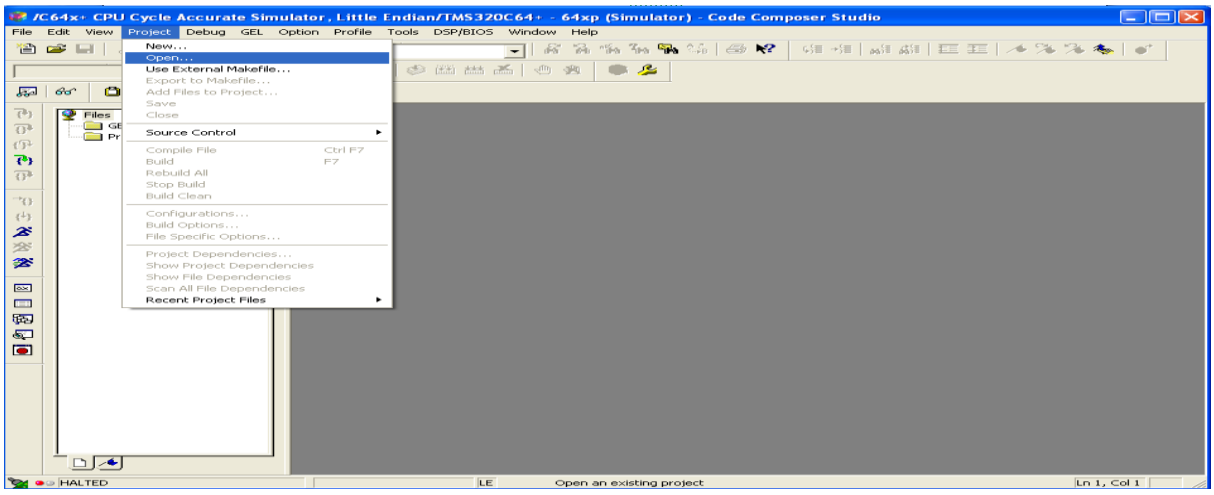


Figure 12 : ouverture d'un projet CCS

9. Vous n'avez pas besoin d'ajouter manuellement tous les fichiers à votre projet, parce que le programme trouve les automatiquement.

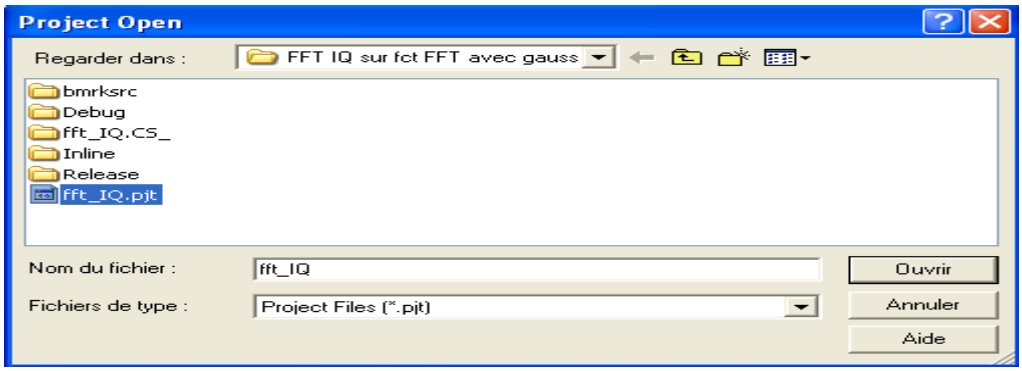


Figure 13: Menu d'ouverture d'un projet de test

Créer un fichier de commande pour l'éditeur de liens :

En plus des fichiers sources, un fichier de commande d'éditeur de liens doit être indiqué pour créer un fichier exécutable et pour se conformer aux spécifications de mémoire du DSP et de la cible sur laquelle le fichier exécutable va être compilé. Un fichier de commande d'éditeur de liens peut être créé en utilisant la même procédure que pour le fichier précédant en choisissant « File→New→Source ».

Pour plus de détail sur le fichier de commande et la configuration du Mémoire, consultez l'Annexe 4.

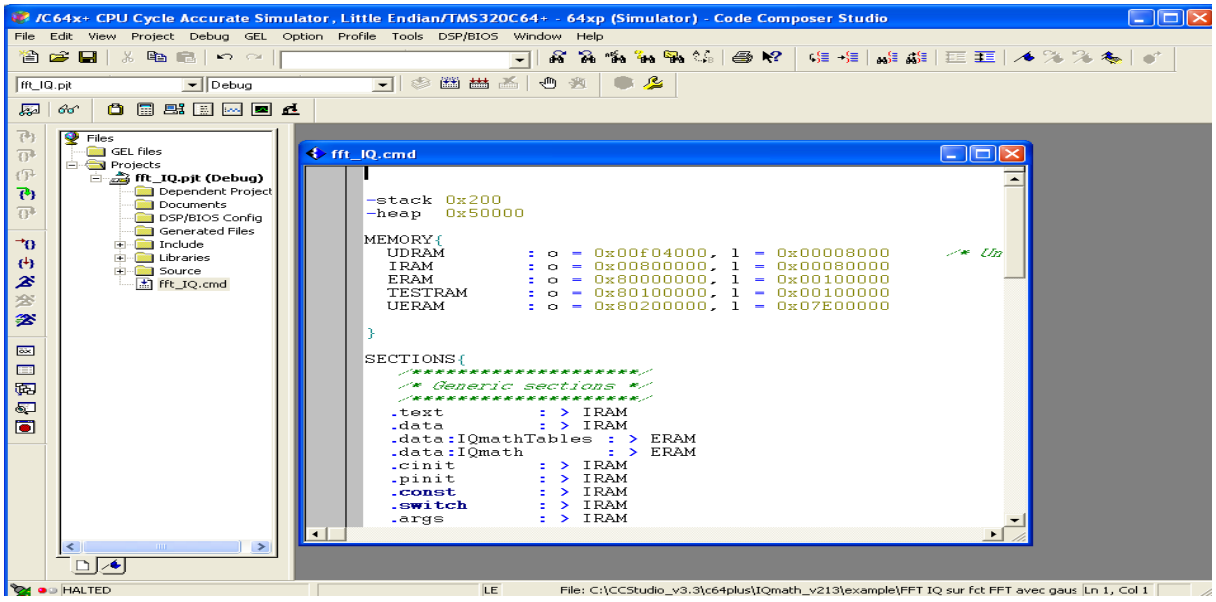


Figure 14: Fichier de commande du projet de test.

b.4. Construction de votre programme

Après avoir ajouté tous les fichiers sources, fichier de commande et fichier de bibliothèque au projet, vous pouvez construire le projet et créer un fichier exécutable. Pour faire cela, choisissez l'élément de menu «Project→Build». Cette fonction permet de compiler, assembler, et joindre tous les fichiers dans le projet.

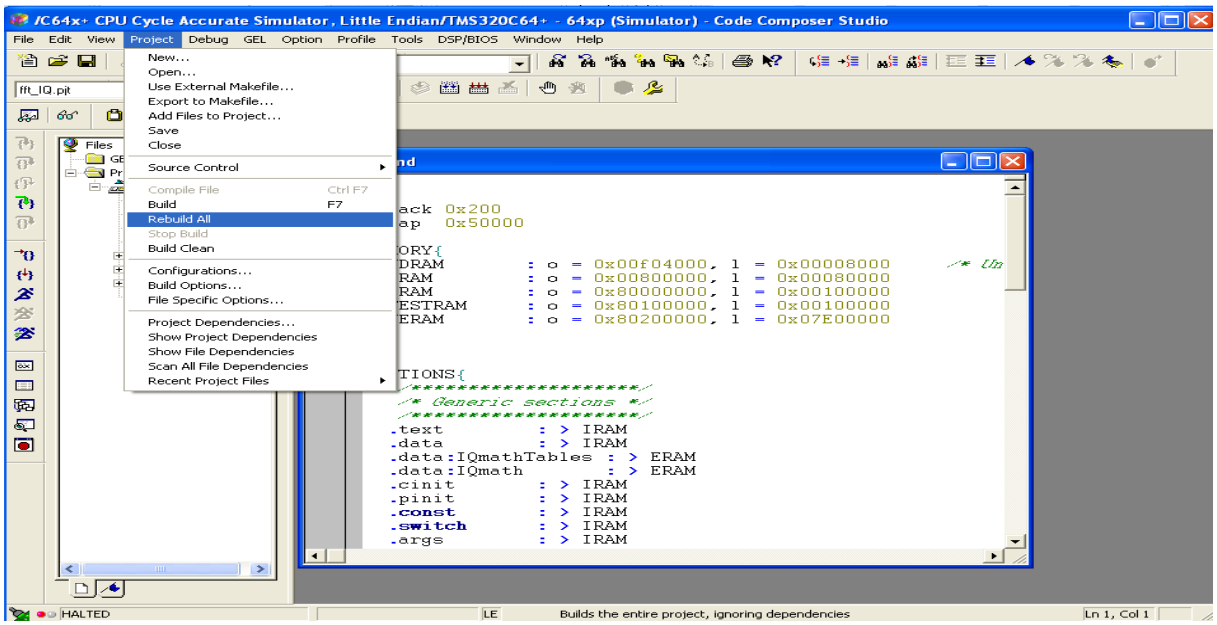


Figure 15: Compilation du projet de test

Si le processus de construction est complété sans erreur, le fichier exécutable «exemple.out» est produit.

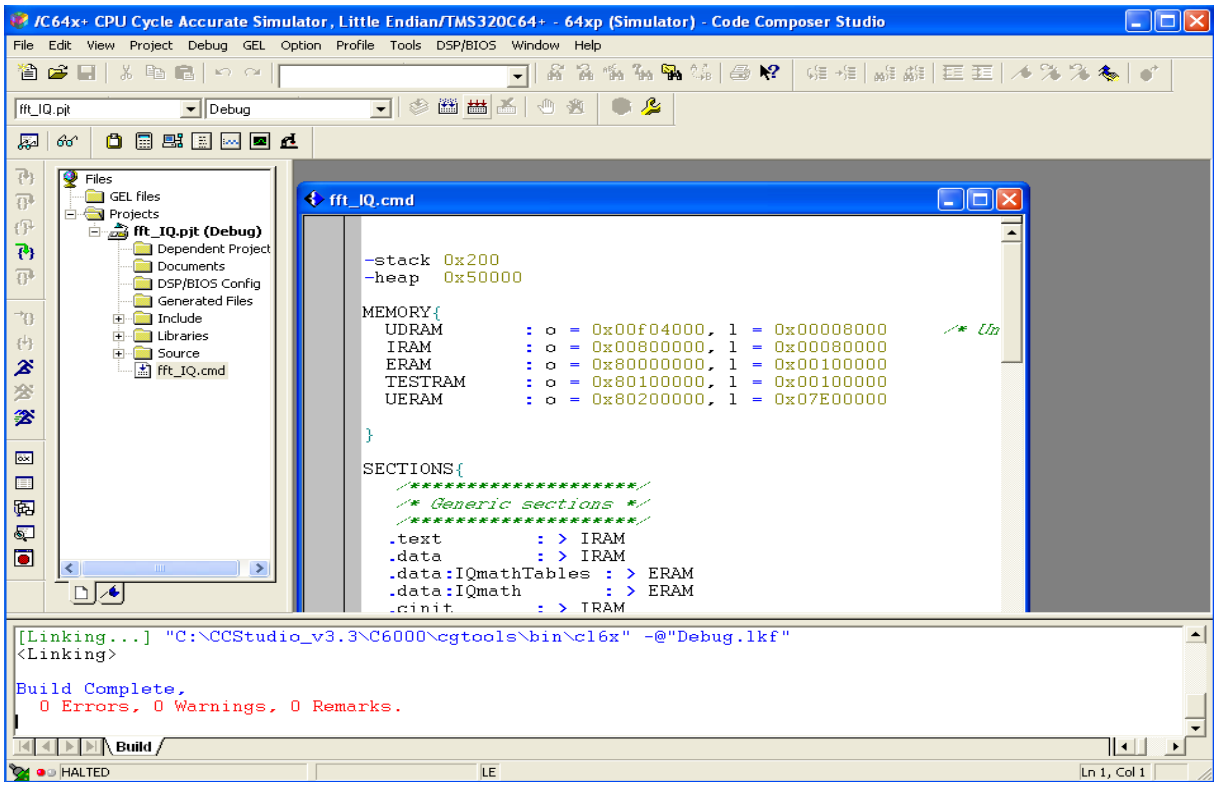


Figure 16: Affichage de messages dans CCS après compilation

Il est également possible de compléter les constructions par incréments ; c'est-à-dire en recompilant ou en assemblant seulement des fichiers changés depuis la dernière construction, en choisissant l'élément de menu « Project→Rebuild ».

CCS fournit des options par défaut, ces options peuvent être changées en choisissant « Project→ Build Options ». Pour changer le nom du fichier exécutable en test.out, choisissez « Project→ Build Options », et cliquer sur « linker» cela fait apparaître la fenêtre « Build Options for Exemple.pjt » comme illustré dans Figure 8.

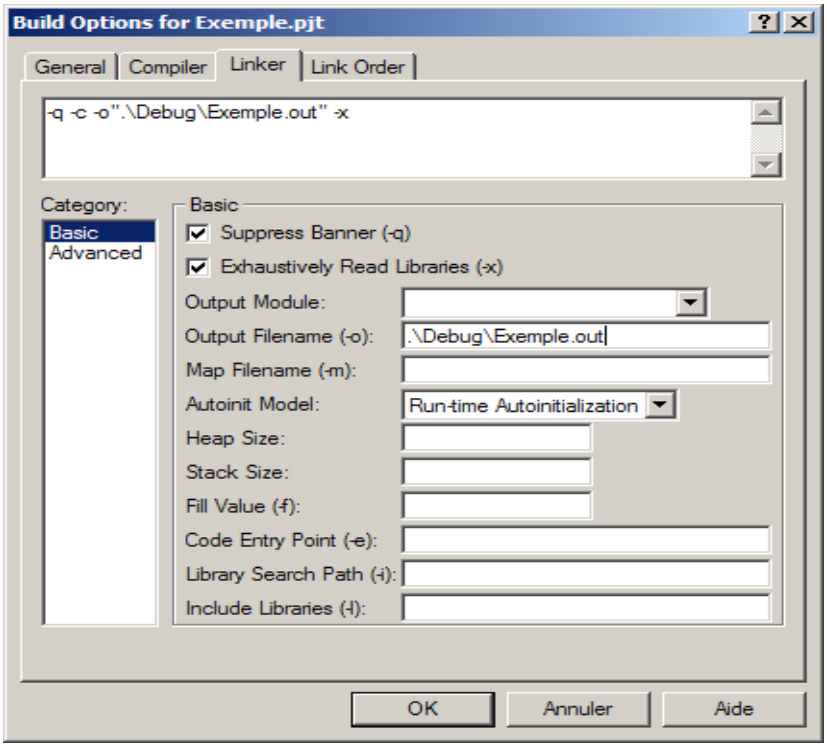


Figure 17: La fenêtre des options de compilations

Dans cette fenêtre parmi les options de construction ; taper alors test.out dans la zone «Output Filename ». Noter que le fichier de commande d'éditeur de liens inclura test.out quand vous cliquerez sur cette fenêtre.

b.5. Chargement du programme

Pour le chargement du programme, vous choisissez l'élément de menu File ->Load Program

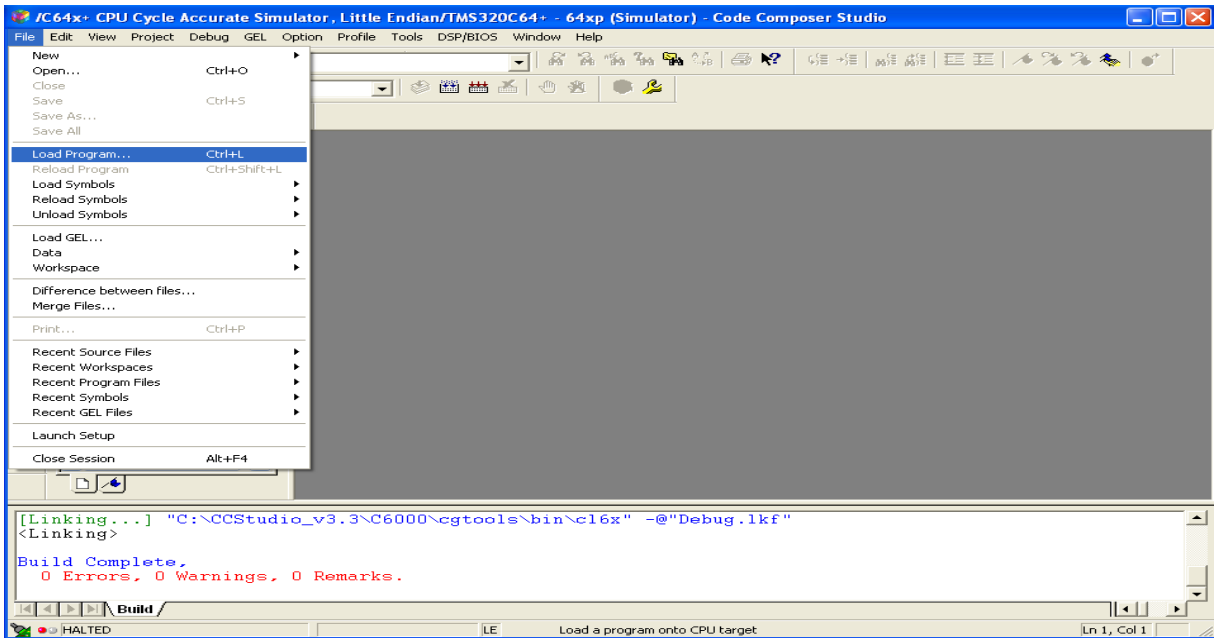


Figure 18: Chargement de l'exécutable

b.6. Exécution du programme

Pour l'exécution du programme, choisir l'élément de menu Debug->Run ou appuyer sur F5

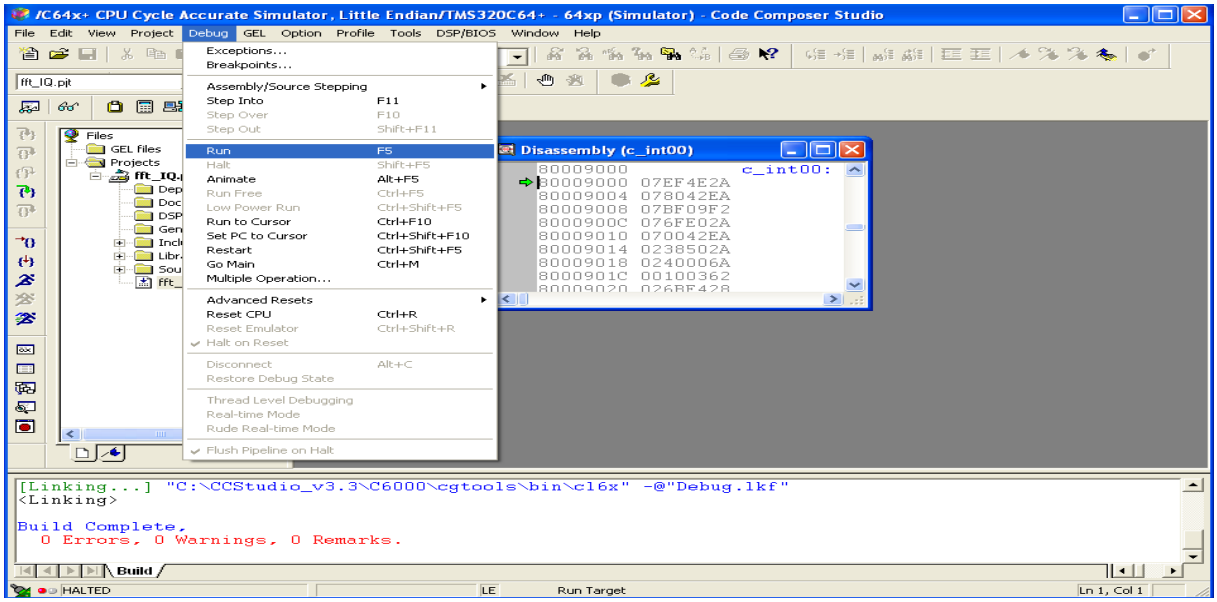


Figure 19: Exécution du programme

c. Utilisation de Breakpoint

Durant le développement ou le test des programmes, en doit souvent contrôler la valeur d'une variable pendant l'exécution du programme. Ceci peut être réalisé en utilisant des points d'arrêt BREAKPOINT et des « Watch Windows »Pour définir un point d'arrêt, placez le curseur sur la ligne désirée et appuyez sur F9. En outre, vous pouvez régler la

breakpoint en sélectionnant le bouton Toggle Breakpoint. Quand un point d'arrêt a été fixé, une icône rouge apparaîtra dans la marge de sélection. Pour supprimer le point d'arrêt, appuyez simplement sur la touche F9. Vous pouvez également ouvrir le Gestionnaire d'arrêt (Debug->Breakpoints) pour afficher tous les arrêts.

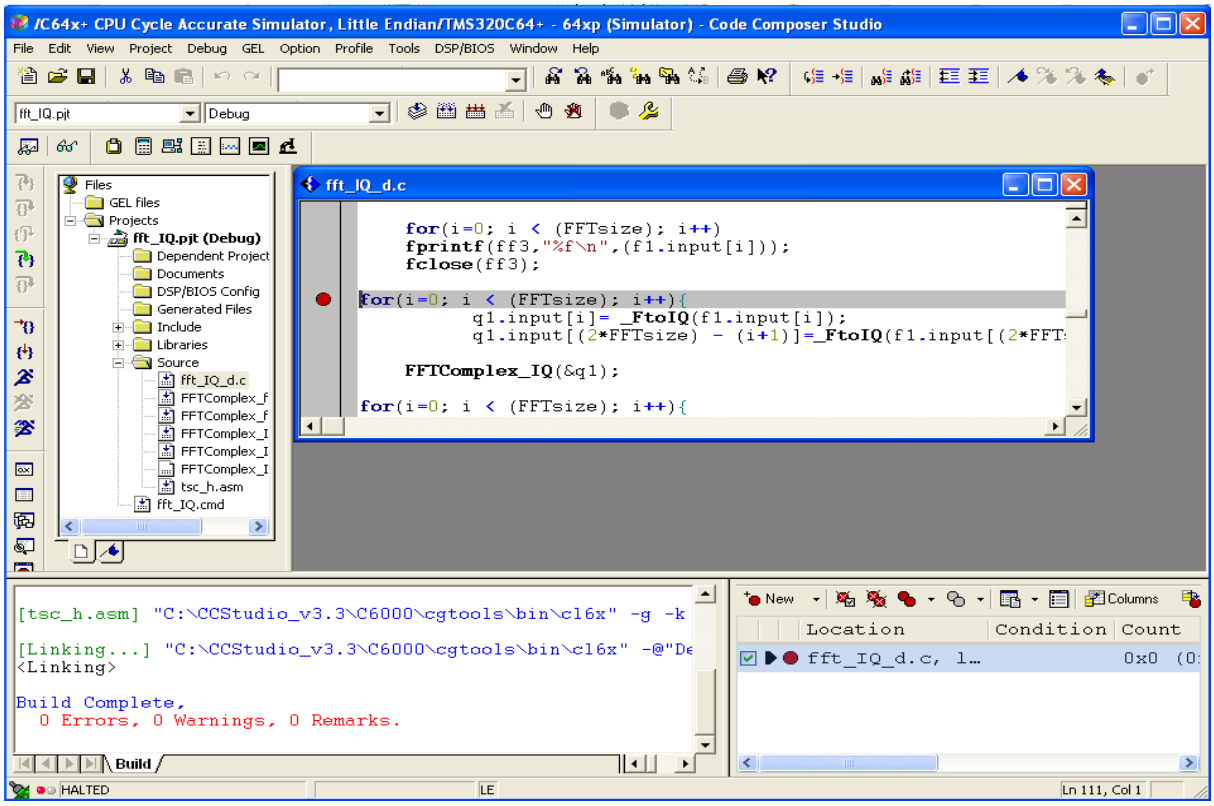


Figure 20: Utilisation de Breakpoint

d. L'outil d'affichage graphique

CCS permet de l'affichage graphique du contenu de la mémoire à un emplacement spécifique. Pour l'affichage graphique du contenu de la mémoire à l'adresse 80000000, choisissez « View→Graph→Tme/Frequency » à partir du menu. La fenêtre de dialogue «Graph Property Display options» i apparaîtra. Remplissez la avec les valeurs voulues, cliquer sur OK. Cela fait apparaître la fenêtre « Graphical Display » comme illustré dans Figure Z.

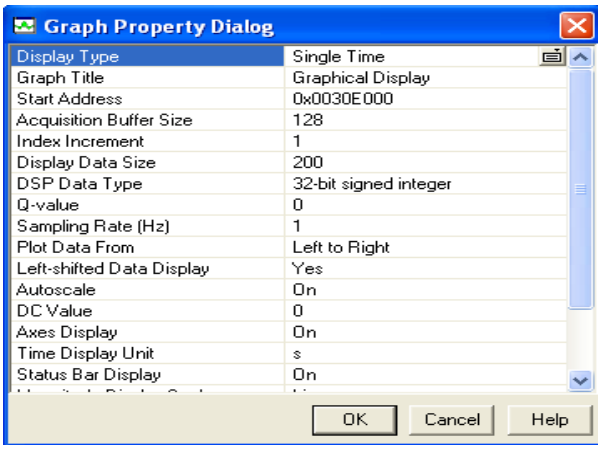


Figure 21 : La fenêtre de propriétés de graphe

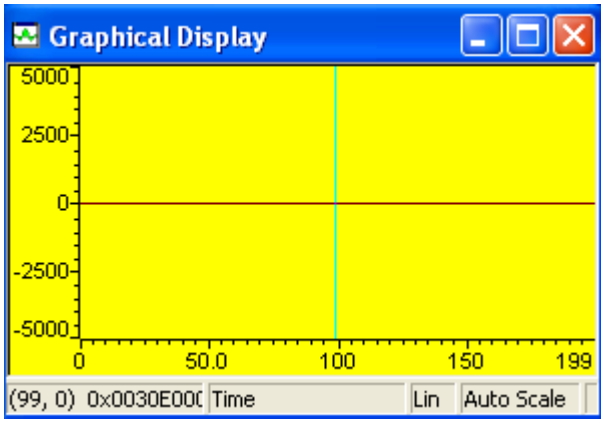


Figure 22: La fenêtre d'affichage de graphe

Annexe 4 : Configuration, compilation et exécution de Linux sur EVM6474

Configuration et compilation du noyau Linux

1 – Créer un répertoire de compilation. Ce répertoire sera noté *<build_dir>* dans la suite de ce document. On suppose qu'on a l'arborescence suivante :

<install_dir>

|___*<dvd_dir>*

|___*<build_dir>*

Où *<dvd_dir>* est le répertoire où est téléchargé le DVD de Texas Instruments contenant les outils de compilation du noyau linux. *<install_dir>* est donc le répertoire de Cross Compilation.

2 – Sélectionner la carte cible de compilation. Le paramètre *<BOARD>* peut prendre les valeurs suivantes :

evm6488-femto, evm6488, evm6488be, evm6482be, evm6482, evm6486be, evm6486, evmc6424, dvevm6446, evmdm6437 or evmdm648.

Dans la suite de ce document *<BOARD>* sera *evm6488*.

3 – Exécuter les commandes suivantes:

$ cd <build_dir>

$ sh <install_dir>/tools-linux/configure \

-p <install_dir>/src/linux-2.6.13/profiles/<BOARD>-linuxonly-nommu \

-b <install_dir>/target-c6x

(Pour info uniquement : pour le big-endian, taper *<install_dir>/target-c6xeb* au lieu de *<install_dir>/target-c6x*)

4 - Editer la configuration du noyau linux:

$ make linux.menuconfig

A ce niveau, on peut modifier la configuration par défaut du noyau Linux. En particulier, l'édition de la ligne de commande de Linux (la ligne de commande du noyau Linux) est

nécessaire. Les paramètres actuels sont consultables sous l'entrée du menu 'Processor type and features'.

Pour la carte Femto EVM6488, cette ligne de commande est configurée pour booter à partir du Flash avec YAFFS2. (Référenciée « MTD configuration » dans la documentation)

console=ttyS1,115200 \

emac_addr=00:0e:99:02:be:fe \

mtdparts=evm6488-nand:8M(boot),-(root) \

root=/dev/mtdblock/1 rootfstype=yaffs rw

Il faut changer cette configuration pour booter à partir du NFS (référenciée « NFS configuration » dans la documentation) :

Taper *gedit <build-dir>/ build-LINUXKERNEL/.config*

console=ttyS1,115200 \

emac_addr=00:0e:99:02:be:fe \

mtdparts=evm6488-nand:8M(boot),-(root) \

root=/dev/nfs rw \

nfsroot=10.0.0.1:/export/linux-root,auto_uid,nfsvers=3 \

ip=10.0.0.2:::255.255.255.0::eth0:

Dans cette configuration, l'adresse IP 10.0.0.2 est attribuée à la carte C6x. L'adresse IP du Host Linux (le PC FEDORA) est 10.0.0.1. Changer ces valeurs selon les paramètres LAN du réseau. Il faut aussi préciser le chemin d'accès au système de fichiers du noyau Linux sur le PC Fedora. Remplacer le chemin du *nfsroot /export/linux-root* par *<chemin d'accès au répertoire linux-root se trouvant sous <build_dir>>*. Ce qui suit est un exemple de ligne de commande après configuration :

CONFIG_CMDLINE="console=ttyS0,115200 emac_addr=00:0e:99:02:be:fe root=/dev/nfs nfsroot=192.168.1.4:/home/coralie/linuxKernel/linuxKernelBuild/linux-root,auto_uid,nfsvers=3 ip=192.168.1.1:::255.255.255.0::eth0: "

La sauvegarde de la configuration est recommandée à ce niveau.

5 – Lancer la compilation du noyau Linux:

$ make

$ make linux.modules_install

$ make linuxroot

Configuration du PC FEDORA

1- s'assurer que le SELinux Management est bien configuré comme suit :

System -> Administration -> SELinux Managment

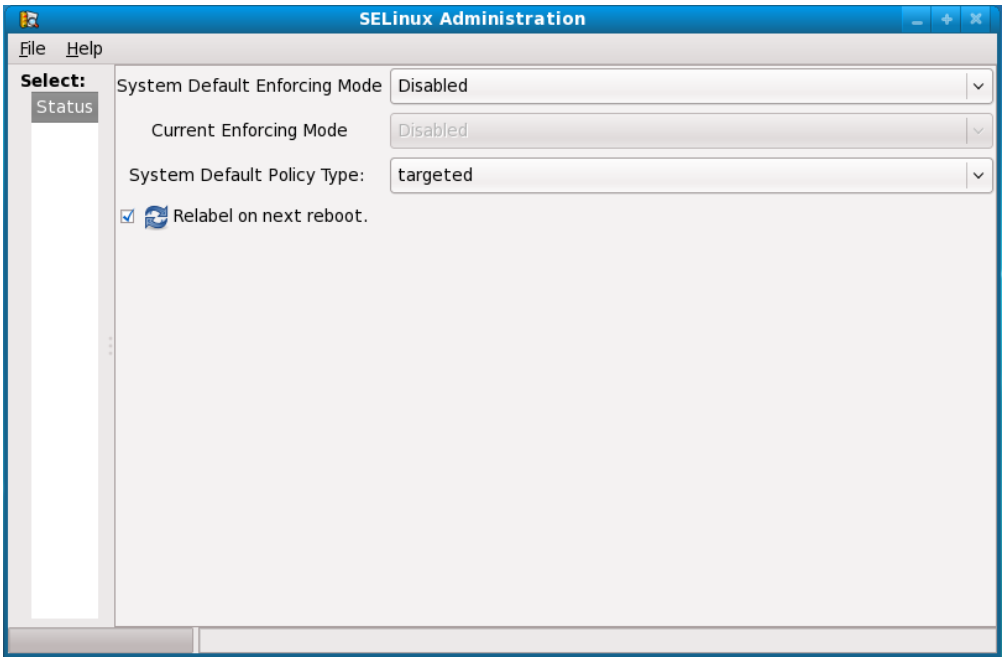


Figure 1 : SELinux Management

2- Pour voir la configuration de la carte réseau, taper */sbin/ifconfig*. Pour regarder une interface ethernet en particulier, passer en argument le nom de l'interface, par exemple *eth0*.

3- Attribuer une adresse IP à l'interface Ethernet. Pour cela, taper

/sbin/ifconfig eth<x> adresse_ip

Afin de vérifier si le lien par réseau Ethernet est bien établi entre deux machines, taper

ping <adresse_ip_autre_machine_sur_reseaux>

Si tout va bien, aucun paquet envoyé ne sera perdu.

4- Désactiver la table de redirection des paquets. Taper
/sbin/service iptables stop

Pour voir l'état de *iptables*, passer l'argument *status* au lieu de *stop*

5- Configurer le serveur NFS. Il faut d'abords préciser les chemins d'accès aux données partagées par le serveur, et le mode d'accès. Ceci est réalisé au niveau du fichier */etc/exports*. Actuellement il est configuré comme suit : */home/coralie *(rw)*

C'est-à-dire que le répertoire */home/coralie*, qui contient, entre autres, le système de fichiers du noyau Linux compilé, est accessible par le serveur NFS sur le PC FEDORA, et que les clients qui s'y connecteront (en particulier le linux sur la carte EVM6474) y accèderont en lecture et écriture.

Dans un deuxième temps le serveur NFS est lancé :

/sbin/service nfs start

Au lieu de *start*, les options *stop*, *restart* et *status* peuvent aussi être utilisées.

Chargement et exécution

Un fichier exécutable « vmlinux » est généré à l'issue de la compilation. Ce fichier ce trouve dans « .../install_dir/build_dir/build-LINUXKERNEL ».

1- Copier ce fichier dans le pc windows.
2- Renommer le fichier en « vmlinux.out »
3- Ouvrir « EVMC6474 CCStudio v3.3 » après avoir mis sous tension la carte et branché le cable USB.
4- Clic droit sur « C64PLUS_1A », clic sur « Connect Device », double clic : CCS est lancé
5- Dans le menu de CCS : File->Load Program, ouvrir « vmlinux.out »
6- View->Memory, dans le champ d'adresse, remplacer « Enter An Address » par « __log_buf »
7- Dans le champ d'encodage spécifier « Character »
8- Lancer l'exécution : Debug->Run

Pour vérifier l'exécution du noyau linux sur la carte :

- Du côté du pc CCS : lire la sortie de linux dans la mémoire (« __log_buf »). Vers la fin du chargement, si le noyau est bien chargé, sera affichée à la sortie des informations indiquant la disponibilité de la console. Sinon, il y aura des messages d'erreur.

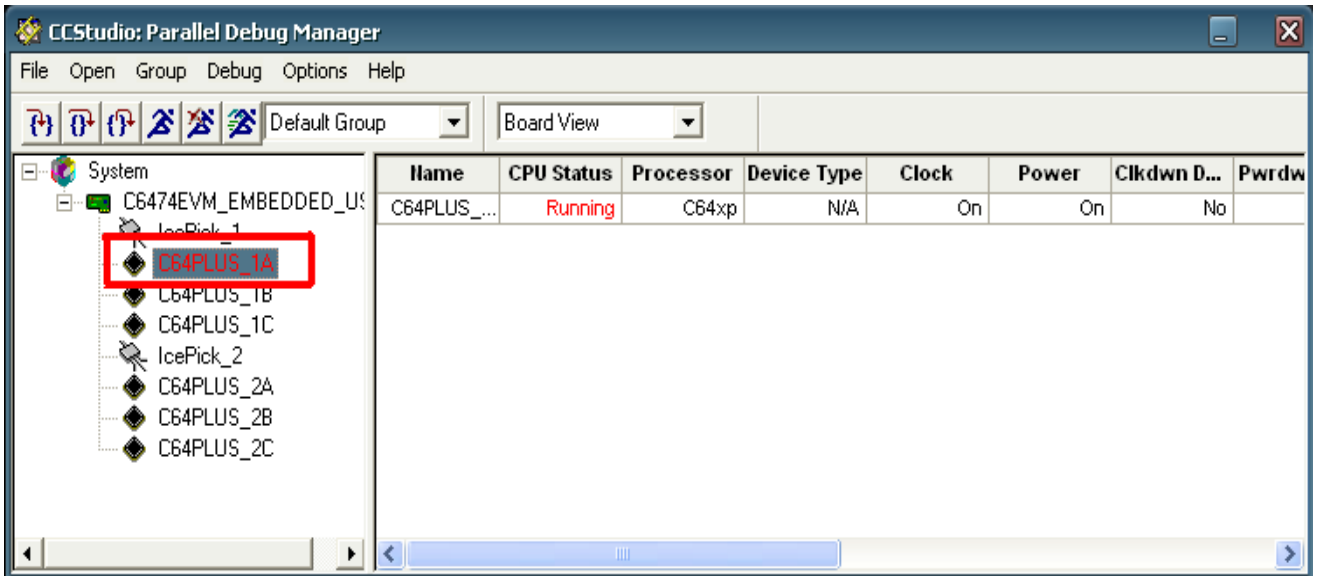


Figure 2 : Management de débogage parallèle

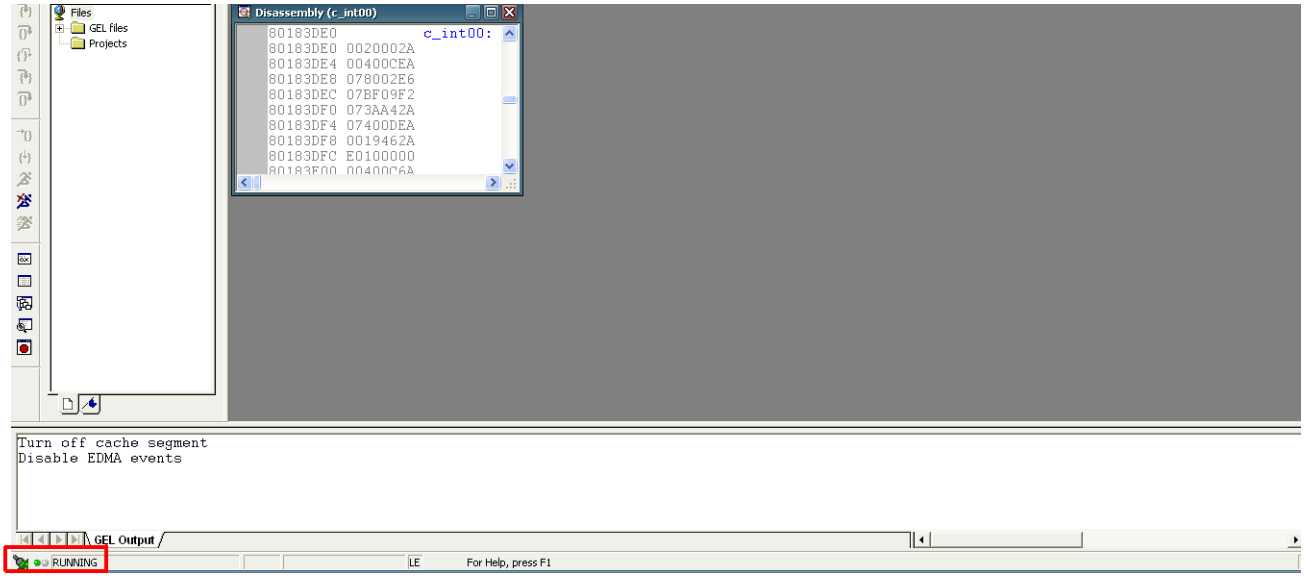


Figure 3 : chargement de Linux

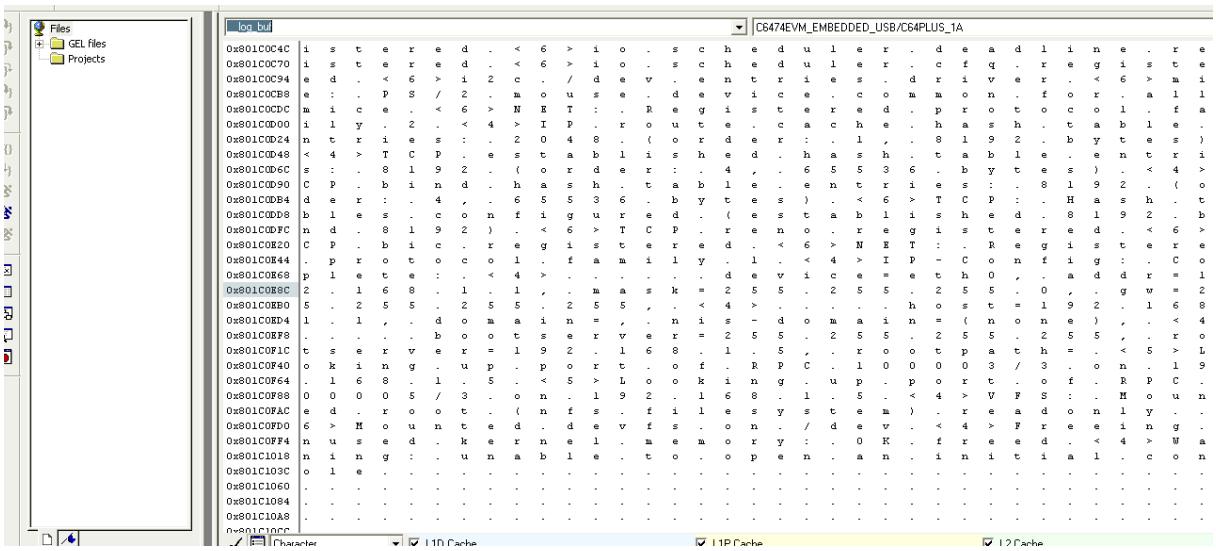


Figure 4 : mémoire système

- Du côté du pc Fedora : taper sur la console la commande telnet en lui passant en argument l'adresse IP de la carte. Si un login est demandé ce sera « root ». Dans ce cas, la compilation et l'exécution sont réussies et vous avez la main sur la console linux tournant sur la carte.

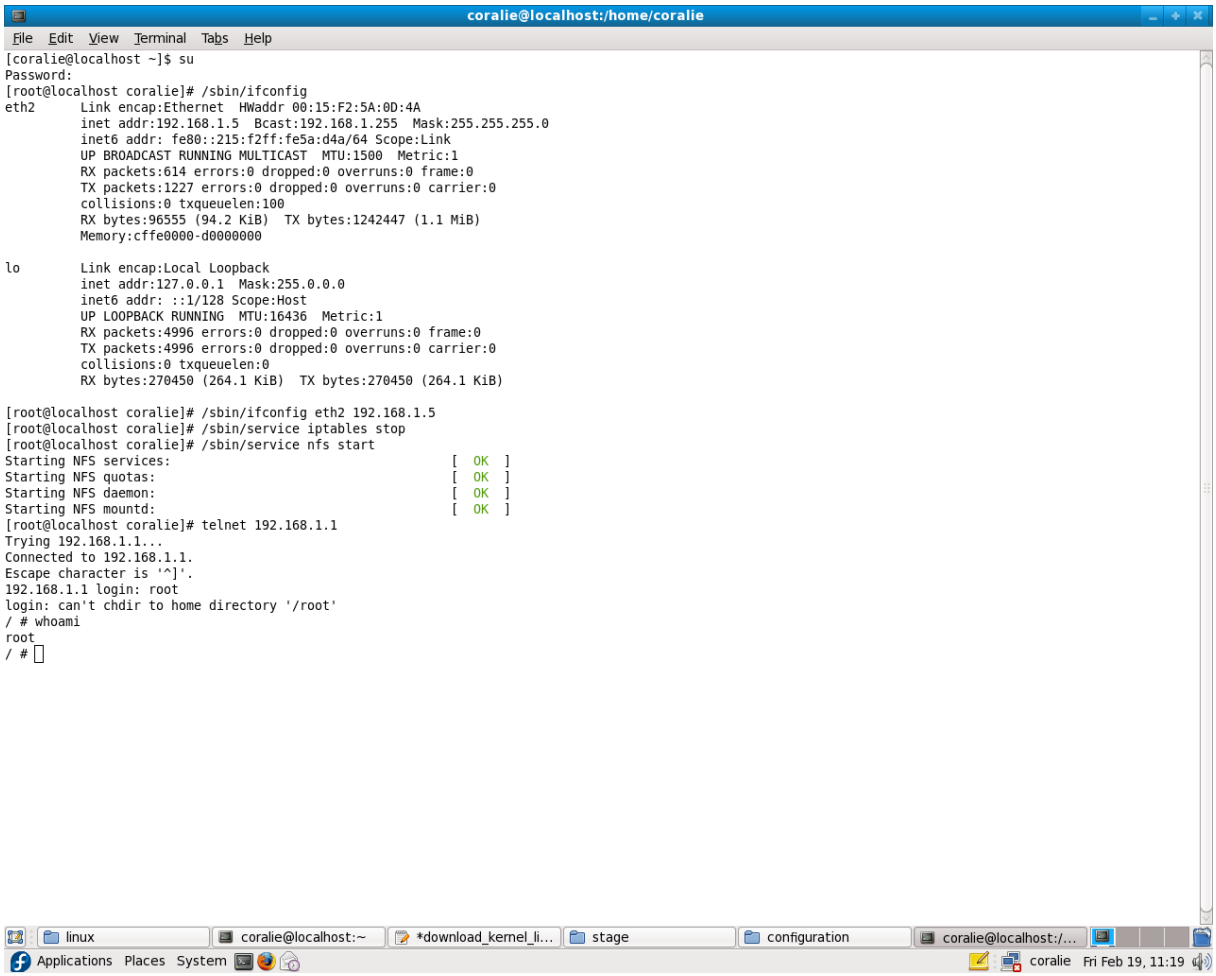


Figure 5 : interface de commande pour accéder à l'EVM6474

- Du côté de la carte d'éval : la led DS clignote et les leds DS11->DS13 sont éteintes. Ceci indique que linux est en train de tourner sur la carte.

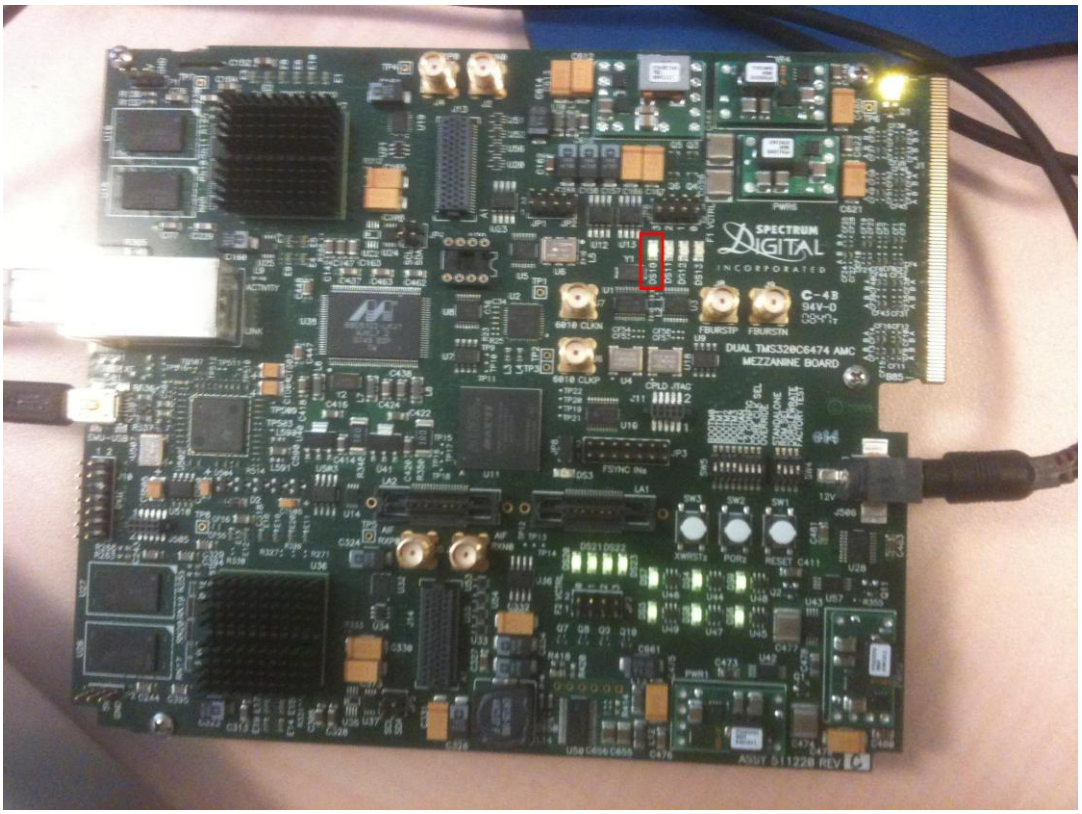


Figure 6 : EVM6474

Précisions supplémentaires :

1- Pour avoir les droits d'administrateur à partir de la console :

Taper: su

Mot de passe : *ma3drm*

2- Login : *blanc1*

3- Au redémarrage du PC, certains paramètres sont à redéfinir (Adresse IP, serveur NFS, IPtables).

4- Plus de détails sur l'environnement de Cross Compilation fourni par Texas Instruments est disponible sous le répertoire doc sous la racine du DVD de Texas Instruments. En particulier, plus de détails sur la compilation du noyau Linux sont disponibles dans le fichier HOWTO-installation.txt.

Annexe 5 : Compilation de processus à exécuter par-dessus de Linux embarqué sur EVM6474

Description de la procédure de compilation :

Le dispositif de compilation dans le répertoire *O2FE* permet de compiler les processus de supervision au niveau du *Host* pour différentes architectures (*Linux*, *ColdFire*, etc.). Dans ce qui suit, on exposera le chemin de compilation de *Ptempo* pour le processeur *ColdFire*, qui a été légèrement modifié pour être compilé pour le processeur *C64x+*.

Sous le répertoire *O2FE*, on retrouve les éléments suivants :

- *bin* : Le répertoire contenant les exécutables des processus après compilation regroupés par architecture cible
- *pgm* : Le répertoire contenant les fichiers sources de chaque processus (*.c*, *.h*, etc.)
- *csh* : Le répertoire contenant les fichiers de compilation
- *makefile* : Les fichiers de *makefile* listant les points d'entrée à la procédure de compilation
- *lib* : Un répertoire contenant les librairies utilisées par les processus de supervision, dont en particulier la librairie *Smart Light*
- *www* : Le répertoire des fichiers *html* d'exploitation de la supervision sur la cible.

Dans le fichier *makefile* sous *O2FE*, une recherche de *PTEMPO_CF* indiquera dans une première occurrence l'entrée pour la compilation du processus *Ptempo*.

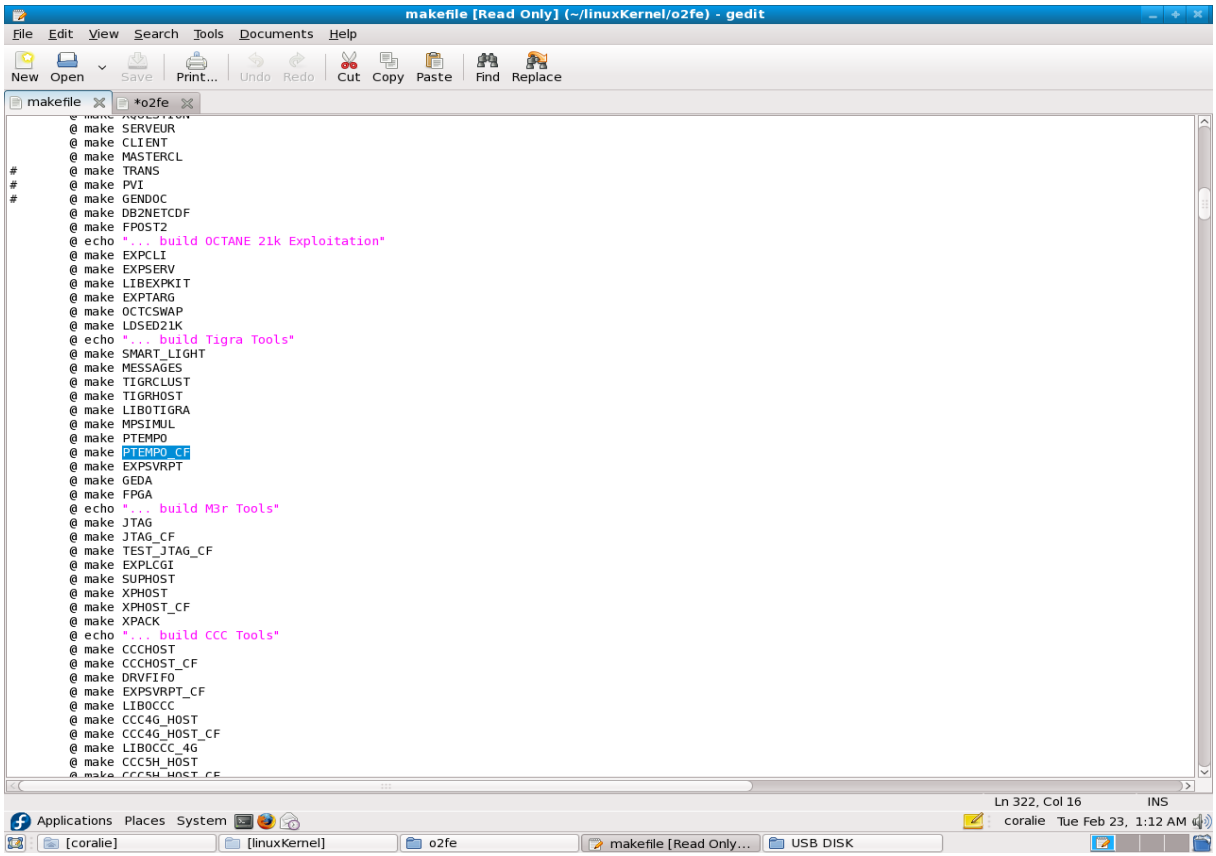


Figure 1 : makefile des fichiers systèmes

Par la suite, le fichier contient l'appel au *makefile* du niveau suivant qui est propre à ce processus. Dans ce cas, le fichier en question est *makefile.ptempo*.

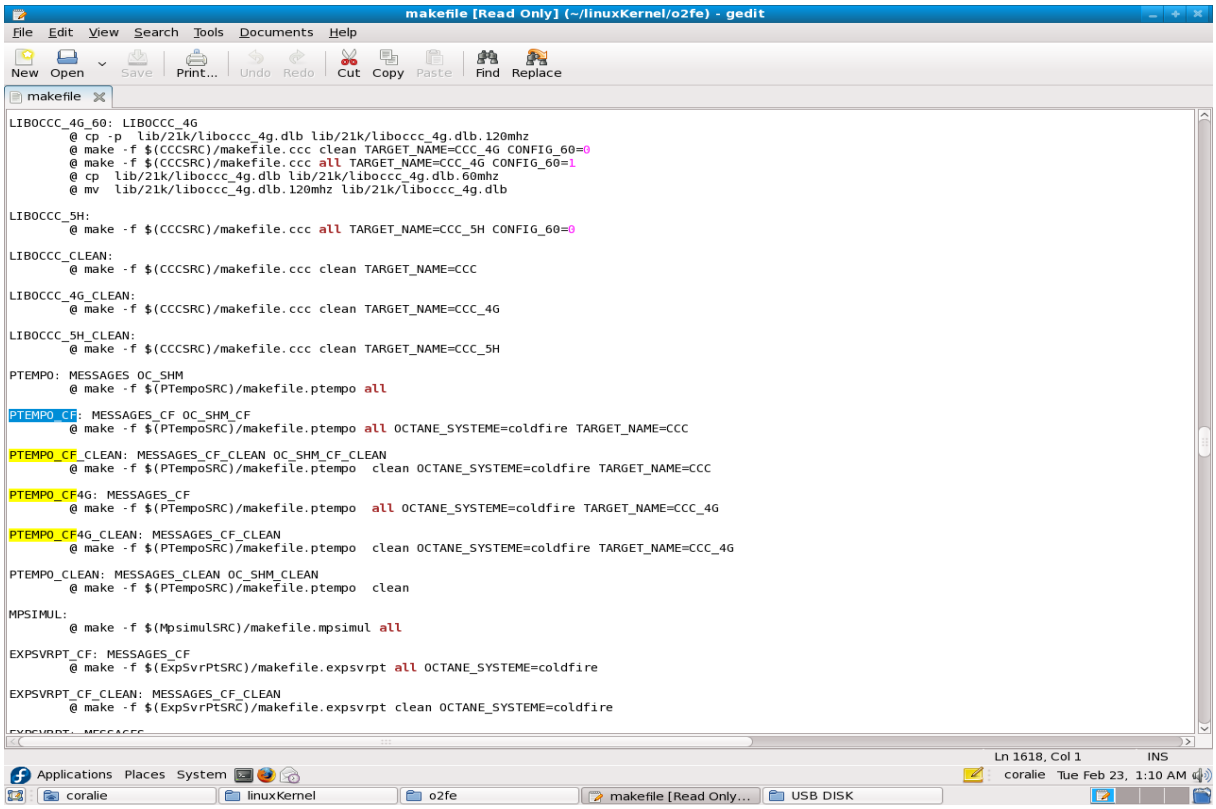


Figure 2 : Makefile montre la cible Ptempo

Dans le fichier *makesys.inc* dans le répertoire *csh* se trouvent les liens vers les *makefile* de chacune des architectures cibles.

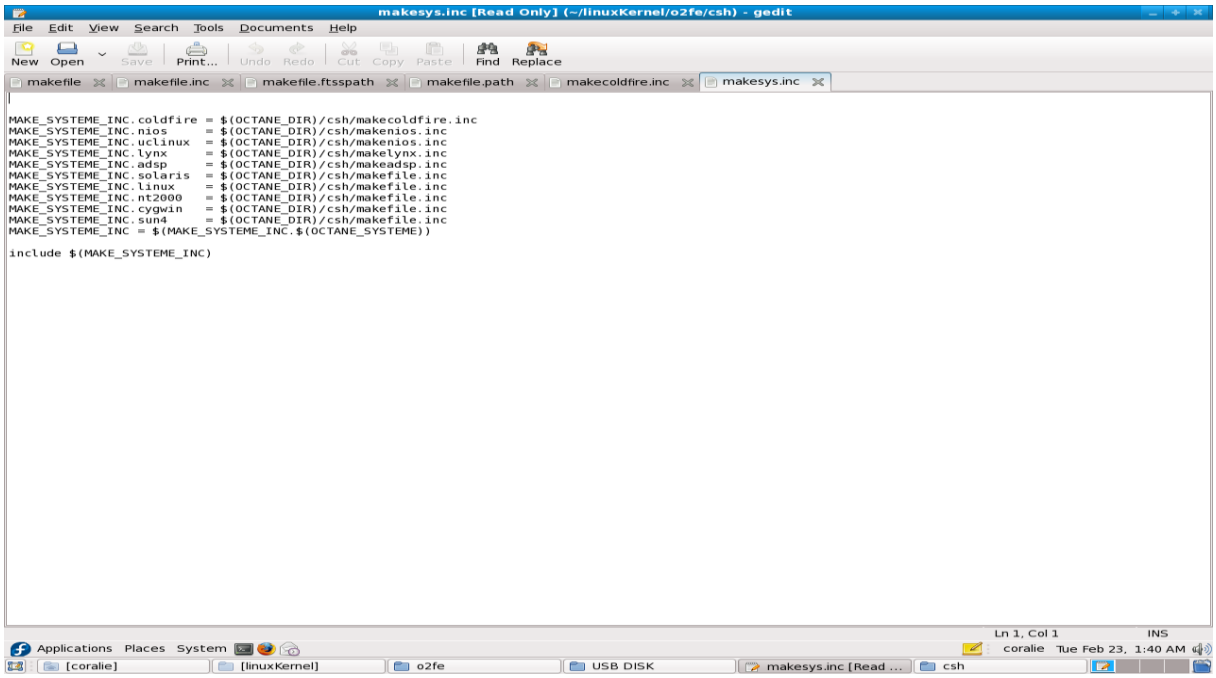


Figure 3 : makesys.inc

Comme le paramètre *coldfire* est passé pour la variable *OCTANE_SYSTEME*, le fichier *makecoldfire.inc* sous le répertoire *csh* sera chargé. Il désignera les commandes et options de compilation nécessaires et dédiés à l'architecture cible désignée, c.-à-d. le processeur *ColdFire*.

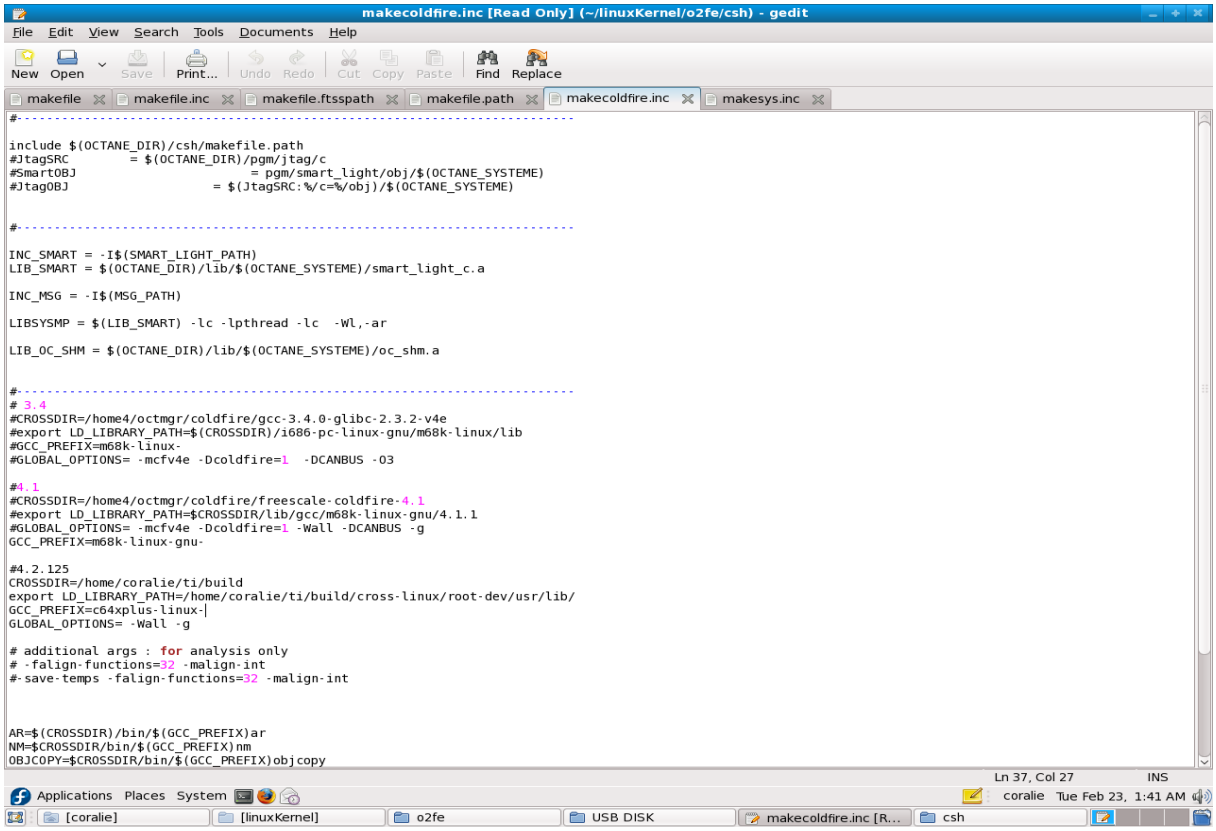


Figure 4 :makecoldfire.inc

Comme la commande de compilation de *Ptempo* dans le fichier de *makefile* fait appel au *makefile* du processus, *makefile.ptempo* dans *O2FE/pgm/c*, celui-ci sera chargé. La compilation du processus est alors effectuée en tenant compte des commandes de compilation convenables selon l'architecture cible.

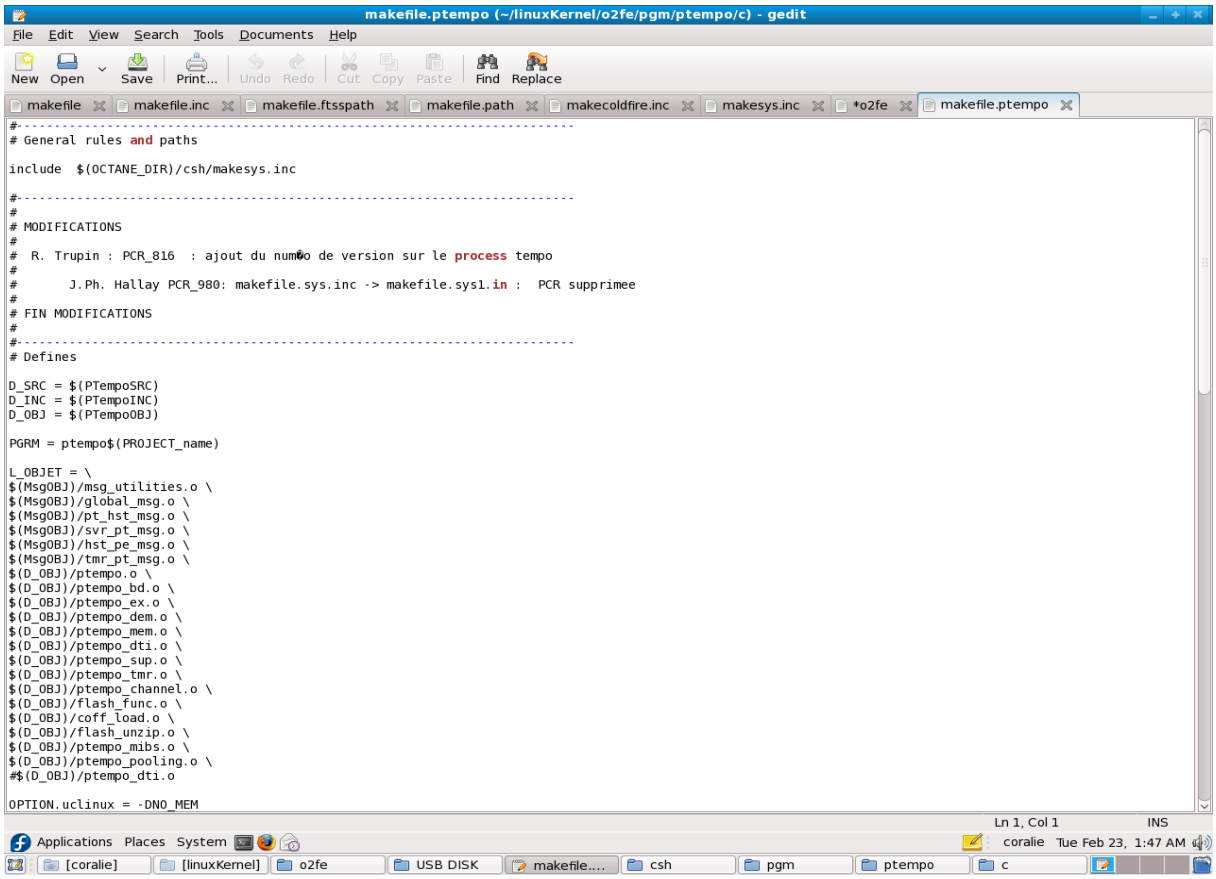


Figure 5 : Makefile.Ptempo

Il est à noter que le fichier *makefile.path* sous le répertoire *csh*, contient la définition des variables désignant les répertoires des fichiers du processus. Par exemple, le répertoire */pgm/ptempo/c* est assigné à *PtempoSRC*.

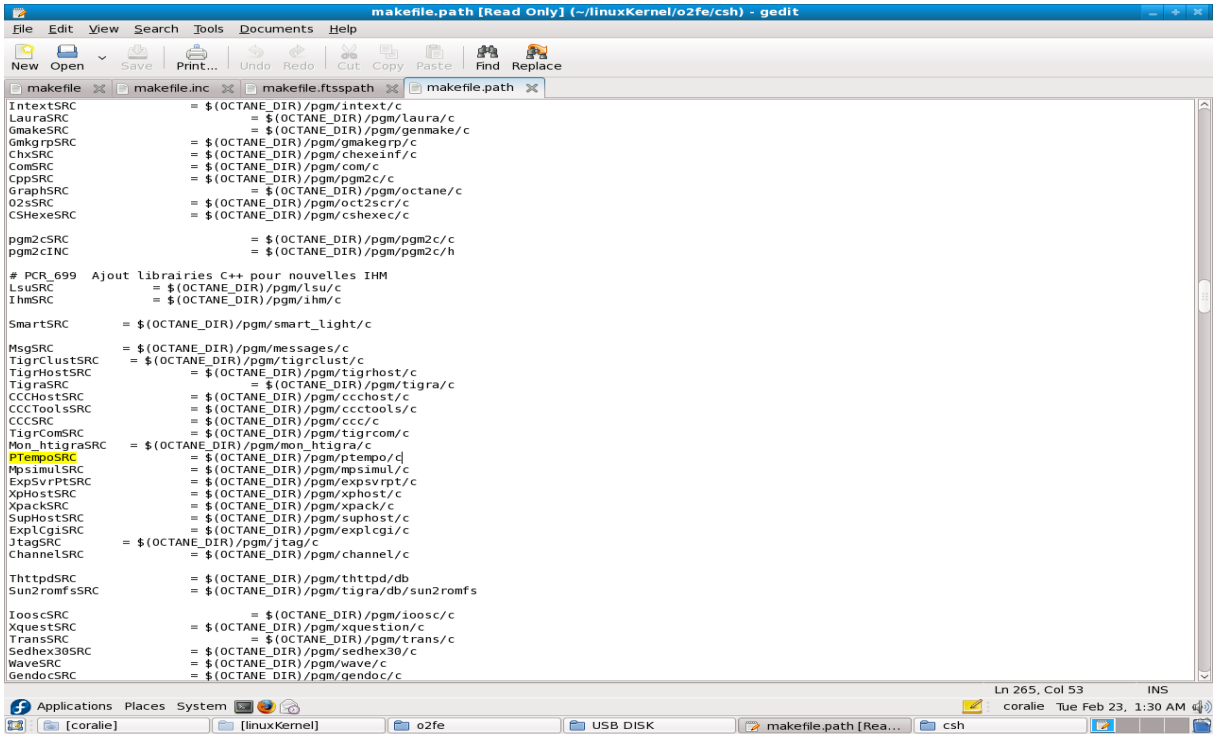


Figure 6 :makefile.path

Dans le système de compilation d'*O2FE*, il n'y a pas actuellement une procédure de compilation dédiée à l'architecture du processeur *C64x+*. Afin de tester l'exécution du processus *Ptempo* sur la carte *EVM6474*, le chemin de compilation pour le *ColdFire* est dévié afin de compiler pour le *C64x+*. La modification consiste en l'ajout d'une nouvelle version de commandes et options de compilation dans le fichier *makecoldfire.inc* dans le répertoire *csh*. La nouvelle version, *4.2.125*, spécifie le répertoire de *Cross Compilation* pour le *C64x+* qui vient d'être installé par *Texas Instruments*. C'est le répertoire noté *<build_dir>* dans le précédent document (Procédure d'installation, configuration, compilation et exécution du noyau Linux). Par conséquent, le compilateur de *Texas Instruments* pour le processeur *C64x+* se chargera de compiler le processus *Ptempo*. Cette modification est temporaire. L'ajout d'un chemin de compilation dédié au processeur *C64x+* dans le système *O2FE* sera effectué ultérieurement.

Compilation de Ptempo et Expsvrpt

Si la variable d'environnement O*CTANE_DIR* n'est pas affectée, il faut lui affecter le chemin d'accès au répertoire *o2fe (<o2fe_dir>)* :

export OCRANE_DIR=<o2fe_dir>

Ensuite, pour compiler :

- ptempo :Taper

Make PTEMPO_CF

- expcvrpt :Taper

Make EXPSVRPT_CF

Remarque : CF = *Cold Fire*

Les fichiers générés se trouvent dans *O2FE/bin/coldfire*.

Exécution

Ces deux processus communiquent via Ethernet localement, c.-à-d. au niveau de la carte, et avec l'extérieur, c.-à-d. sur le réseau Ethernet. Au niveau de la carte, ils communiquent sur l'adresse *localhost*, qui doit alors être configurée. Au niveau du réseau, il faut spécifier l'adresse IP de la carte en la passant en argument à l'exécution. Dans cet exemple, le noyau lui est assigné à la configuration l'adresse *192.168.1.1*. Dans la suite du document, on pose *<ip_evm>* l'adresse IP de la carte d'évaluation.

1- Dans un premier temps, il faut charger les binaires générés dans le système de fichier du noyau Linux dédié à la carte d'évaluation. D'autres fichiers de configurations doivent être chargés également car utilisés par *Ptempo* et *Expsvrpt*. La librairie de communication *Smart Light* doit être chargée aussi. Ci-dessous la liste de fichiers requis :

- *o2fe/bin/coldfire/expsvrpt*
- *o2fe/bin/coldfire/expsvrpt.map*
- *o2fe/bin/coldfire/ptempo*
- *o2fe/pgm/ccchost/db/filesystem_ccc/octanefsdir/sys/ptempo.ini*
- *o2fe/pgm/ccchost/db/filesystem_ccc/octanefsdir/sys/smart.ini*

Ces fichiers doivent être placés dans le système de fichier du noyau tournant sur la carte. Soit *<exec_dir>* un répertoire de *build-LINUXKERNEL/linux-root* (*home* par exemple). On place les fichiers listés ci-dessus dans *<exec_dir>* par un simple Copier-Coller. Sur la console du linux qui tourne sur la carte, on peut vérifier la visibilité de ces fichiers par rapport au noyau en tapant la commande *ls* dans *<exec_dir>*.

2- Ensuite, on lance *Expsvrpt* dans un terminal :

./expsvrpt –hpt localhost –loc *<ip_evm>*

Par exemple:

./expsvrpt –hpt localhost –loc 192.168.1.1

Où *<ip_evm>* = 192.168.1.1.

Pour configurer *localhost* sur la carte :

/sbin/ifconfig localhost 127.0.0.10

Trace d'exécution :

```
~ # ./expsvrpt -hpt localhost -loc 192.168.1.1

=======================================

|     PTEMPO EXPLOITATION SERVER     |

=======================================

version OCTANE : AA.02

Ptempo hostname = localhost:1234

Server hostname for expcli = 192.168.1.1:5678

Expsvrpt running...

Client 2 Connected <pl104675> (octcli)

Machine State received

Client 2 Disconnected

Machine State received

=======================================

|          END EXPL SERVER          |
```

3- Finalement, on lance Ptempo dans un autre terminal :

./ptempo –mon –hbd <ip_evm> –v

Pour affecter la variable d'environnement *GATEWAY* :

Export GATEWAY = *<ip_evm>*

Annexe 6 : Compilation et exécution et encapsulation des exemples de Texas de la communication inter-cœurs

1) EXEMPLE 1: communication inter-cœurs en utilisant les bus EDMA, la librairie CSL, les modules IPC et Sémaphores et le noyau DSP /Bios

Un des mécanismes utilisés pour la communication inter- cœurs pour C6474 est l'implémentation des registres hardware de sémaphore qui peuvent être accédés par les trois cœurs de DSP d'une manière atomique qui sont utilisés pour assurer une gestion efficace des ressources partagées.

1. Le code source:

Pour avoir le code source de ce premier exemple il faut, qu'on soit déjà inscrit dans les forums de Texas Instrument. Ce code source peut être téléchargé d'un forum de Texas dont le lien est :

http://e2e.ti.com/support/dsp/tms320c6000_high_performance_dsps/f/112/p/34382/123512.aspx

1.2 Description:

Cet exemple de code montre la capacité du DSP C6474 d'exécuter du code identique sur les trois cœurs. Ce code effectuera des transferts EDMA d'un cœur à l'autre, à partir de cœur 0. Il envoie ensuite une interruption IPC au cœur de réception pour lui informer que des données sont arrivées. Quand un cœur reçoit un IPC, il éditera ces données et commencera un nouveau transfert EDMA pour l'envoyer au cœur suivant, dans l'ordre. Lorsque le transfert est complet, il enverra un IPC au cœur de réception pour lui indiquer de faire la même chose.

1.3 Compilation et exécution

Pour compiler cet exemple, on doit d'abord s'assurer que le programme accède à la librairie CSL (indiquer donc le chemin dans **"build option"** comme montré dans le capture d'écran ci-dessous). On peut utiliser la version de DSP/bios "bios_5_33_01" ou une autre version plus récente disponible avec CCS.

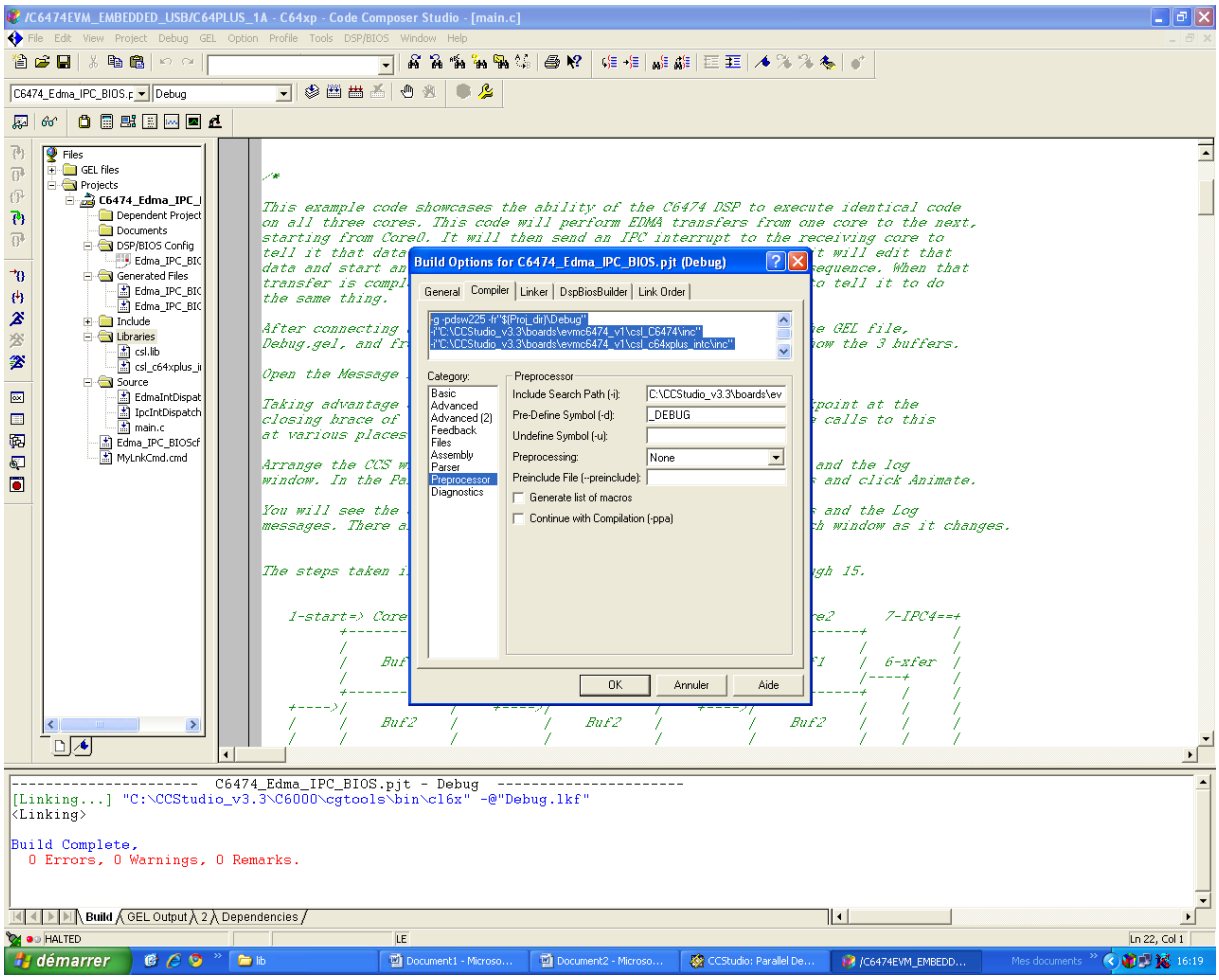


Figure 1 :options de configuration pour le compilateur

D'abord on connecte et on charge ce programme dans chaque cœur, puis on charge le fichier GEL: **Debug.gel** ensuite à partir de bar de menu, on exécute **GEL->Debug->MemWindows** pour afficher les trois mémoires tempons (**Buffers**). Enfin on œuvre la fenêtre log des messages (**Message Log window**).

La capture d'écran ci-dessous indique l'état des buffers avant l'exécution:

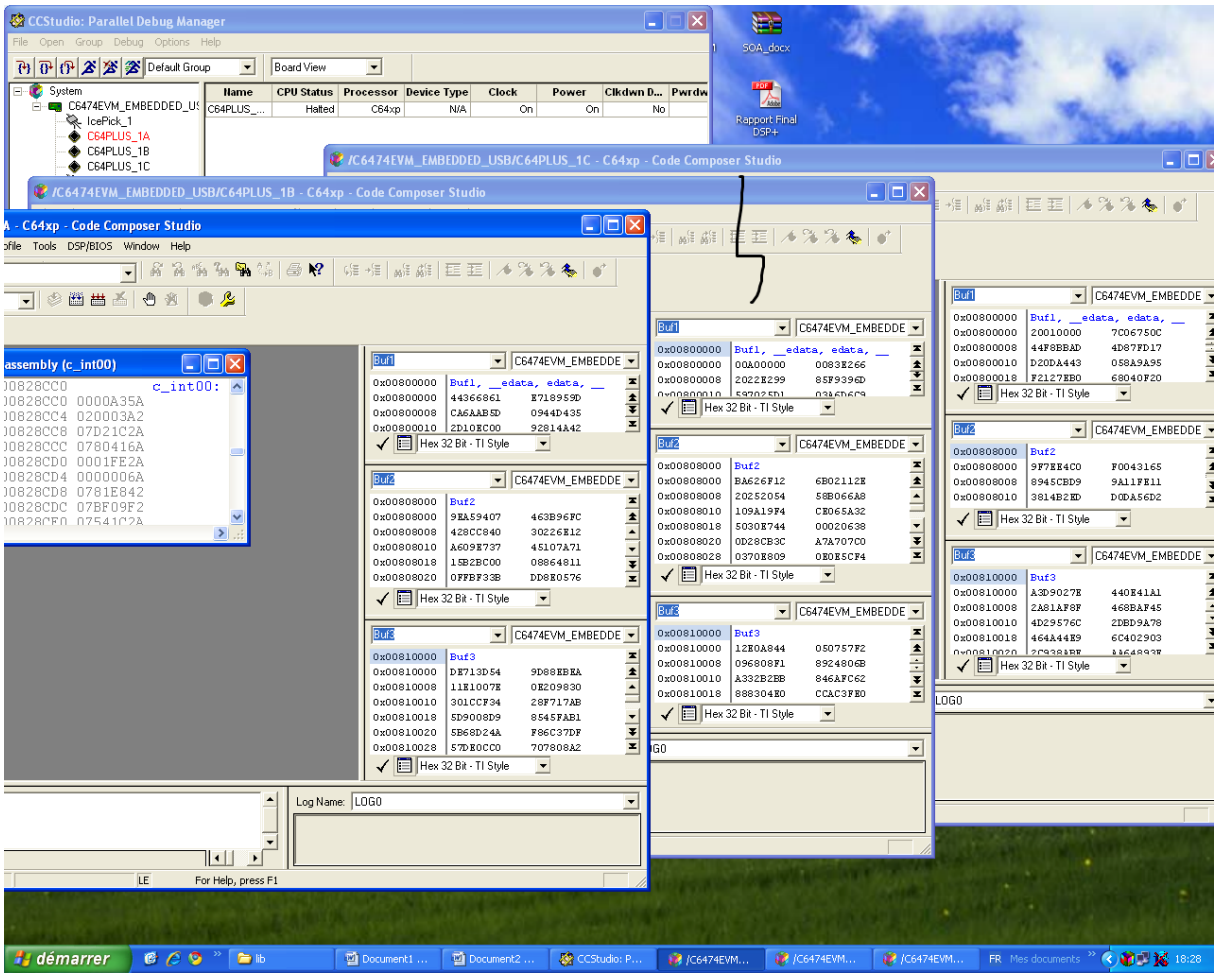


Figure 2 : les fenetres de mémoire avant execution

Profitant de la fonction d'atterrissage **Bkpt()**, pour définir un point d'arrêt unique à l'accolade fermante de la **Bkpt ()** qui se trouve juste au-dessus de **main ().** Il ya les appels vers divers endroits dans deux tâches (stage 1 et stage 2).

On essaye d'arranger les fenêtres CCS afin qu'on puisse voir toutes les fenêtres de la mémoire et la fenêtre de log. En **PDM**(**Parallel Debug Manager**), on sélectionne toutes les cœurs puis on clique sur « **animate »**.

La capture d'écran en-dessous indique l'état des **buffers** après l'exécution:

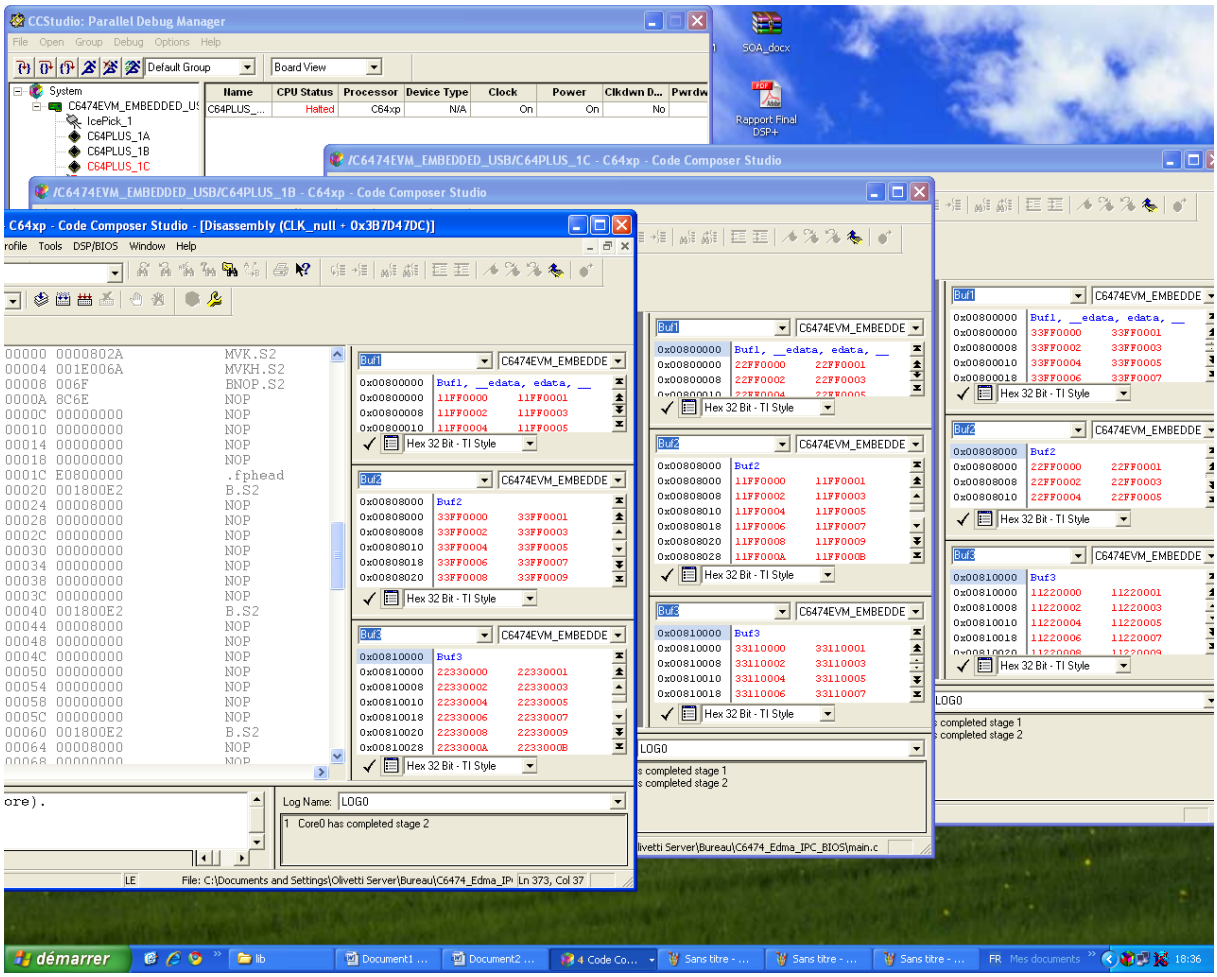


Figure 3 : les fenetres de mémoire aprés execution

A ce niveau, nous pourrons voir le déplacement des données entre les fenêtres de la mémoire, et les messages du dans la fenêtre log des messages (**Log window**). Il ya **TSK_sleep ()** qui reçoit en argument le temps pour nous laisser voir chaque fenêtre ainsi les changements pendant cette durée.

Les étapes prises dans cet exemple sont simplifié par le schéma suivant:

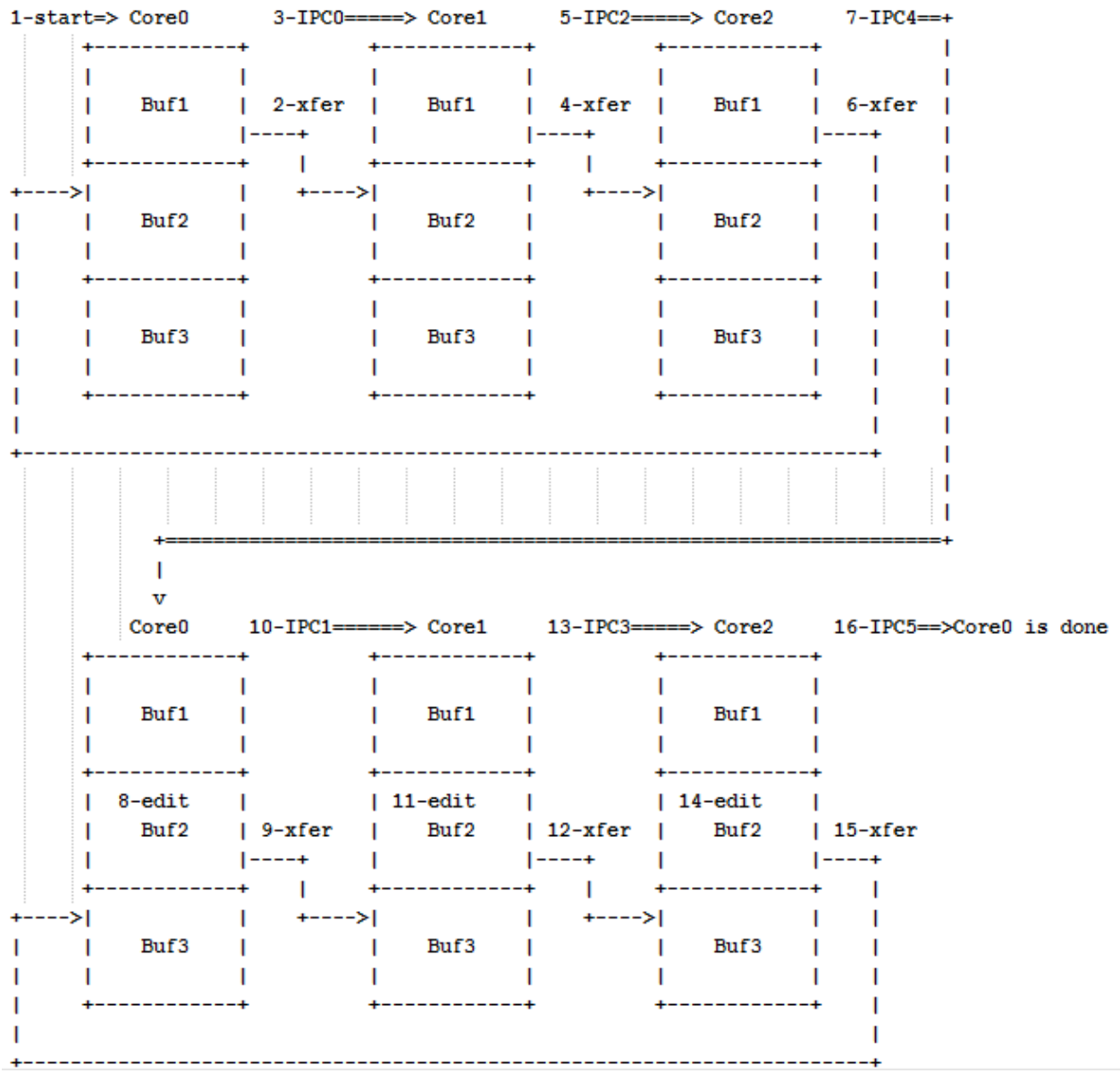


Figure 3 : les étapes d'exécution

Une chose importante à noter est le contenu de **BUF2** dans la fenêtre de la mémoire (**Memory Window**). Ils sont mis en évidence dans la cache **L1D**. Cela signifie que les vraies valeurs de ces places de mémoire ne sont pas stockées en **L2**, mais sont toujours dans le cache **L1D**. Si nous décochons la case **L1D** au bas de cette fenêtre, nous verrons alors les anciennes valeurs qui sont encore stockés dans **L2**. Mais les données qui ont été envoyé au cœur 1 de **Buf3** sont les nouvelles données qui sont encore en **L1D** cœur 0. L'interface avec **EDMA3 L2** fait une opération de lecture à **L1D** pour voir si des données ont besoin d'être tirées de cette place.

Si nous modifions l'exemple et utilisons les adresses mémoires externes pour BUF1-3, ou tout simplement BUF2, nous obtiendrons des données anciennes, sauf si nous exécutons manuellement les commandes de cohérence de cache.

1.4. Encapsulation de programme principal 'main.c':

Le programme principal '**main.c'** de cet exemple est complexe et illisible ou incompréhensible pour l'utilisateur si ce dernier n'a pas de connaissances techniques sur la programmation avec la bibliothèque **csl** et aussi avec les différents modules de EDMA et de IPC. Aussi dans le but de lisibilité de programme et dans le but de jouer sur les différents paramètres de cet exemple, on a implémenté un nouveau programme à partir de l'ancien **main.c** comme montrer en- dessous. Ainsi une bibliothèque est réalisée de ce nouveau programme principal.

```
//pour identifier Uint32 (unsigned int 32 bit), on va mettre avec les includes cette déclaration:

#include <tistdtypes.h>

//all_functions.h : contient la déclaration de toutes les fonctions utilisées dans cet exemple

#include "all_functions.h"

//BUF_SIZE_W définit  la taille de données et aussi  de celle des mémoires tampons (buffers)

#define BUF_SIZE_W 8192

// Ceci est la déclaration d'un tableau de données de type Uint32

 Uint32 data2[BUF_SIZE_W];

 // Il est utilisé pour la boucle for de données

 Uint32 i;

 // Il s'agit de définir l'adresse locale de la source buffer1, mais il est converti à l'adresse globale dans le programme

 Uint32 add_buf1_src;

  // Il s'agit de définir l'adresse locale de la destination buffer1 mais il est converti à l'adresse globale dans le programme
```

```
Uint32 add_buf2_dst;

 // Il s'agit de définir l'adresse locale de la source buffer2  et c'est la meme que celle de la destination //buffer1  mais il est converti à l'adresse globale dans le programme

Uint32 add_buf2_src;

// Il s'agit de définir l'adresse locale de la destination buffer2 mais il est converti à l'adresse globale dans le programme
```

```
Uint32 add_buf3_dst;

        // Pour les effets visuels, il est utilisé ici pour TSK_sleep dans le  stage1

int nticks=5000;

// Pour les effets visuels, il est utilisé ici pour TSK_sleep dans le  stage2

int n_ticks=5000;

void main(void)

{

// Fonction principale de l'initialisation et la configuration des modules  edma, IPC et DSP / Bios avec l'utilisation de la bibliothèque CSL

setup_edma_main() ;

}

/*Cette fonction décrit la première étape et d'effectuer le transfert edma dans son premier canal tel qu'il est configuré  entre les données de buffer1 de chaque cœur et de buffer2 de cœur suivant (cœur de destination)*/

void stage1()

{

for ( i = 0; i < BUF_SIZE_W; i++) {

                data2[i]=2*i;  }

add_buf1_src=0x00800000 ;
```

```
add_buf2_dst=0x00808000;

TSK_Stage1_CreateForwardData(data2,add_buf1_src,add_buf2_dst,nticks);

}

/*Cette fonction décrit la première étape et d'effectuer le transfert edma dans son seconde
canal tel qu'il est configuré  entre les données de buffer2 de chaque cœur et de buffer3 de
cœur suivant (cœur de destination)*/

)

void stage2()

{

 for ( i = 0; i < BUF_SIZE_W; i++) {

                data2[i]=2*i+1;  }

          add_buf2_src=0x00808000;

   add_buf3_dst=0x00810000;
```

```
TSK_Stage2_ProcessForwardData(data2,add_buf2_src,add_buf3_dst,n_ticks);

 }
```

2. EXEMPLE 2: Communication Inter-Coeurs via IPC module

Le mécanisme d'échange de données tampons entre des cœurs implique généralement qu'un cœur écrit des données à une zone mémoire particulière - soit par **CPU**, mémoire cache ou via l'accès direct à la mémoire (**DMA**), dans la mémoire partagée, comme la mémoire **DDR2** (**double data rate**) ou en l'un des zones de mémoire statique (**SRAM**), et d'informer le(s) cœur(s) de destination que les données sont disponibles.

2.1. Le code source de l'exemple:

Ce document a le lien pour l'exemple 2 de la communication entre les cœurs et plus de détails pour compilation et l'exécution:

http://focus.ti.com/dsp/docs/dspsupporttechdocsc.tsp?sectionId=3&tabId=409&familyId=1635&abstractName=sprab20

2.2 Build Environment

Avant d'utiliser le projet, le variable **CSLforIPC** de Code Composer Studio doit être définie dans **c:/CCStudio_v3.3/cc/bin/macro.ini**. Cette variable est utilisée pour créer le chemin de recherche pour la bibliothèque **CSL**. Il y a une deuxième solution très simple, il s'agit de remplacer cette variable par **C:/CCStudio_v3.3/boards/evmc6474_v1/csl_C6474** dans **build options** de compilateur. Comme on le voit en-dessous:

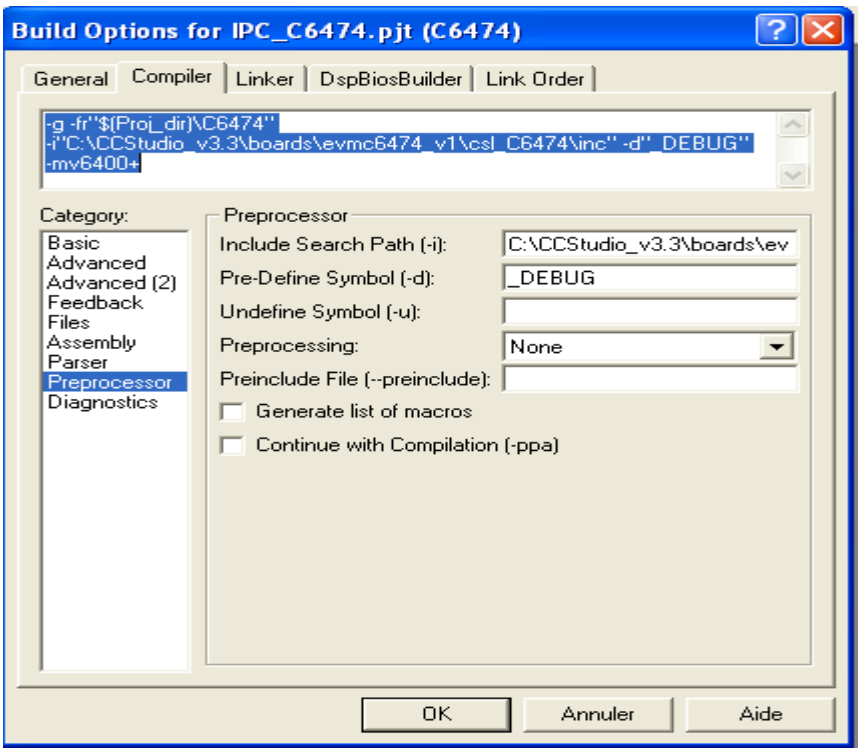


Figure 5 : options de configuration de compilateur

La version de DSP/BIOS requis est 5.32.04 ou plus récent est nécessaire pour supporter les fichiers de la Plate-forme EVM6474. Dans ce projet, on a utilisé la version 5_33_01.

2.3. Le fonctionnement de l'application

Cette Application s'exécute sur tous les trois cœurs. Chaque cœur envoie de messages via les registres IPC vers les deux autres cœurs. Le message est envoyé périodiquement en se basant sur les événements de **timer** de la carte EVM6474. Lorsque le cœur qui génère le message (le cœur source) écrit dans le registre IPCGR qui correspond au cœur de réception (le cœur de destination), le cœur de destination reçoit une interruption. Cela correspond à la routine de service d'interruption (**ISR : Interruption Service Routine**), qui s'exécute sur le cœur de destination, et qui détermine le cœur qui a été la source du message et écrit le

sémaphore logiciel correspondant, il ya trois sémaphores logiciels au total, un par cœur. L'écriture (**Posting**) du sémaphore logiciel produit un changement de tâche qui était en attente (**pending)** sur le sémaphore. Il ya trois tâches et chaque processus de messages est l'origine d'un cœur particulier. Notez que sur un cœur donné, uniquement deux tâches sont actifs puisque, dans cet exemple, un cœur n'envoie pas un message à lui-même.

L'utilisation de bits de SRCS pour les trois registres IPCGR est définie comme suit:
· SRCS0-7 et SRCS24 sont utilisés par cœur 0
· SRCS8-15 et SRC25 sont utilisés par cœur 1
· SRCS16-23 et SRC26 sont utilisés par cœur 2

Les bits de poids forts (SRCS24-26) sont utilisés par le cœur de destination afin de déterminer le cœur origine de l'interruption. Les bits de poids faible (SRCS0-23) sont utilisés pour les drapeaux (flags). Dans cette simple application, tous les 8 bits sont utilisés pour vérifier facilement que le cœur de réception a reçu toutes les interruptions.

2.4. Exécution de l'application sur la carte d'évaluation EVM6474

Les étapes pour compiler et exécuter cette application sur l'EVM6474 sont les suivantes:

1. Configurer **Code Composer Studio setup** pour le C6474.
2. lancer **Code Composer Studio**.
3. ouvrir **C64PLUS_F1A**, **_F1B** et **_F1C** de gestionnaire PDM (**Parallel Debug Manager**). Ces sessions correspondent aux cœurs 0, 1, et 2 sur le DSP0.
4. Procédez comme suit dans chaque session :
a. Project → Open … et naviguer à ipc_c6474.pjt.
b. Debug → Connect.
c. Project → Build (F7).
d. File → Load … naviguer à IPC.out.
e. ouvrir DSP/BIOS => Message Log.
5. Choisir Debug => Run de PDM. Les trois cœurs se mettront en marche en même temps. Aprés 5-10 seconds, le **Message Log** devrait commencer à diffuser des messages similaires à la figure ci-dessous:

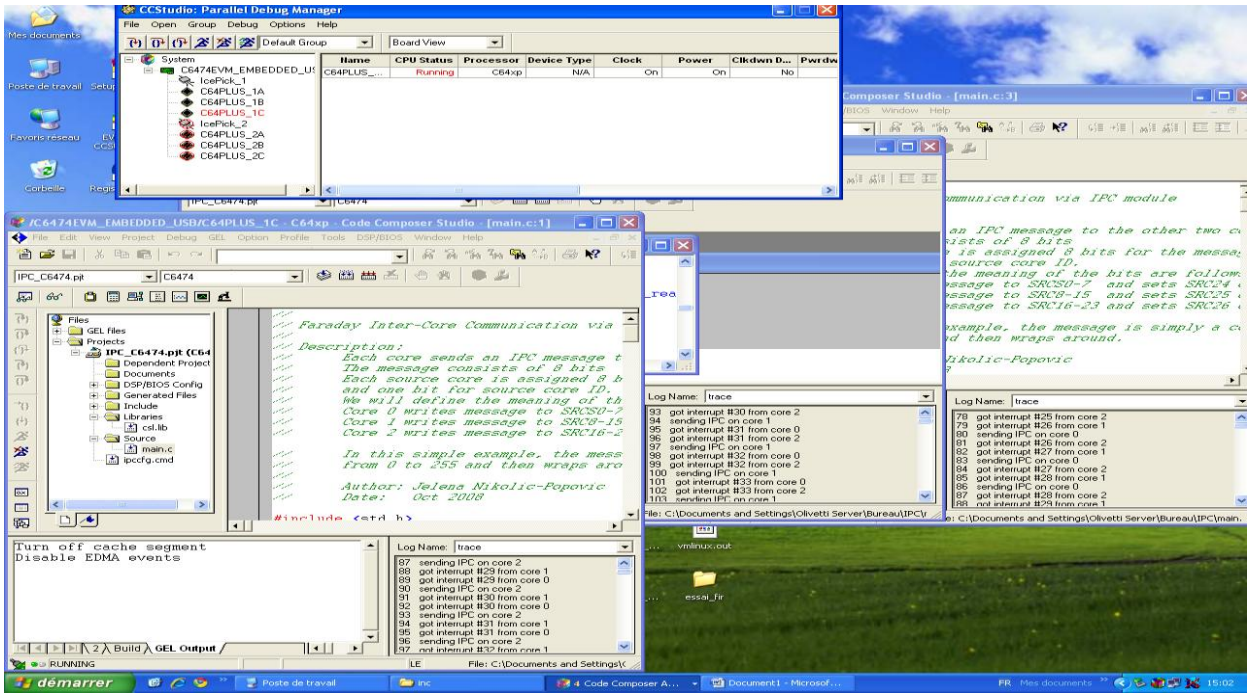


Figure 6 :execution et résultats

Annexe 7: Création de la librairie c6474_edma_ipc_bios_lib.lib

I. Introduction

La bibliothèque est utilisée pour faciliter l'accès aux programmes secondaires, appelés par le programme principal. Cette bibliothèque sera crée à partir d'un exemple de Texas pris dans le site de Texas **C6474_Edma_IPC_BIOS.pjt** dont Le programme **main.c** est surchargé par les fonctions avant encapsulation. L'encapsulation ici consiste à déclarer les différentes fonctions dans un en-tête (**all_functions.h**) ensuite les définir dans un autre fichier C **(all_function.c**) et enfin, appeler et utiliser ces fonctions dans un nouveau programme principal qu'on l'appelle aussi **main.c**. Notre bibliothèque va contenir donc les programmes **all_function.c** et d'autres programmes qui n'ont pas été utilisées directement dans le programme principal à savoir **EdmaIntDispatcher.c**, **Edma_IPC_BIOScfg_c.c** et **IpcIntDispatcher. C** (qu'elles existaient à l'extérieur de l'ancien programme **main.c** avec des fichiers sources).

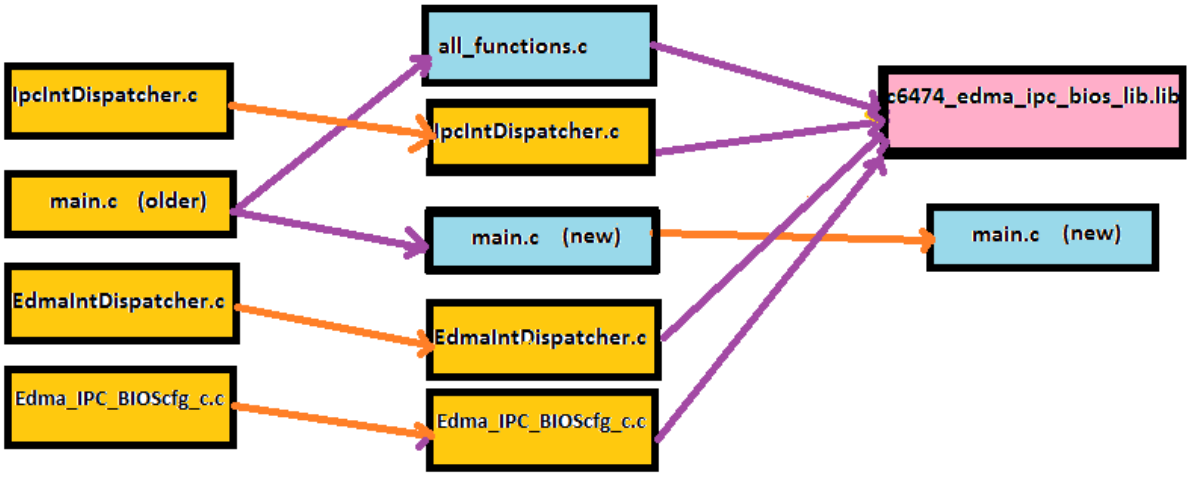


Figure 1 : Procédure de l'encapsulation de et de réalisation de la librarie c6474_edma_ipc_bios_lib.lib

II. les étapes de compilation :

Dans le menu projet on choisit crée une librairie comme le montre la figure suivante :

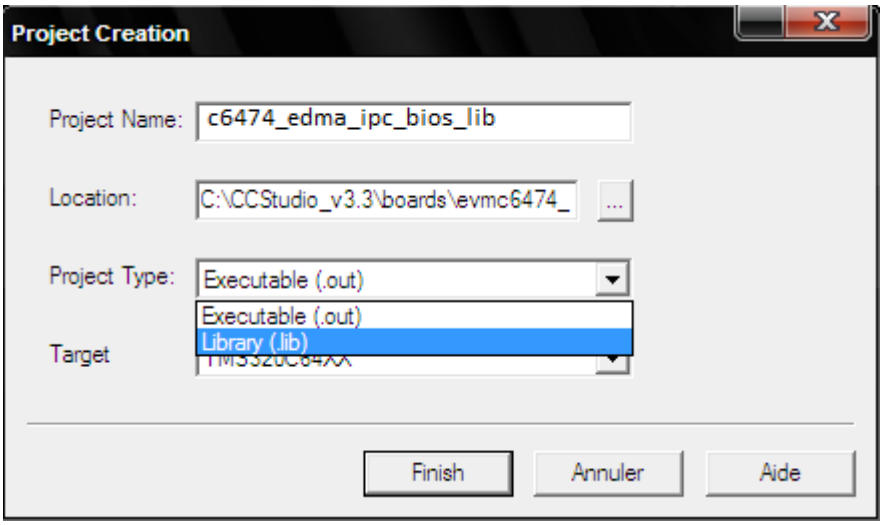


Figure 2 : création d'une nouvelle librairie

On insère les différentes sources qu'on a citées auparavant et aussi le fichier **all_function.h** qui contient les constantes et les déclarations des fonctions figure 2 (voir **projects**).

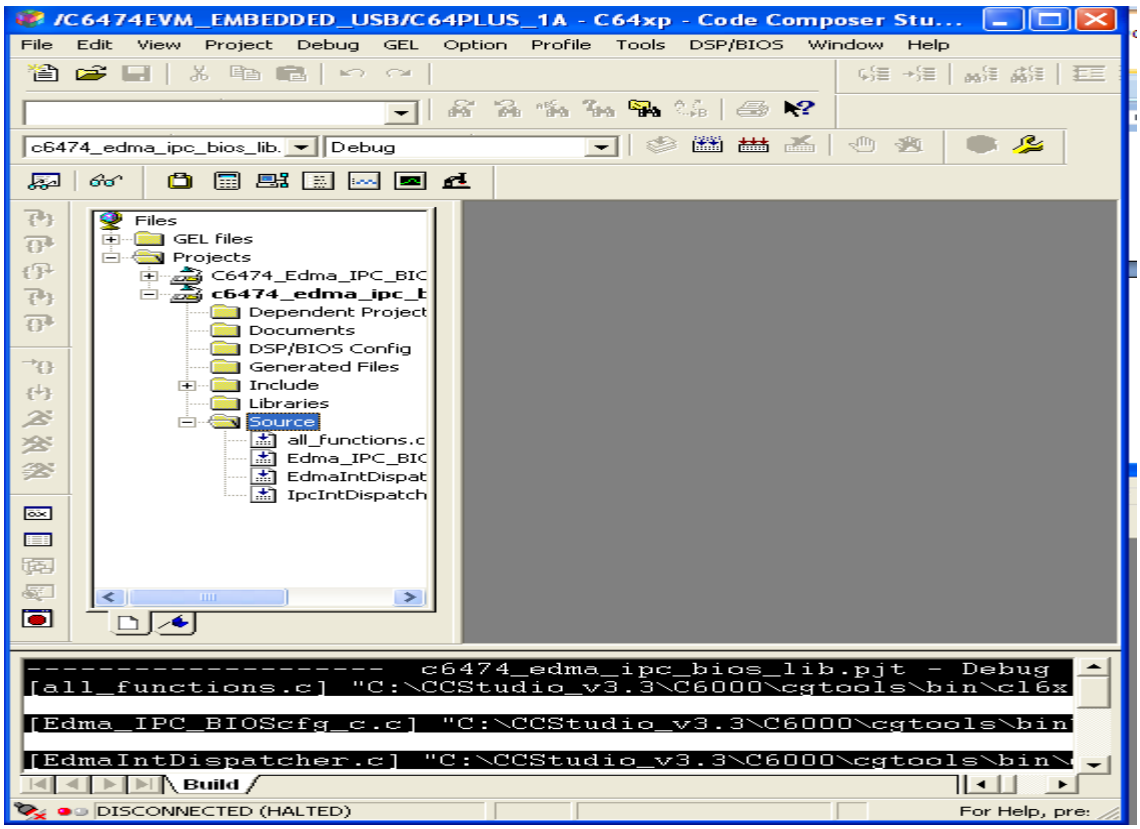


Figure 3 : l'ajout des fichiers sources de la librairie

On enregistre le fichier **all_function.h** dans un dossier (par exemple le dossier des Inc.) et on sélectionne Project-→build option pour monter au compilateur le chemin de ce header (figure suivante) ainsi que les autres header CSL.

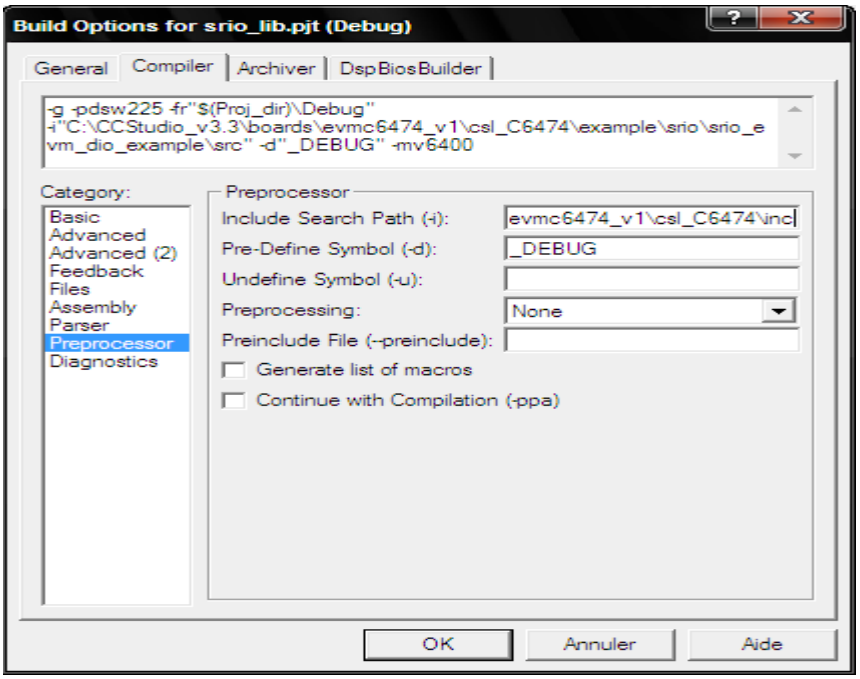


Figure 4 : options de compilateur

Remarque

Il ne faut pas oublier de sélectionner l'option –mv6400+ pour le compilateur au lieu –mv6400.

On compile notre projet et on a la création de la bibliothèque **c6474_edma_ipc_bios_lib.lib** dans le dossier Debug de projet courant.

Trace de compilation

```
---------- c6474_edma_ipc_bios_lib.pjt - Debug ----------

[all_functions.c] "C:\CCStudio_v3.3\C6000\cgtools\bin\cl6x" -g -pdsw225 -fr"C:/Documents and Settings/Olivetti Server/Bureau/c6474_edma_ipc_bios_lib/c6474_edma_ipc_bios_lib/Debug" -i"C:/CCStudio_v3.3/boards/evmc6474_v1/csl_C6474/inc" -i"C:/CCStudio_v3.3/boards/evmc6474_v1/csl_c64xplus_intc/inc" -d"_DEBUG" -mv6400 -@"../Debug.lkf" "all_functions.c"
```

```
[Edma_IPC_BIOScfg_c.c] "C:\CCStudio_v3.3\C6000\cgtools\bin\cl6x" -g -pdsw225 -fr"C:/Documents
and Settings/Olivetti Server/Bureau/c6474_edma_ipc_bios_lib/c6474_edma_ipc_bios_lib/Debug" -
i"C:/CCStudio_v3.3/boards/evmc6474_v1/csl_C6474/inc" -
i"C:/CCStudio_v3.3/boards/evmc6474_v1/csl_c64xplus_intc/inc" -d"_DEBUG" -mv6400+ -
@"../Debug.lkf" "Edma_IPC_BIOScfg_c.c"


[EdmaIntDispatcher.c] "C:\CCStudio_v3.3\C6000\cgtools\bin\cl6x" -g -pdsw225 -fr"C:/Documents and
Settings/Olivetti Server/Bureau/c6474_edma_ipc_bios_lib/c6474_edma_ipc_bios_lib/Debug" -
i"C:/CCStudio_v3.3/boards/evmc6474_v1/csl_C6474/inc" -
i"C:/CCStudio_v3.3/boards/evmc6474_v1/csl_c64xplus_intc/inc" -d"_DEBUG" -mv6400+ -
@"../Debug.lkf" "EdmaIntDispatcher.c"


[IpcIntDispatcher.c] "C:\CCStudio_v3.3\C6000\cgtools\bin\cl6x" -g -pdsw225 -fr"C:/Documents and
Settings/Olivetti Server/Bureau/c6474_edma_ipc_bios_lib/c6474_edma_ipc_bios_lib/Debug" -
i"C:/CCStudio_v3.3/boards/evmc6474_v1/csl_C6474/inc" -
i"C:/CCStudio_v3.3/boards/evmc6474_v1/csl_c64xplus_intc/inc" -d"_DEBUG" -mv6400+ -
@"../Debug.lkf" "IpcIntDispatcher.c"


[Archiving...] "C:\CCStudio_v3.3\C6000\cgtools\bin\ar6x" @"Debug.lkf"

  ==>  new archive 'C:\Documents and Settings\Olivetti
Server\Bureau\c6474_edma_ipc_bios_lib\c6474_edma_ipc_bios_lib\Debug\c6474_edma_ipc_bios_lib
.lib'

  ==>  building archive 'C:\Documents and Settings\Olivetti
Server\Bureau\c6474_edma_ipc_bios_lib\c6474_edma_ipc_bios_lib\Debug\c6474_edma_ipc_bios_lib
.lib'


Build Complete,

  0 Errors, 0 Warnings, 0 Remarks.
```

III. Test de la librarie c6474_edma_ipc_bios_lib.lib

Le but de cette partie est de tester la librairie créé **c6474_edma_ipc_bios_lib.lib** dans n'import quel projet par exemple **test_lib_intrer_core_edma_ipc.pjt**
On a deux méthodes pour que le compilateur prenne en compte cette bibliothèque :

- Soit on montre le chemin de cette biblio dans la rubrique project→build Option (déjà vue)
- Soit on l'ajoute au projet comme la montre la figure ci-dessous:

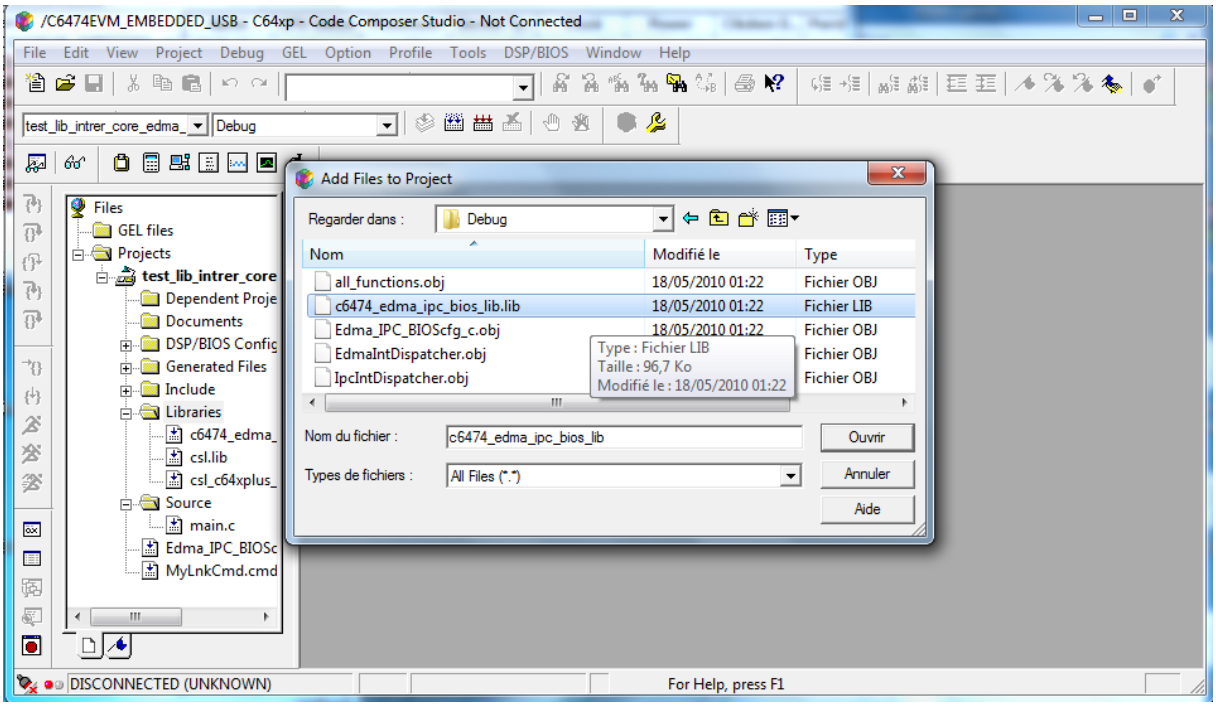


Figure 5 : répertoire de la librairie

- On écrit un programme principal (nouveau **main.c**) qui utilise les fonctions de cette librairie comme la montre la figure ci-dessous :

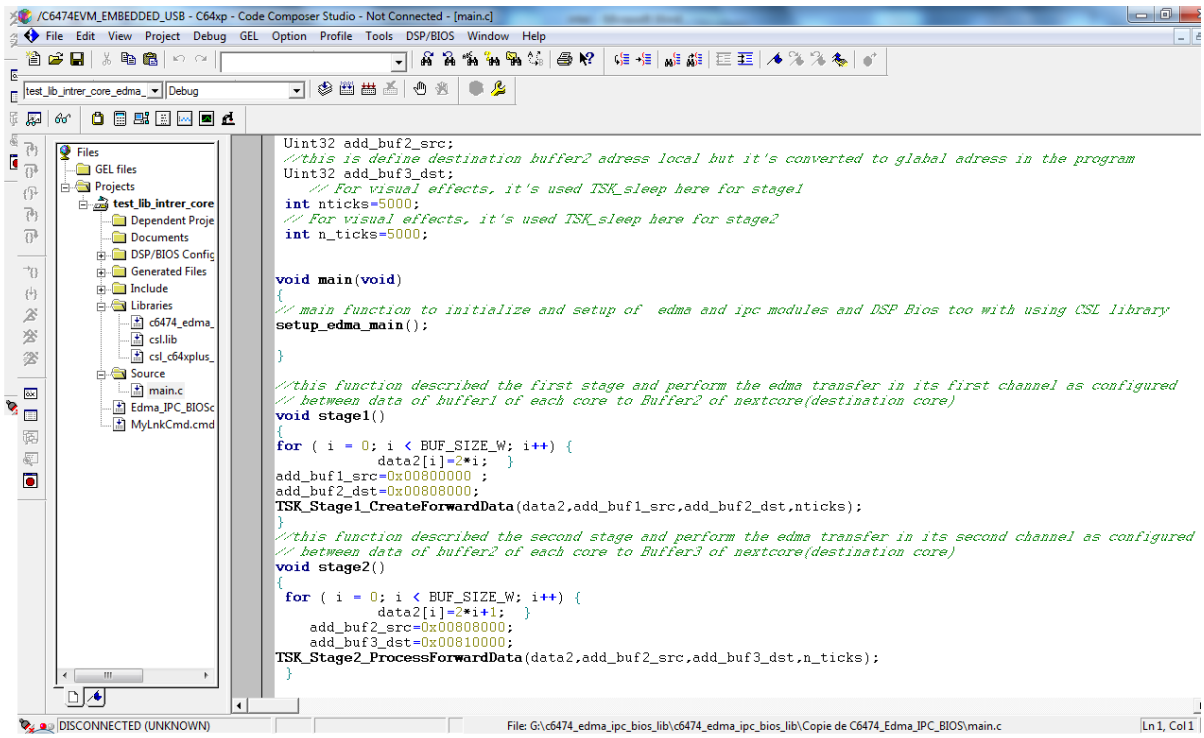


Figure 6: programme principale appelle la librairie crée

Avant de compiler notre projet on ne doit pas oublier de montrer au compilateur le chemin des fichiers headers utilisés, au Linker les librairies utilisées et on exécute notre projet.

Annexe 8 : Benchmark applicatif de référence

1- Code du host

Le code source de host représente le code de cœur 0 de DSP de calcul qui est basé sur les mécanismes de communication inter –cœur tels qu'IPC, sémaphore, DSP/BIOS et SRIO. La mission principale de ce cœur est de gérer et interroger les deux autres cœurs 1 et 2 pour qu'ils fassent des traitements de façon indépendante et au même temps récupérer les résultats de calculs, les traces et les **status** de supervision afin de les envoyer par Srio vers le DSP de supervision.

1.1 L'algorithme

Début de Programme:

- Host notifie C1 de lire data1 (tableau de 1 à 255 incrémenté) à une adresse précisée par le host
- Host notifie C2 de lire data2 (tableau de 0 à 510 incrémenté deux fois 0 2 4… par exemple) à une adresse précisée par le host
- Wait() ;//attend que c1 ou c2 termine le calcul ou le traitement de données.
- Si C1 notifie le host :

  -aller à l'adresse précisée par C1 et récupérer les résultats, les traces et status (out1).

  - envoyer par srio ces résultats à c0 de dsp1 de supervision

- Si C2 notifie le host :

  -aller à l'adresse précisée par C2 et récupérer les résultats, les traces et status (out2).

  - envoyer par srio ces résultats à c0 de dsp1 de supervision

Fin de programme ;

1.2 Compilation du code

Le programme du host s'exécute sous DSPBIOS, pour cela, avant la compilation, il faut bien configurer ce dernier en termes de

- mémoire, en effet il faut préciser les plages des adresses mémoire pour toutes les sections mémoires existées dans le fichier de commande.
- LOG, pour afficher les messages dans le ''Message Log''.
- ''Scheduling'' en changent dans les propriétés de ''PRDipcsend'' le mode en ''one-shot''.

Le fichier de configuration de DSPBIOS illustre les changements effectués pour les autres primitives.

2 . Code source du coeur1 et coeur2 de DSP0 de calcul

Le code source de ces cœurs est basé sur les mécanismes de communication inter-cœurs tels qu'IPC, sémaphore et DSP/BIOS.

2.1 Algorithme de cœur 1 (c1) :

- C1 attend jusqu'à avoir une notification de host
- C1 lit les data1 à l'adresse précisée par le host
- C1 effectue le traitement approprié (calcul de out1=2Xdata1, récupère les status, les traces… par exemple)
- C1 stock les résultats de calcul à un endroit de mémoire connue dans ddr2 et précisé par C1 ou le host.
- C1 notifie le host fin de calcul

Fin de programme C1

2.2 Algorithme de cœur 2 (c2) :

- C2 attend jusqu'à avoir une notification de host

- C2 lit les data2 à l'adresse précisée par le host

- C2 effectue le traitement approprié (calcul de out2=2Xdata2+1, récupère les status, les traces… par exemple)

- C2 stock les résultats de calcul à un endroit de mémoire connu dans ddr2 et précisé par C2 ou le host.

- C2 notifie le host fin de calcul

Fin de programme C2

3. Code source du coeur0 de DSP1 de supervision

Le code source de ce cœur est celui de Serial rapid IO. Il permet d'activer, initialiser, et ouvrir le protocole sans faire aucune transaction. En effet il reçoit les données transmises par le host via SRIO.

4. résultats :

Dans cette application, on définit les adresses mémoires dans le DDR2 de chaque DSP.

- src(0x8000D000) : présente l'adresse de premier tableau de données (figure1).
- srcc(0x8000DFOO): présente l'adresse de deuxième tableau de données (figure2).
- Out 1(figure 3) présente l'adresse mémoire dans DDR2 de DSP de calcul qui réalise un premier traitement de donnée (données pointées à src)
- Out 2(figure 4) présente l'adresse mémoire dans DDR2 de DSP de calcul qui réalise un deuxième traitement de donnée (données pointées à srcc)
- DST1(figure 5) présente l'adresse mémoire dans DDR2 de DSP de destination (DSP de supervision) qui affiche les traitement et les résultats transférées par SRIO de cœur 1 de DSP de calcul.
- DST2(figure 6) présente l'adresse mémoire dans DDR2 de DSP de destination (DSP de supervision) qui affiche les traitement et les résultats transférées par SRIO de cœur 2 de DSP de calcul.

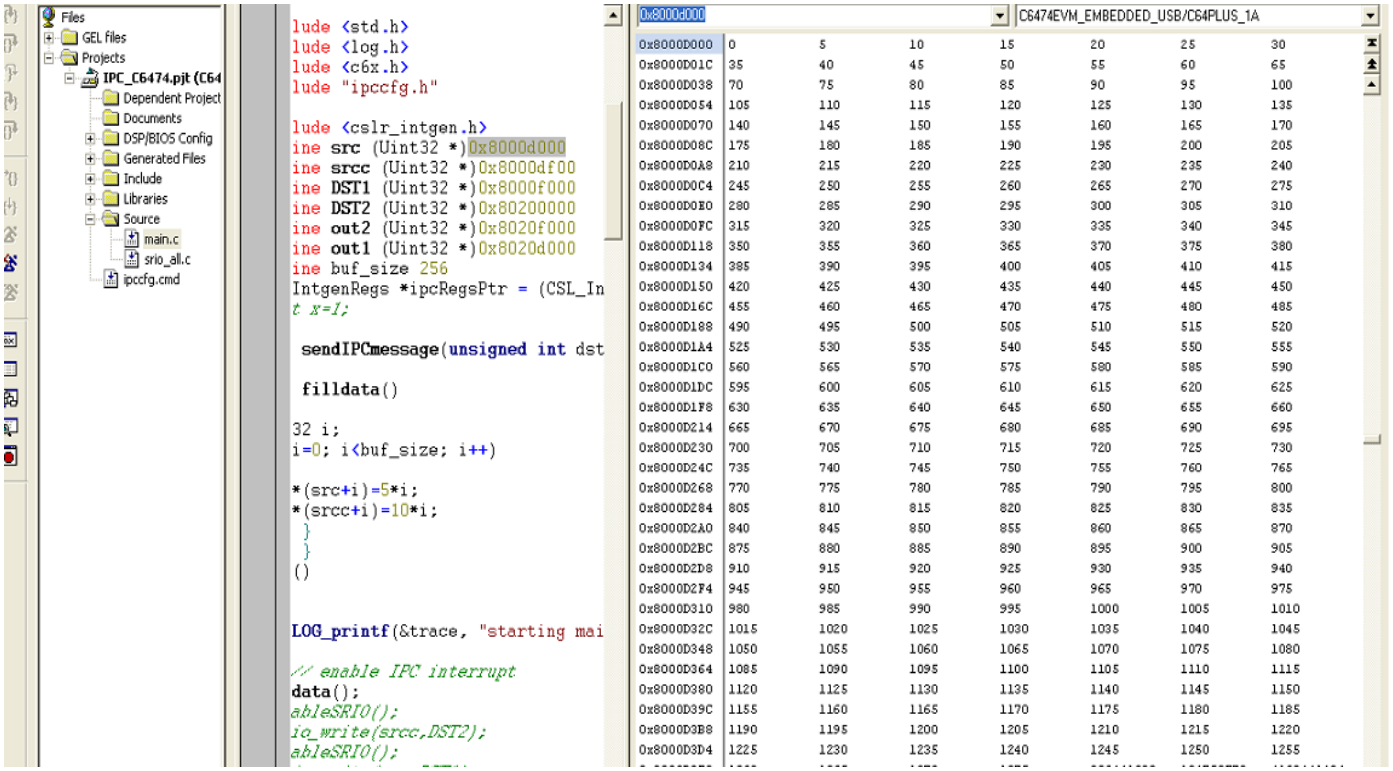


Figure 1 : les données à partir de l'adresse src

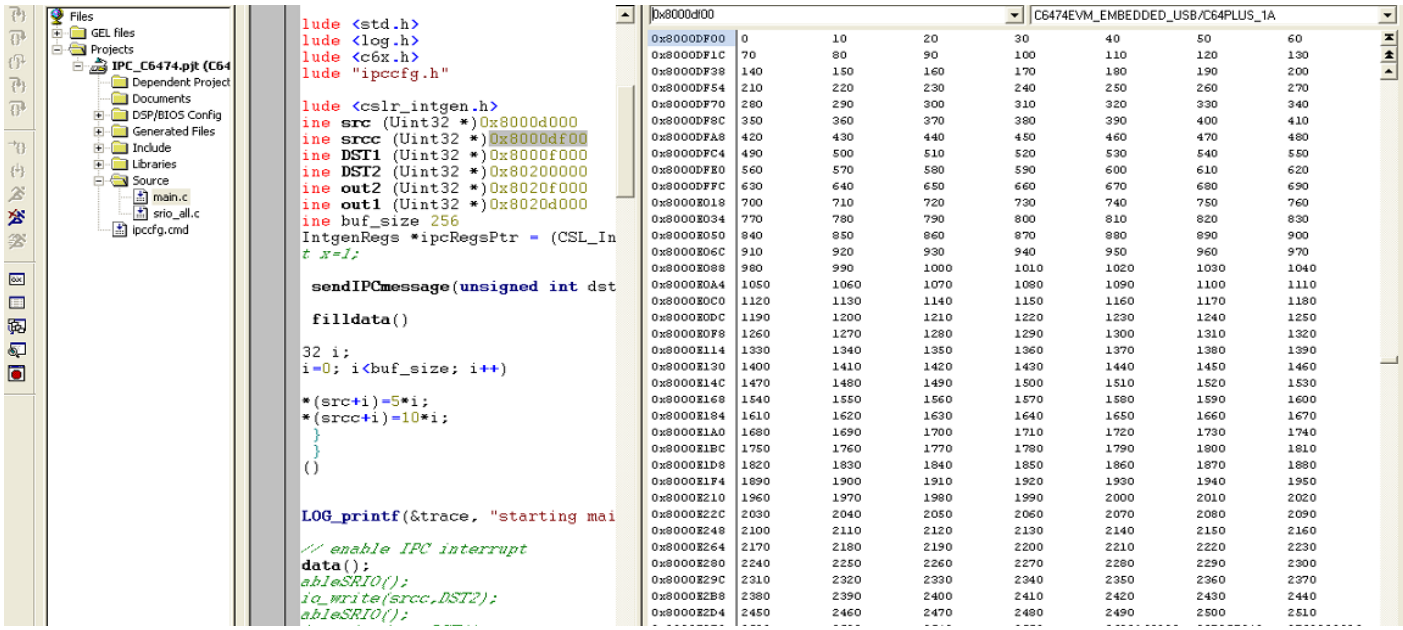


Figure 2 : les données à partir de l'adresse srcc

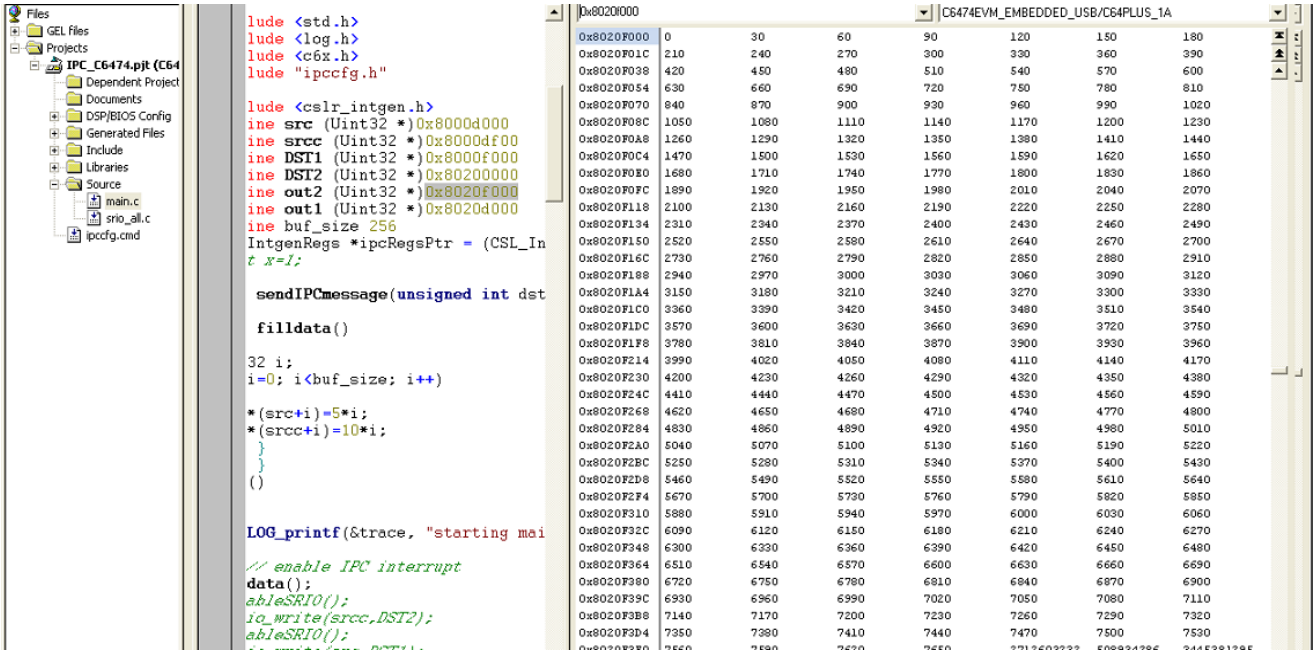


Figure 3 : les calculs de cœur 1(out1)

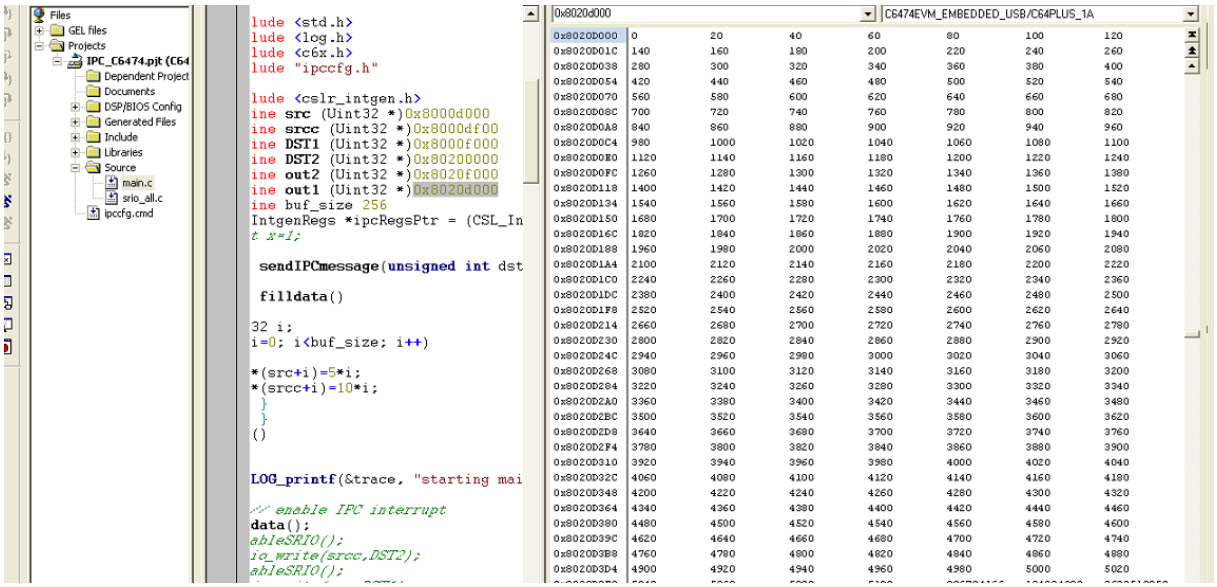


Figure 4 : les calculs du cœur 2 (OUT2)

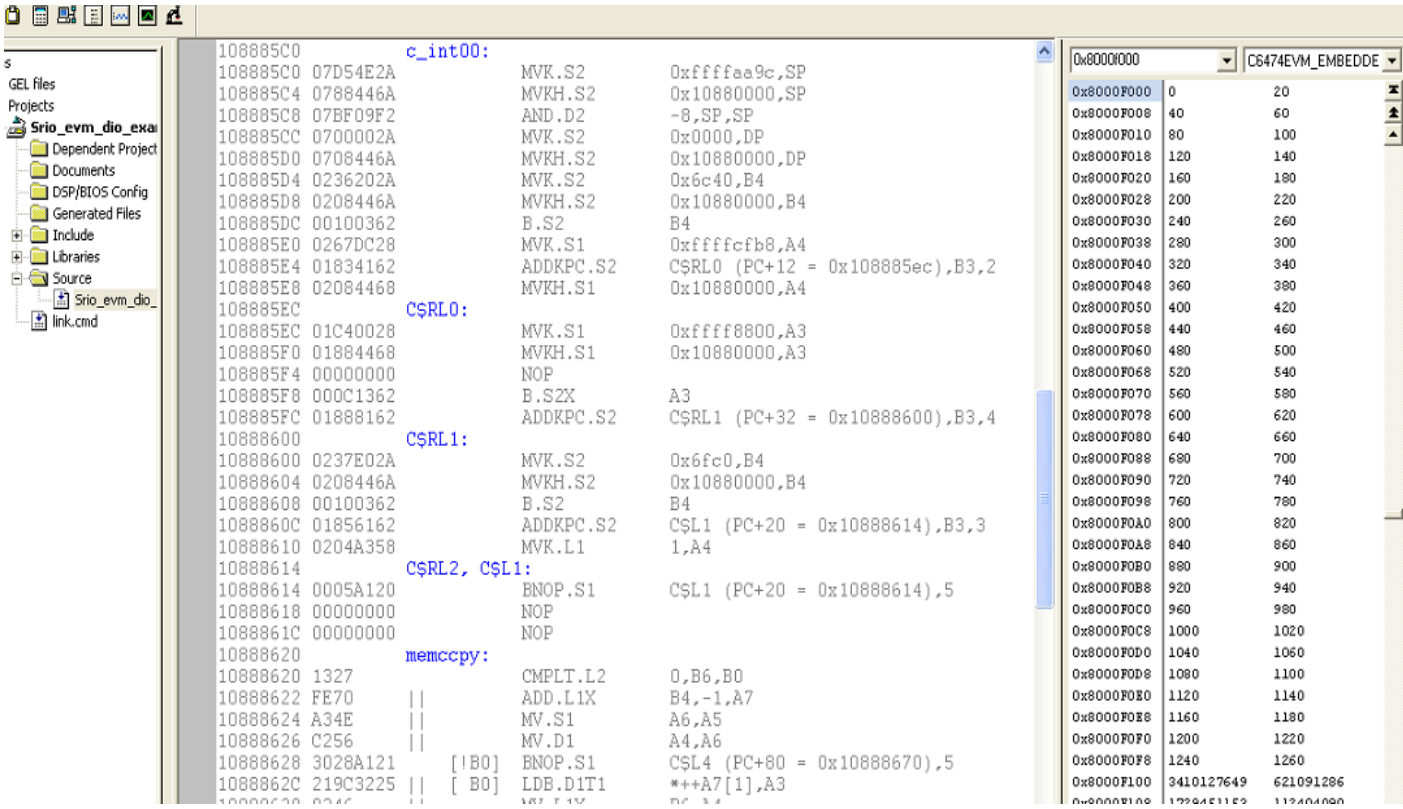


Figure 5: les données en DST1

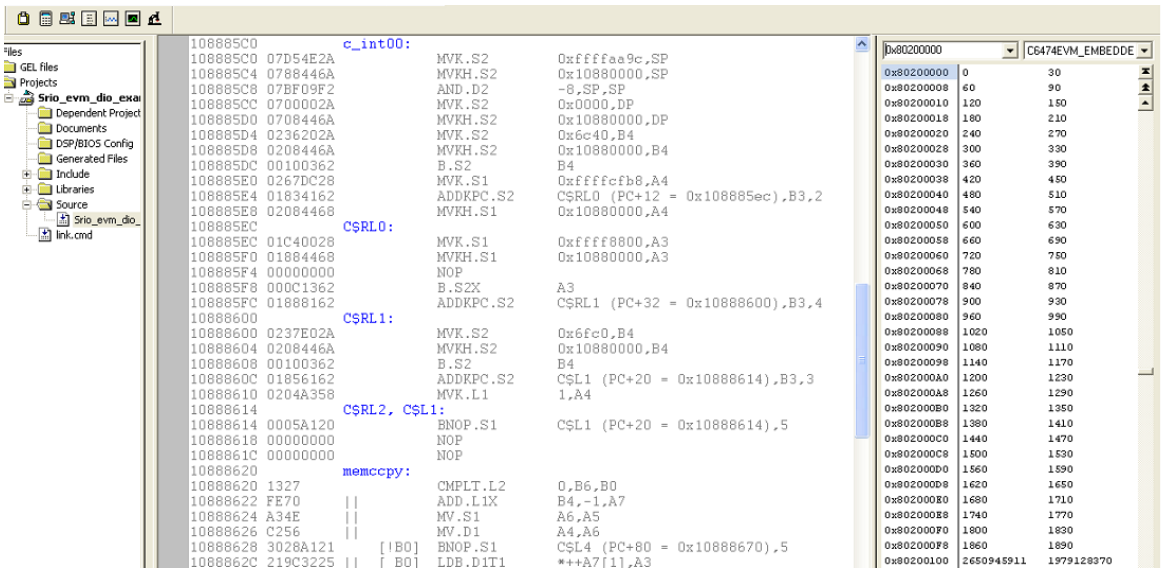


Figure 6 : les données en DST2

5. conclusion

Cette application montre la capacité de TMS320C6474 d'exécuter des programmes et de réaliser la communication inter-cœurs via les mécanismes IPC, sémaphore, DSP / BIOS et SRIO. Dans la suite de cette application, on essaierait d'afficher les résultats dans une page HTML dans le centre de supervision à travers le serveur http développé auparavant pour le DSP de supervision dont son cœur 0 tourne sur le noyau Linux générée dans la première partie de ce projet.

Annexe 9 : Portage de serveur http, compilation et exécution

Introduction:

Un serveur HTTP ou daemon HTTP ou HTTPd (HTTP daemon) ou (moins précisément) serveur Web, est un logiciel servant des requêtes respectant le protocole de communication client-serveur Hypertext Transfer Protocol (HTTP), qui a été développé pour le World Wide Web.

Le dispositif de compilation dans le répertoire *O2FE* permet de compiler les processus de supervision au niveau du *Host* pour différentes architectures (*Linux*, *ColdFire*, etc.). comme on a fait avec PTEMPO et EXPSVRPT, on exposera le chemin de compilation de *httpserver* pour le processeur *ColdFire*, qui a été légèrement modifié pour être compilé pour le processeur *C64x+*.

Cette partie illustre donc la procédure de compilation et d'exécution de serveur http ainsi un test son fonctionnement pour les deux architectures le PC dual cœurs et pour EVM6474

1. **Compilation**

1.1 code source

Le code source qu'on va utiliser, se trouve avec les produits Octane sous le répertoire :

/le/chemin/ vers/ le/ répertoire/o2fe/pgm/HTTPserver

Ou

$(OCTANE_DIR)/pgm/HTTPserver.

Le makefile utilisé pour la compilation pour le Coldfire est: **pgm / HTTPServer / Dbg_HTTPserver_Coldfire / makefile**. La cible en question est "all".

Ce **makefile** est généré automatiquement par Eclipse et utilise d'autres makefiles au niveau bas. Mais on peut compiler par des lignes de commandes (à l'extérieur de l'environnement éclipse). Ces makefiles vont être changés pour que le serveur http se compile pour le PC dual cœurs et EVM6474 respectivement.

1.2 Compilation pour pc dual core

Pour compiler serveur http pour le pc, on doit tout simplement appeler le compilateur de pc : gcc dans différentes **makefiles**:

1) dans le **makefile** localisé à :
   pgm / HTTPServer / Dbg_HTTPserver_Coldfire / makefile
   On replace cette ligne (36)

m68k-linux-gnu-gcc -mcpu=5485 -o"HTTPserver.exe" $(OBJS) $(USER_OBJS) $(LIBS)

Par:

gcc –Wall –O2 –o "HTTPserver.exe" $(OBJS) $(USER_OBJS) $(LIBS)

2) Dans le fichier :
   pgm / HTTPServer / Dbg_HTTPserver_Coldfire / pgm/c/subdir.mk
   On replace ces lignes (41):

/home4/octmgr/coldfire/freescale-coldfire-4.1/bin/m68k-linux-gnu-gcc -mcfv4e -DOCTANE_SYSTEM=LINUX -DGSOAP=1 -DLINUX=1 -D__USE_LOGFILE__=0 -DPRINTF_DEBUG=0 -DSSI_VERSION=101 -I../pgm/h -I../pgm/gen -I../pgm/import -g -w -c -fmessage-length=0 -MMD -MP -MF"$(@:%.o=%.d)" -MT"$(@:%.o=%.d)" -o"$@" "$<"
@echo 'Finished building: $<'
@echo ' '

Par:

gcc –Wall –O2 -DOCTANE_SYSTEM=LINUX -DGSOAP=1 -DLINUX=1 -D__USE_LOGFILE__=0 -DPRINTF_DEBUG=0 -DSSI_VERSION=101 -I../pgm/h -I../pgm/gen -I../pgm/import -g -w -c -fmessage-length=0 -MMD -MP -MF"$(@:%.o=%.d)" -MT"$(@:%.o=%.d)" –o "$@" "$<"
@echo 'Finished building: $<'
@echo ' '

La même chose pour les autres fichiers **subdir.mk** dans les répertoires :
pgm / HTTPServer / Dbg_HTTPserver_Coldfire / pgm/gen/subdir.mk
pgm / HTTPServer / Dbg_HTTPserver_Coldfire / pgm/gsoap/subdir.mk
pgm / HTTPServer / Dbg_HTTPserver_Coldfire / pgm/cgi/subdir.mk
pgm / HTTPServer / Dbg_HTTPserver_Coldfire / pgm/ws/factory/subdir.mk

On remplace:
/home4/octmgr/coldfire/freescale-coldfire-4.1/bin/m68k-linux-gnu-gcc -mcfv4e
Par:
gcc –Wall –O2
Avant la compilation, on doit remplacer **# include "VERSION.h"** dans le fichier **'global_msg.h'** dont le répertoire est **/home/coralie/o2fe/pgm/HTTPserver/pgm/cgi**
Par **# include "version.h".**

Après, dans le **makefile : / home/coralie/o2fe/pgm / HTTPServer / Dbg_HTTPserver_Coldfire / makefile**, on va remplacer la ligne de commande qui exécute la cible : **post-build**

Post-build :

cp HTTPserver.exe ../sbin/. && cp HTTPserver.exe /octane/bin/coldfire

par:

Post-build :

cp HTTPserver.exe ../sbin/. && cp HTTPserver.exe /home/coralie/o2fe/bin/coldfire

Pour compiler le serveur http:

$ cd pgm / HTTPServer / Dbg_HTTPserver_Coldfire
$ make all

L'exécutable est **HTTPserver.exe** doit être en: / **HTTPServer / Dbg_HTTPserver_Coldfire /**

Pour exécuter **HTTPserver.exe**, on tape **./HTTPserver.exe –help** cela permet d'afficher les options possibles :

```
[portagesv@localhost Dbg_HTTPserver_Coldfire ]$./HTTPserver.exe  --help
usage:  ./HTTPserver.exe [-C configfile] [-D] [-p port] [-d dir] [-dd data_dir] [-c cgipat] [-u user]
[-h hostname] [-r] [-v] [-l logfile] [-i pidfile] [-T charset] [-P P3P] [-M maxage]
```

1.3 Compilation for EVM6474 :

Nous allons procéder de la même manière comme le cas de PC mais cette fois ci, on va choisir le compilateur de EVM6474, et donc on va remplacer pour tous les fichiers changés précédemment ; **gcc –Wall –O2** par:

/home/coralie/noyau_linux/build_dir/bin/c64xplus-linux-gcc -Wall -O2 -Wl,-ar

Par exemple on remplace:

/home4/octmgr/coldfire/freescale-coldfire-4.1/bin/m68k-linux-gnu-gcc -mcfv4e -DOCTANE_SYSTEM=LINUX -DGSOAP=1 -DLINUX=1 -D__USE_LOGFILE__=0 -DPRINTF_DEBUG=0 -DSSI_VERSION=101 -I../pgm/h -I../pgm/gen -I../pgm/import -g -w -c -fmessage-length=0 -MMD -MP -MF"$(@:%.o=%.d)" -MT"$(@:%.o=%.d)" -o"$@" "$<"
@echo 'Finished building: $<'

```
	@echo ' '
	Par :
/home/coralie/noyau_linux/build_dir/bin/c64xplus-linux-gcc -Wall -O2  -Wl,-ar -
DOCTANE_SYSTEM=LINUX -DGSOAP=1 -DLINUX=1 -D__USE_LOGFILE__=0 -
DPRINTF_DEBUG=0 -DSSI_VERSION=101 -I../pgm/h -I../pgm/gen -I../pgm/import -g -w -c -
fmessage-length=0 -MMD -MP -MF"$(@:%.o=%.d)" -MT"$(@:%.o=%.d)" –o "$@" "$<"
	@echo 'Finished building: $<'
	@echo ' '
```

Remarque :

Il faut laisser un espace entre -o and ˝$@" dans tous les fichiers **subdir.mk** et entre –o et **"HTTPserver.exe"** dans le makefile.

Avec ces changements, la compilation n'est pas faite à cause d'erreurs de **redefinition** dans la majorité des programmes c comme **sStml** pour les tous codes c et **t_ssi_buf_mgt** pour **ssi.c** et **ssi_mem.c**. Une autre erreur est celle de **undefined symbole** de **_isnan** dans le fichier **stdsoap2.c** comme il est vu en compilant avec **make all**.

Pour remédier à ces erreurs, on peut faire ces changements dans les codes sources :

stdsoap2.c

On ajoute d'abord cette définition pour _isnan:

```
double n ;
#ifndef soap_isnan
# ifdef HAVE_ISNAN
#define soap_isnan(n) isnan(n)
#else
#define soap_isnan(n) (0)
#endif
#endif
```

Ce code est pris de **stdsoap2.h**. Ce premier changement a permet d'éliminer l'erreur de « **undefined symbol _isnan »**.

1) **main.h**

l'erreur de redéfinition de **"sStml"** dans la majorité de programmes de code source de serveur http, peut être résolue en ajoutant **extern** à **struct sStml** . donc dans le fichier **main.h** localisé à **HTTPserver/pgm/h à la ligne 306,** on remplace: **struct sStml** par **extern struct sStml**.

2) **ssi.c**

L'erreur de redéfinition de **"t_ssi_buf_mgt"** dans **ssi.c** et **ssi_mem.c** **localisé à** **HTTPserver/pgm/cgi** peut être résolue en ajoutant **extern** à la ligne 134:

```
extern
        ssi_thread_buf_mgt t_ssi_buf_mgt[NUM_THREADS]
```

Dans le même fichier, il y a un problème d'organisation de code à la ligne 390:

Code origine:

```
<HR><H2>%s</H2>\The requested server-side-includes filename, %s,\tried to use the
directive %s with an unknown tag, %s.\
<HR>", title, filename, directive, tag ));
   }
        else
        {
/*              printf( "\
The requested server-side-includes filename, %s,\
tried to use the directive %s with an unknown tag, %s.\
\n", filename, directive, tag );*/
                SOAP_PRINTF( print_string, current_thread_index, ( print_string, "\The
requested server-side-includes filename, %s,\tried to use the directive %s with an unknown
tag, %s.\
\n", filename, directive, tag ));
        }
}
```

Le code après organisation de code (retour à la ligne "\") :

```
<HR><H2>%s</H2>\
The requested server-side-includes filename, %s,\
tried to use the directive %s with an unknown tag, %s.\
<HR>", title, filename, directive, tag ));
   }
        else
        {
/*              printf( "\
The requested server-side-includes filename, %s,\
tried to use the directive %s with an unknown tag, %s.\
\n", filename, directive, tag );*/
                SOAP_PRINTF( print_string, current_thread_index, ( print_string, "\
The requested server-side-includes filename, %s,\
tried to use the directive %s with an unknown tag, %s.\
\n", filename, directive, tag ));
```

```
        }
}
```

Avant la compilation, on doit remplacer **# include "VERSION.h"** dans le fichier **'global_msg.h'** dont le chemin est **/home/coralie/o2fe/pgm/HTTPserver/pgm/cgi**
Par **# include "version.h".**

Après, dans le makefile**: / home/coralie/o2fe/pgm / HTTPServer / Dbg_HTTPserver_Coldfire / makefile**, on va remplacer la ligne de commande qui exécute la cible **post-build**

Post-build :

cp HTTPserver.exe ../sbin/. && cp HTTPserver.exe /octane/bin/coldfire

par:

Post-build :

cp HTTPserver.exe ../sbin/. && cp HTTPserver.exe /home/coralie/o2fe/bin/coldfire

Maintenant, on peut compiler avec la commande : make all.
L'exécutable HTTPserver.exe se trouve en **HTTPServer / Dbg_HTTPserver_Coldfire /**

Pour l'exécuter, comme il est vu pour le pc, on utilise **./HTTPserver.exe** et pour avoir plus d'information : **./HTTPserver.exe --help**

```
/home # ./HTTPserver.exe  --help
usage:  ./HTTPserver.exe [-C configfile] [-D] [-p port] [-d dir] [-dd data_dir] [-c cgipat] [-u user]
[-h hostname] [-r] [-v] [-l logfile] [-i pidfile] [-T charset] [-P P3P] [-M maxage]
/home #
```

2) **tester le fonctionnement de serveur HTTP**

Dans le répertoire : **pgm / HTTPServer / sbin** (et les sous répertoires), il y a des fichiers importantes de base pour l'exécution:

HTTPserver.cfg (fichier de configuration)

pgm / HTTPServer / sbin / index.html (page HTML par défaut)

Ce répertoire contient aussi le fichier **background.jpg**.

On doit d'abord faire une copie de répertoire de **sbin** qui contient tous les pages web de serveur et aussi le fichier de configuration qui se trouve au répertoire dont le chemin est: **/ home/coralie/o2fe/pgm/HTTPserver** dans les fichiers systèmes de noyau linux qu'on a créé pour le EVM6474.

Pour tester le serveur HTTP pour EVM6474 ou même pour le PC dual core, il suffit donc de lancer la commande : **./ HTTPserver.exe.** Puis on tape tout simplement dans un navigateur l'adresse IP de serveur et la page Web qu'on veut afficher ou qu'on veut accéder comme montré par l'exemple suivant :

http://adress_IP_de_serveur:num_de_port/page.html

Le numéro de port par défaut est 80 pour le serveur http qu'on peut le changer via le fichier de configuration. Le changement de numéro de port est nécessaire dans le cas où un autre serveur http occupe se port comme il est était le cas pour notre application.

EXEMPLE: http://192.168.1.1:1240/index.html

Cela doit afficher une page html ayant comme nom index.html (la page d'accueil de CCC)

Références

[1] Rapport de PFE. Youssef EL GMILI, Master SMTII 2009, FSTF.

[2] http://b.l.free.fr/Page3.html

[3] www.ti.com

[4] TMS320C6474 DSP

DDR2 Memory Controller
User's Guide sprug19

[5] TMS320C6474 DSP
64-Bit Timer User's Guide

[6] DOCUMENT DE CONCEPTION DU LOGICIEL (SDD SSSI DEMSV) POUR LE SSSI DEMSV DU SYSTEME DAMB M3R (document de Thales)

[7] SOFTWARE DESIGN DESCRIPTION (SDD) FOR OCTANE CSCI COMPONENT CCC HOST OF SYSTEM M3R Demonstrator (document de Thales)

[8] http://linux.maruhn.com/sec/dhrystone.html

[9] http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=MCF527X&fsrch=1

***[10]http://www.freescale.com/webapp/sps/site/taxonomy.jsp?code=PCPPCMPC85XX#*53**

[11] TMS320C6474 DSP
Antenna Interface User's Guide

[12] http://linux.maruhn.com/sec/ttcp.html

[13] http://sourceforge.net/projects/iperf/

[14] http://alasir.com/software/ramspeed/

[15] http://icl.cs.utk.edu/projects/llcbench/cachebench.html

[16] http://www.tux.org/~mayer/linux/bmark.html

[17] http://www.bitmover.com/lmbench/

[18] TMS320C6474 DSP

Inter-Core Communication on TMS320C6474 sprab20.pdf

[19] TMS320C6474 DSP

Semaphore User's Guide sprug14.pdf

[20] TMS320C6474 DSP

How to Approach Inter-Core Communication on TMS320C6474 sprab25.pdf

[21] TMS320C6474 DSP

Multicore Programming Guide sprab27a.pdf

[22] TMS320C6474 DSP

TMS320C6474 Multicore Digital Signal Processor tms320c6474.pdf

[23]http://e2e.ti.com/support/dsp/tms320c6000_high_performance_dsps/f/112/p/34382/123512.aspx

[24]http://focus.ti.com/dsp/docs/dspsupporttechdocsc.tsp?sectionId=3&tabId=409&familyId=1635&abstractName=sprab20

[25] http://www.planete-sciences.org/robot/wikibot/index.php/DSP

[26] TMS320C6474 DSP

Serail RapidIO(SRIO) User's Guide

[27]http://www.cpu-world.com/benchmarks/

[28] TMS320C6474 DSP

Sofware-Programmable Phase-Locked Loop(PLL) Controller User's Guide

[29] TMS320C6474 DSP

Ethernet Media Access Controller (EMAC)/Management Data Input/output(MDIO) User's Guide

[30] TMS320C6474 DSP

General-Purpose Input/output(GPIO) User's Guide

[31] TMS320C6474 DSP

Inter-Integrated Circuit(I2C) Module User's Guide

[32] TMS320C6474 DSP

Bootloader User's Guide

[33] TMS320C6474 DSP
Turbo-Decoder Coprocessor2 (TCP2)

[34] TMS320C6474 DSP
Viterbe-Decoder Coprocessor2 (VCP2)

[35] TMS320C6474 DSP
Mltichannel Buffered Serial Port(McBSP)

[36] SPRU509h.pdf

[37] TMS320C64x/C64x+ DSP
CPU and Instruction Set
User's Guide spru732h.pdf

[38] DSP Architecture et Application.pdf

[39] Méthodologie de conversion automatique en virgule Fixe pour les applications de traitement du signal.pdf
D.MENARD
Équipe de recherché R2D2-IRISA/INRIA
ENSSAT-Université de Rennes1

[40] TMS320C6474 Mezzanine EVM Board
Technical Reference

[41] FAQ Texascommunity

[42] TMS320C6000 Optimizing Compiler v 6.1
User's Guide spru187o.pdf

[43] http://www.snmp.com/protocol/ietf_activities.shtml

[44]TMS320C6000 DSP
Enhanced Direct Memory Access (EDMA) Controller Reference Guide
[45] TMS320C6000 DSP
CacheUser's Guide

Résumé

Ce rapport porte sur ***«l'étude & développement de logiciels de supervision Radar sur Linux pour la nouvelle génération de DSP développé par Texas Instrument »***.

Dans une première partie, j'ai étudié les performances du noyau Linux généré à partir des sources Linux développées par la société Virtual Logix. Ces tests ont permis de positionner le TMS320C6474 par rapport aux autres architectures et d'apprécier les gains par rapport à l'architecture ColdFire utilisée actuellement.

Ensuite j'ai porté le Serveur http pour le TMS320C6474 à partir des sources utilisées pour l'architecture de ColdFire. Ce serveur servira à afficher les pages html situées avec les fichiers systèmes du noyau linux tournant sur le DSP, à partir d'un central superviseur via Ethernet ainsi via les protocoles réseaux tels que NFS et Telnet.

Enfin j'ai étudié la communication inter cœur et inter DSP pour les applications de la supervision des Radars numériques.

Ce stage a été concluant et m'a poussé à intégrer le monde des DSP et les systèmes embarqués de façon générale.

Abstract

This report focuses on ***«the study and development of radar monitoring software on Linux for the new generation of DSP developed by Texas Instruments»***.

In the first part, I studied the performances of Linux kernel generated from Linux sources, developed by the company Virtual Logix. These tests were used to compare the TMS320C6474 to other architectures and to appreciate the gains from the ColdFire architecture currently used.

Then I succeeded the portage of HTTP server for the TMS320C6474 from sources used for the ColdFire architecture. This server will be used to display html pages located with the kernel Linux systems running on the DSP from a central supervisor via Ethernet and network protocols such as NFS and telnet.

Finally I studied inter-cores and inter-DSPs communications for applications of digital Radar supervision.

The training was successful and prompted me to join the world of DSP and embedded systems in general fashion.