Notice Bibliographique

Auteur :	Rachid ALILOU
Année :	2012
Établissement :	Faculté des sciences et Techniques de Fès (3éme année du cycle d'ingénieur)
Spécialité :	Systèmes Électroniques et Télécommunications
Établissement d'accueil :	THALES AIR SYSTEMS & MASCIR Embedded Systems
Cadre du rapport :	Projet de fin d'études

Titre du projet de stage :

Etude de faisabilité d'un multiboot sous Linux sur DSP multicores et de performances de l'interface PCIe dans un contexte de supervision Radar.

Encadrement entreprise :

Mr. Noreddine KACHKACHI	Ingénieur MASCIR Embedded Systems
Mr. François BOURZEIX	Manager Equipe MASCIR Embedded Systems
Mr. Philippe LE GALL	Manager Equipe THALES AIR SYSTEMS

Encadrant pédagogique :

Pr Mouhcine RAZI	Professeur à la FST de Fès

Date de début et de fin du stage:

Début :	le 01 mars 2012
Fin :	le 30 juin 2012

Dédicaces

A la mémoire de mon cher père

Qui a tant attendu ce jour

Que ton âme repose en paix

A ma chère mère

Qui m'a comblé de son soutien et m'a voué un amour inconditionnel

Puisse Dieu, le très haut, t'accorder bonheur et longue vie

A mes frères et à ma chère sœur

A tous mes amis et tous ceux qui me sont chers

Remerciements

Je tiens à remercier dans un premier temps, toute l'équipe pédagogique de la FST de Fès et les intervenants professionnels responsables de la formation Systèmes Electroniques et Télécommunications, pour avoir assuré le bon déroulement de celle-ci.

Je remercie également mon encadrant, le professeur **Mouhcine RAZI**,, pour l'aide et les conseils concernant les missions évoquées dans ce rapport, qu'il m'a apporté lors des différents suivis.

Je tiens à remercier tout particulièrement et à témoigner toute ma reconnaissance aux personnes suivantes, pour l'expérience enrichissante et pleine d'intérêt qu'elles m'ont fait vivre durant ces quatre mois :

Monsieur **François BOURZEIX**, manager du département Embedded Systems à MASciR, pour son accueil et la confiance qu'il m'a accordé dès mon arrivée.

Mes tuteurs de stage : Monsieur **Noreddine KACHKACHI**, ingénieur Systèmes Embarqués à MASciR et Monsieur **Philippe LE GALL**, manager d'équipe THALES AIR SYSTEMS, pour m'avoir accueilli, facilité mon intégration, pris en charge, confié des tâches, fait confiance, conseillé, encouragé…

Tables des matières

Liste des figures et des tableaux

Liste des abréviations

FTP : File Transfer Protocol

NFS : Network File System

TFTP : Trivial File Transfer Protocol

ELF : Executable and Linkable File

JSON : Javascript Object Notation

DSP : Digital Signal Processor

MAD : Multicore Application Deployment

MULAC : Multiplication and accumulation

FPGA : Field Programmable Gate Array

PCIe : Peripheral Component Interconnect express

CCS : Code Composer Studio

EMIF : External Memory Interface

JTAG : Join Test Action Group

IBL : Intermediate Bootloader

MAP : Multicore Application Prelinker

DDR : Double Data Rate

TLPs : Transaction Layer Packets

DLLPs : Data Link Layer Packets

PLPs : Physical Layer Packets

BAR : Base Address Register

API : Application Programming Interface

Résumé

Dans le cadre de développement de nouvelle génération des radars numériques et à usages multiples, THALES AIR SYSTEMS souhaite porter un système d'exploitation embarqué, fiable et performant sur le DSP TMS320C6678 de Texas Instruments, la mise en œuvre d'une architecture de déploiement du noyau Linux au sein des huit cœurs du même DSP ainsi que l'étude des communications PCIe entre le FPGA Cyclone IV d'Altera et une architecture PC.

La première phase consiste à porter le système d'exploitation Linux embarqué sur le TMS320C6678 et à répondre à la problématique du déploiement et au boot du noyau Linux sur l'ensemble des cœurs du DSP dans un contexte de supervision RADAR. La deuxième phase a pour but d'étudier les performances des communications via le protocole de communication série à haut débit "PCI Express" (PCIe) entre un FPGA chargé du prétraitement des signaux dans la chaîne RADAR d'une part et d'un PC chargé du contrôle du FPGA de l'autre part.

Introduction générale

De nos jours, l'industrie de la défense se tourne vers des radars à usages multiples qui peuvent répondre aux besoins du contrôle aérien, la sécurité intérieure, et l'observation météorologique. Ces radars sont l'avenir de l'industrie et seront entièrement numériques.

Le besoin en puissance de calcul pour les traitements d'informations, sans cesse croissants, conduit pour la quasi-totalité des fonctions, à développer des machines comportant plusieurs processeurs de même type ou de types différents opérants de manière concourante, on parle alors de machines parallèles.

Vue la nature des traitements radar, basée sur le traitement numérique du signal (filtrage, extraction de signaux, etc.), les processeurs les plus adaptés pour construire la chaîne radar sont le DSP (Digital Signal Processor) et le FPGA (Field Programmable Gate Array).

Vu que le radar intègre un grand nombre de microprocesseurs, il est nécessaire qu'un ou plusieurs microprocesseurs se chargent du suivi et de la supervision de l'ensemble du réseau et qu'il y ait un protocole qui permette la communication à haut débit entre les différents composants du système radar.

L'architecture du système radar pose deux problèmes fondamentaux intimement liés :

- La communication entre la chaîne de prétraitement des signaux constitué d'un FPGA et une architecture PC permettant le contrôle des FPGAs.
- Un système d'exploitation pour la supervision des DSPs présents dans la chaîne de traitement des signaux.

Mon projet avait comme objectif de mettre en œuvre des solutions aux problèmes cités ci-dessus. Le projet a été découpé en deux phases principales traitant de l'aspect multiboot sous Linux sur les DSPs multicoeurs de nouvelle génération de Texas Instruments et de la communication entre le FPGA chargé du prétraitement des signaux et une architecture PC permettant le contrôle du FPGA via le protocole PCIe.

La première phase consiste, à porter et à lancer le système d'exploitation embarqué Linux, fiable, performant et adapté à l'architecture du DSP utilisé (C6678 de Texas Instruments) sur les huit cœurs du DSP.

La deuxième phase traite quand à elle de la prise en main du protocole PCIe qui sera utilisé pour la communication des composants embarqués dans le RADAR, puis dans un second temps le développement d'un pilote PCIe et différents scenarios de test sous Linux qui vont nous permettre de tester puis d'analyser les performances de la communication entre le FPGA et le PC en utilisant PCIe.

Chapitre 1 : Contexte du stage

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. 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

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

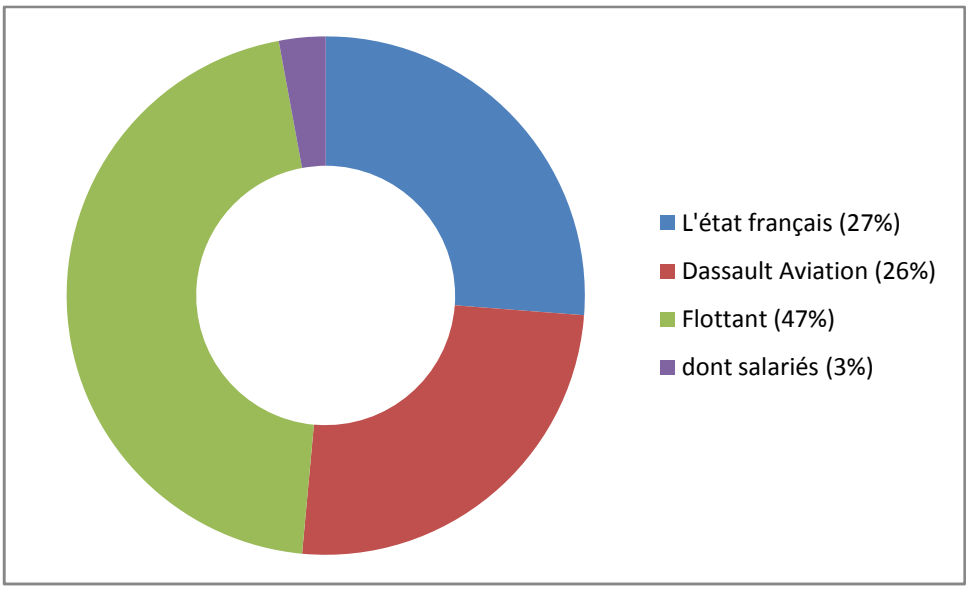


FIGURE 1 : REPARTITION DU CAPITAL DE THALES AU 31 MAI 2009

1.2. THALES Air Systems

La division Air Systems du Groupe Thales conçoit et fournit des solutions globales de sécurité aérienne dans les domaines civil et militaire. Dans le domaine de l'aviation civile, la division développe des systèmes de gestion du trafic aérien, des outils d'aide à l'atterrissage et à la navigation ainsi que des solutions pour la sécurité des zones aéroportuaires. Dans le cadre de ses activités de défense aérienne, la division propose une gamme complète de radars de surface, des systèmes de conduite des opérations aériennes, des solutions de protection du champ de bataille et des sites sensibles ainsi que des systèmes d'armes antiaériens, terrestres et navals. En appui de cette offre, la division propose une gamme complète de services de maintenance, de rénovation et d'extension de durée de vie ainsi que des services d'ingénierie logistique permettant de concevoir dès la phase de développement, la solution de maintenance adaptée au système vendu.

Thales Air Systems est :

- Leader mondial des systèmes de gestion du trafic aérien (hors USA)
- Leader mondial des centres de commandement et de contrôle des opérations aériennes
- Troisième des radars de surveillance naval et terrestre
- Premier fournisseur européen de fusées, d'autodirecteurs de missiles et de munitions guidées

2. Présentation de MAScIR

MAScIR (Moroccan foundation for Advanced Science Innovation and Research) est une fondation à utilité publique créée en 2007 par le gouvernement marocain pour promouvoir la recherche et développement. Son local est sur Rabat Shore Center, Technopolis.

La nanotechnologie, la biomédecine, et les microélectroniques représentent les institutions de recherche de cette fondation :

- MAScIR Micro : créée en 2008 pour la recherche dans le domaine de la microélectronique
- MAScIR bio : travaille dans la biotechnologie des drogues et biocides.
- INanoTech : forme la troisième division de MAScIR dans le domaine de la nanotechnologie

2.1. MAScIR micro

La figure suivante montre la hiérarchie du département Micro de la fondation MAScIR :

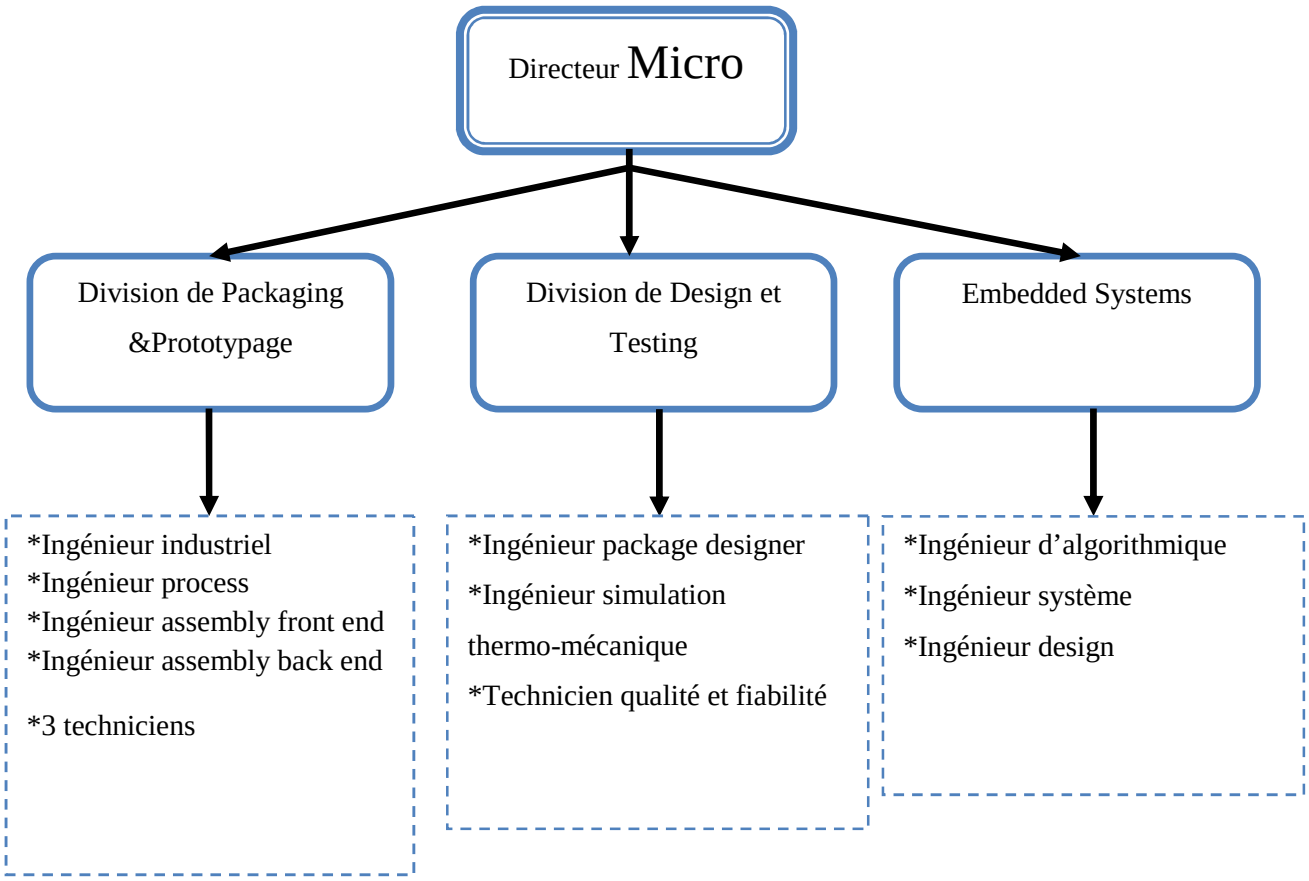


FIGURE 2: HIERARCHIE DU DEPARTEMENT MICRO

MAScIR Micro est un centre d'innovation et développement de technologie dans le domaine de la microélectronique. Il se focalise sur la simulation, les tests, le design, le packaging, la qualification et le prototypage des produits microélectroniques.

MAScIR Micro fournit des services pour des clients industriels, mais elle développe aussi son propre buisness dans les domaines suivants :

- L'intégration et la miniaturisation des systèmes microélectroniques
- L'analyse de fiabilité et défaillance des produits
- Modélisation des systèmes complexes

- Prototypage et industrialisation des produits innovants
- Industrialisation des idées et résultats académiques

MAScIR possède plusieurs laboratoires équipés de technologie avancée :

- Chambre blanche
- Laboratoire optique
- Laboratoire électronique

3. Présentation du Projet

Les nouveaux radars de THALES sont des systèmes distribués contenant un grand nombre de DSPs qui sont chargés de l'exécution des algorithmes de traitement de signal et de calcul. Les DSPs, comme le montre la figure 1, sont liés par des switches en formant un réseau de calcul.

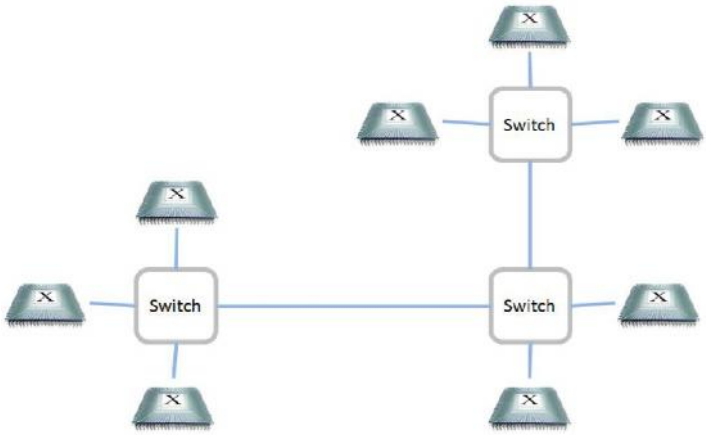


FIGURE 3: RESEAU DE PROCESSEURS

Le nombre important des processeurs de calculs dans le réseau nécessite la présence d'un ou plusieurs processeurs dédiés au suivi et à la supervision des autres unités de calcul (les cibles).

La figure suivante montre l'architecture de supervision :

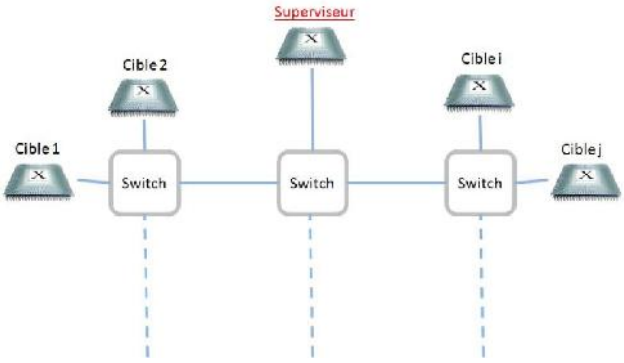


FIGURE 4 : ARCHITECTURE DE SUPERVISION

Une telle architecture de supervision nécessite des communications entre le processeur superviseur et les processeurs cibles, responsables de l'exécution des calculs radar.

Le processeur superviseur, maître, est responsable de charger et lancer les programmes de calculs qui seront exécutés par les processeurs cibles. Après l'exécution des programmes, il devient responsable du suivi de cette exécution (vérifier l'état de fonctionnement du réseau de DSP). Des messages sur l'état de chaque processeur cible sont redirigés vers un écran d'un PC via une liaison Ethernet.

Des communications inter-cœurs du même DSP et avec le PC se trouvant sur le réseau Ethernet nécessitent un système d'exploitation embarqué, fiable, performant et permettant une gestion des ressources physiques (la liaison Ethernet, les interruptions …). Le choix s'est dirigé vers le système d'exploitation embarqué Linux.

La chaîne de prétraitement des signaux intègre un nombre important de FPGAs dont le rôle principal est le filtrage des signaux reçus à partir de la chaîne d'acquisition. Les signaux prétraités sont ensuite envoyer à la chaîne de traitement des signaux à travers une liaison PCIE. Les FPGAs intégrés dans la chaîne de prétraitement seront entièrement contrôlés par un ordinateur distant via une liaison PCIe.

3.1. Cahier des charges

Dans le cadre du développement des radars numériques à usages multiples, THALES AIR SYSTEM et MASCIR EMBEDDED SYSTEM souhaitent évaluer les performances du Linux embarqué sur des processeurs multi-cœurs de Texas Instruments (C6678) ainsi que le protocole de communication PCIE. Le sujet a été divisé en deux phases et un ensemble de tâches a été fixé au départ :

Phase I :

Tâche 0 : Prise en main du DSP, du noyau Linux fourni par Texas Instruments et de l'atelier du développement.

Tâche 1 : Porter le noyau Linux sur un cœur du DSP C6678, vérifier sa mise en route et son fonctionnement.

Tâche 2 : Déployer le noyau Linux sur les huit cœurs du DSP, réaliser une architecture multiboot.

Phase II :

Tâche 0 : Prise en main de la carte d'évaluation FPGA (Cyclone IV d'Altera) et étude du protocole PCIe.

Tâche 1 : Développement d'un pilote PCIe sous Linux pour la carte FPGA.

Tâche 2 : Développement de scenarios de test pour évaluer les performances.

Tâche 3 : Analyse des performances de la communication entre le FPGA et le PC via PCIE.

Tâche 4 : Rédaction des spécifications logicielles et de la documentation technique pour les différentes applications développées.

3.2. Planning du projet

Au cours de la période du stage, un ensemble de points de contrôle a été mis en place pour bien établir un suivi du déroulement des étapes du projet, ainsi que la discussion et la validation des livrables. La figure suivante illustre cette planification

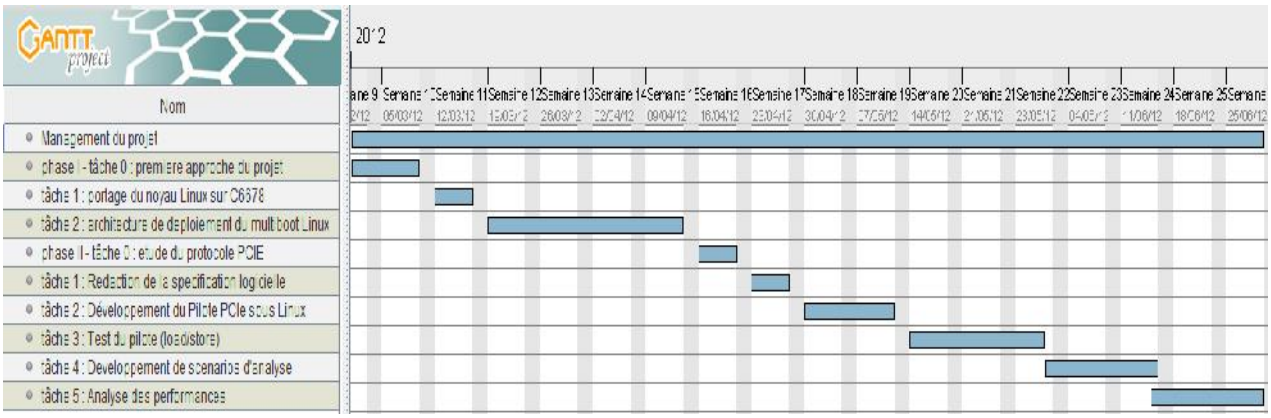


FIGURE 5 : PLANNING DU PROJET

3.3. Eléments d'entrée

THALES et MASCIR ont mis à notre disposition un kit de développement composé de la carte d'évaluation TMS320C6678 EVM Board, de l'outil de développement Code Composer Studio et l'émulateur JTAG XDS560v2 pour la réalisation de la première phase du projet puis d'un kit de développement Cyclone IV GX FPGA et d'un ordinateur de bureau doté d'un processeur Core 2 DUO.

La carte d'évaluation TMS320C6678 comporte un DSP C6678 programmable à l'aide du logiciel Code Composer Studio (CCS). Ce logiciel inclut une suite d'outils utilisés pour développer et débugger les applications ainsi qu'un simulateur.

Le kit d'évaluation Cyclone IV GX comporte un circuit FPGA Cyclone IV et est doté d'un connecteur PCIe x4 qui lui permet d'être connecté à la carte mère de l'ordinateur et ainsi tester les communications entre le FPGA et le PC.

3.4. Eléments de sortie

En retour, la société THALES et la fondation MASCIR souhaitent recevoir un rapport d'étude, de mise en œuvre et de performance pour les tâches citées dans le cahier des charges.

Des rapports d'avancement hebdomadaire « weekly report » décrivant les points abordés au cours de la semaine passée et ceux qui vont être abordés dans la semaine qui suit ont été demandés, ainsi que des présentations quasi mensuelles sur l'état d'avancement du projet.

Les codes sources et programmes développés doivent respecter les règles de codages décrites par THALES, afin de garantir une meilleure lisibilité et une seule structure unifiée. Les programmes doivent être commentés en Anglais et livrés sur une zone de partage sous le serveur FTP de MAScIR.

Chapitre 2 : Multiboot Linux sur DSP C6678

Le but de ce chapitre est de décrire d'une manière générale les processeurs de traitement numérique des signaux (DSP) et présenter spécialement celui de Texas Instruments, le DSP TMS320C6678, leurs caractéristiques, type, architecture et domaine d'application, puis sera présenté l'aspect multiboot sous Linux sur le DSP C6678.

1. Processeur de traitement de signal numérique

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.

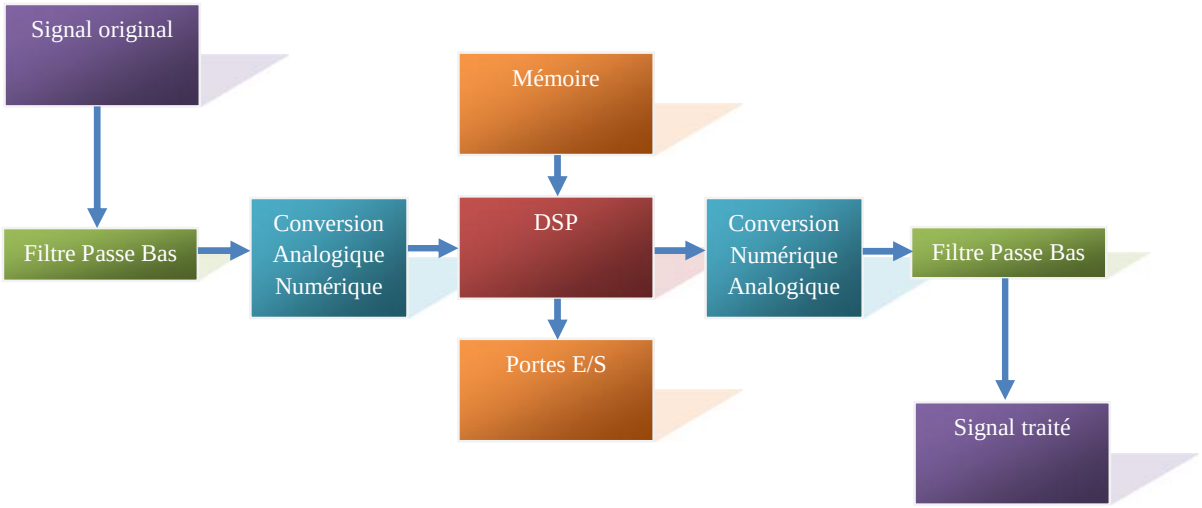


FIGURE 6 : CHAINE COMPLETE TYPIQUE D'UN SYSTEME DE TRAITEMENT NUMERIQUE DE SIGNAL

Certains DSPs 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).

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.

1.1. Différence entre DSP et microprocesseurs ordinaires

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

1.1.1. L'opération MULAC

Le signal se présente sous la forme d'une suite de valeurs numériques discrètes. Cette suite de valeurs (ou échantillons) est apte 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.

En total, il a besoin donc de 80 cycles pour calculer A **[1]**, ce qui rend impossible d'implémenter des algorithmes de traitement de signal temps réel sur ce processeur. Les DSPs sont donc conçus pour effectuer les opérations arithmétiques en un temps très court.

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 TMS320C6678 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 ».

1.1.2. 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'opération simultané 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 un 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.

1.2. Autres Caractéristiques des DSPs

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.
- 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 de l'application, et de la complexité des opérations à effectuer.

2. TMS320C6678

La famille de processeurs la plus répandue actuellement est celle de Texas Instruments, car elle détient environ 70% du marché, les 30% restant sont partagé entre Motorola, Analog Devices, Lucent Technologies, et d'autres.

Les DSPs de TI sont répartis en trois classes :

- La famille hautes performances C6000 formée de C62x, C64x, C66x, et C67x
- La famille C5000 de coût et performances moyennes
- La famille C2000 de faible consommation pour les applications de contrôle

Le tableau suivant regroupe les principales caractéristiques de chaque famille :

Famille	Application	Type	Performances
C66x	Traitement numérique de signal, calcul en entiers/flottants	32 bits virgule fixe/flottante	320 GMACs/s 160 GFLOPs/s
C64x	Traitement numérique de signal, calcul en entiers	32 bits virgule fixe	24000 MIPS
C54x	Applications de télécommunication : terminaux mobiles, voix sur IP…	16 bits virgule fixe	30-200 MIPS
C20x	Applications grand public, appareils photo, contrôleurs de disques durs…	16 bits virgule fixe	20-40 MIPS

Tableau 2 : Principales familles de DSPs de TI

2.1. Caractéristiques

Le DSP C6678 est un DSP à haute performance à virgule fixe et flottante basée sur une architecture multicœur. Intégrant huit cœurs C66x, le DSP fonctionne à une vitesse de 1 GHz par cœur.

L'architecture multicoeurs fournit une plate-forme programmable intégrant divers sous-systèmes, des sous-systèmes de mémoire et des périphériques en plus d'un navigateur multicoeurs (Multicore Navigator) qui permet de gérer les données entre les différents composants du DSP **[2]**.

Le bus Teranet permet un échange rapide de données internes à la puce. Le contrôleur multi cœur de mémoire partagée (MSMC) permet d'accéder quand à lui à la mémoire partagée et à la mémoire externe directement sans passer par le bus externe. La figure 7 montre le schéma fonctionnel du DSP C6678.

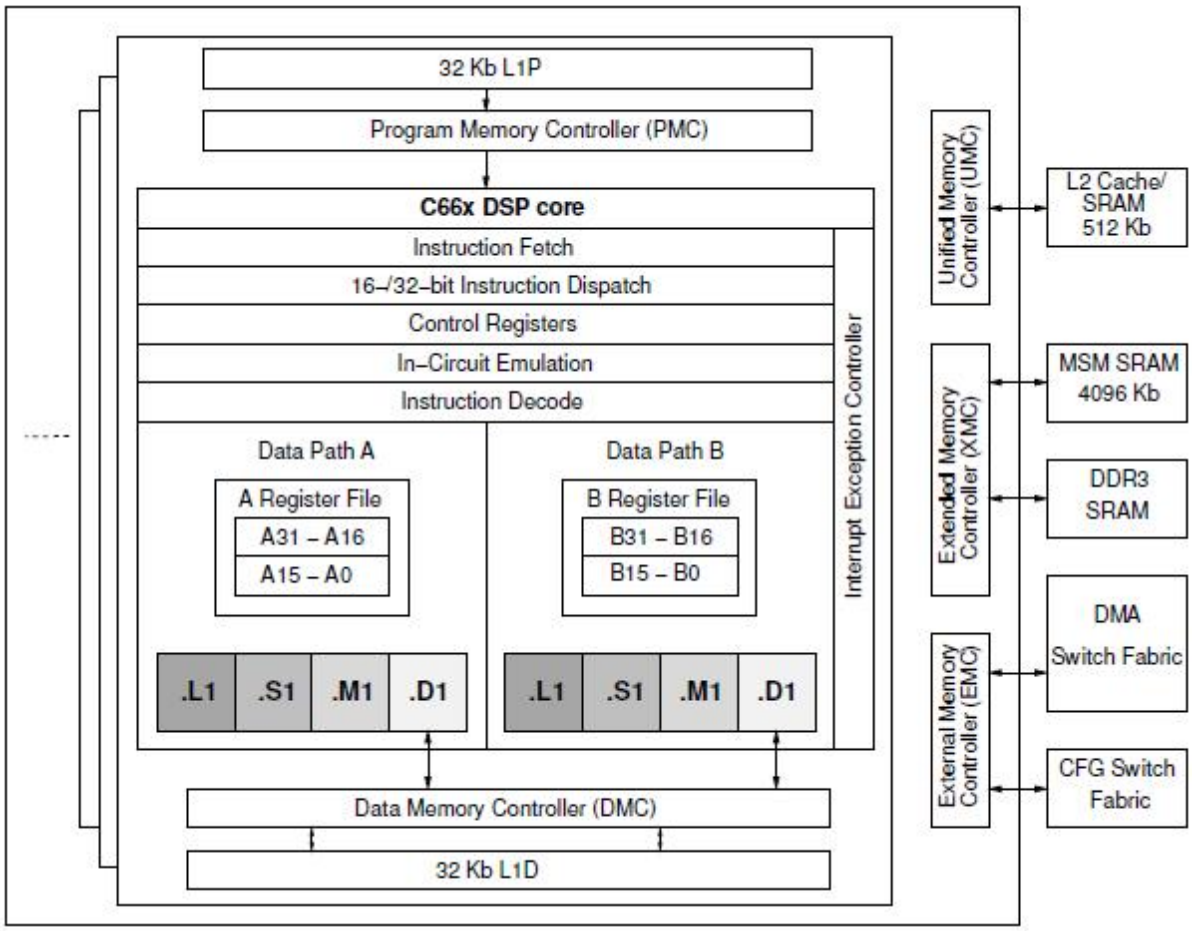


FIGURE 7 : DIAGRAMME FONCTIONNEL DU DSP C6678

2.2. L'architecture

L'architecture du DSP TMS320C6678 est caractérisée par une diversité de composants, permettant un traitement fluide des données, un adressage efficace, et des périphériques de

communication supportant différents protocoles de communication (Serial RapidIO, Antenna Interface, 10M/100M/1G Ethernet).

La figure suivante montre le schéma fonctionnel du C6678 **[3].**

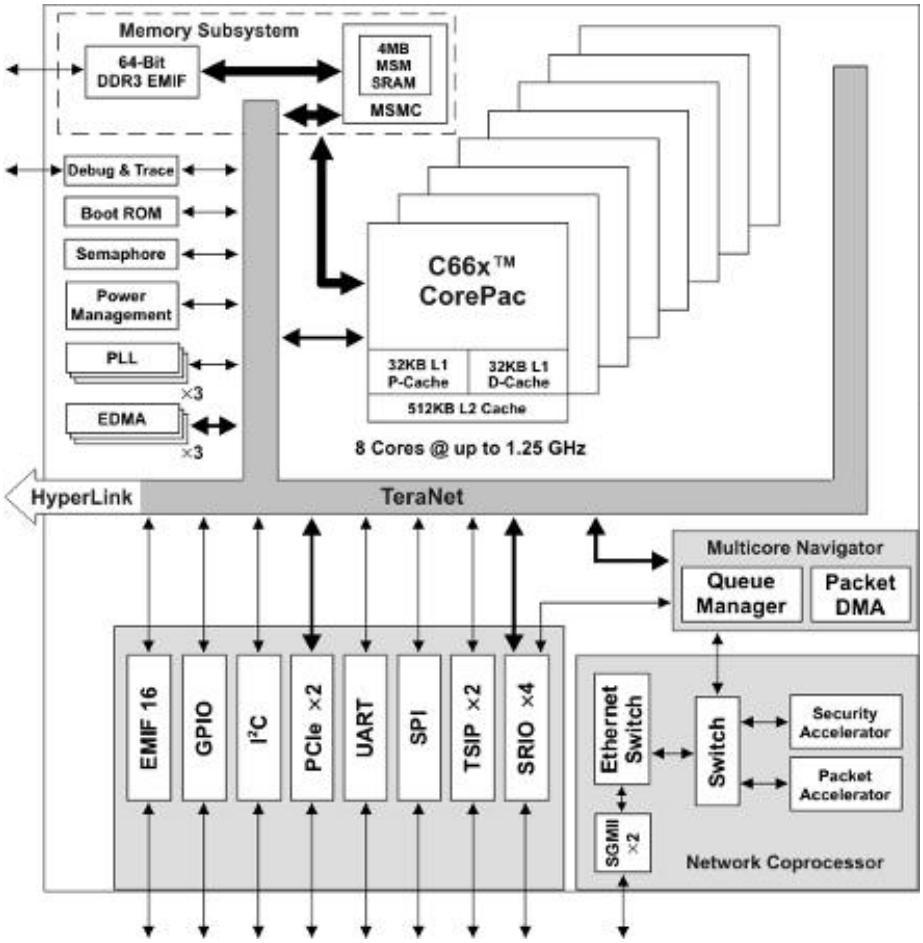


FIGURE 8 : SCHEMA FONCTIONNEL DU CŒUR DSP – C66X

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

Le DSP TMS320C6678 contient huit cœurs DSP C66x, des périphériques de haute vitesse et une grande quantité de mémoire interne. La figure qui suit représente l'architecture interne du cœur C66x du DSP C6678 **[3]**.

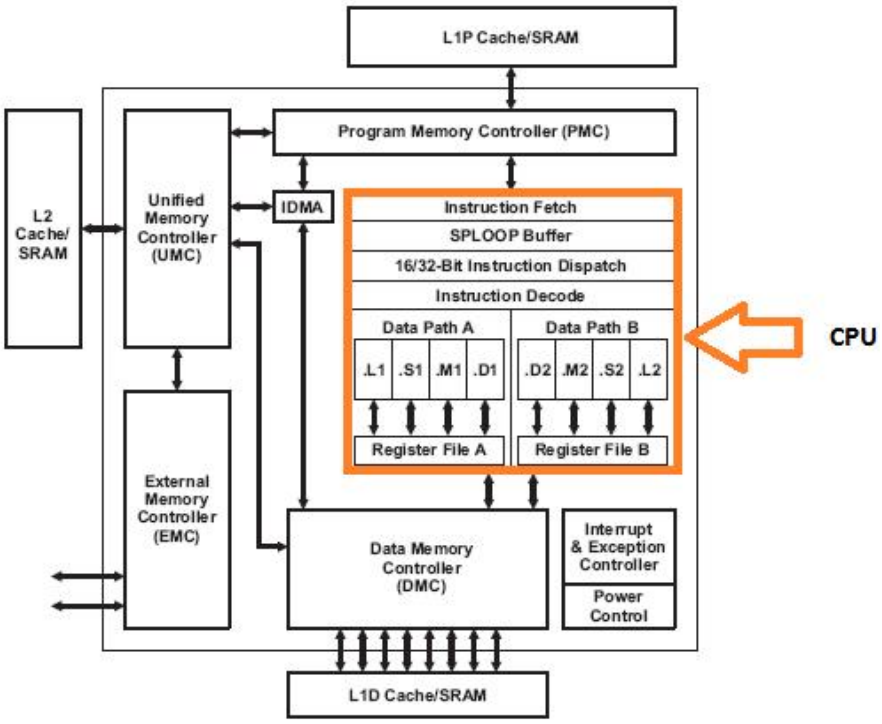


FIGURE 9 : DIAGRAMME BLOC DU CŒUR C66X

2.3. Le 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 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.

La figure 10 montre le schéma fonctionnel d'un cœur DSP TMS320C66x **[4]**:

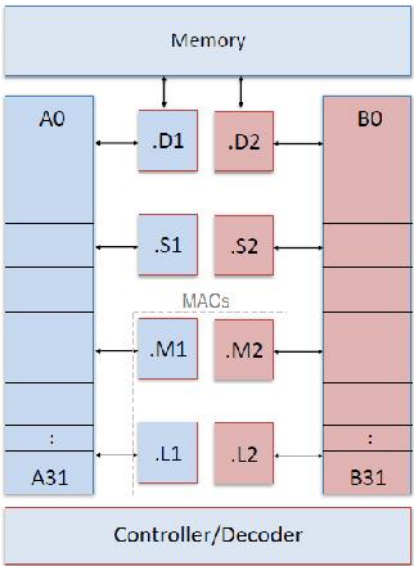


FIGURE 10 : SCHEMA FONCTIONNEL DU CŒUR C66X

Le tableau 3 détaille le rôle de chaque unité fonctionnelle, présentée dans la figure 10.

Unité fonctionnelle	Rôle
.M (multiplier)	Effectue toutes les opérations de multiplication
.L (unité arithmétique et logique)	Effectue en parallèle les opérations +/-
.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 : Unités fonctionnelles du CPU et leurs rôles

2.4. La mémoire interne

Le DSP C6678 intègre une grande capacité de mémoire on-chip. Cette mémoire est organisée en trois niveaux comme illustré sur la figure suivante :

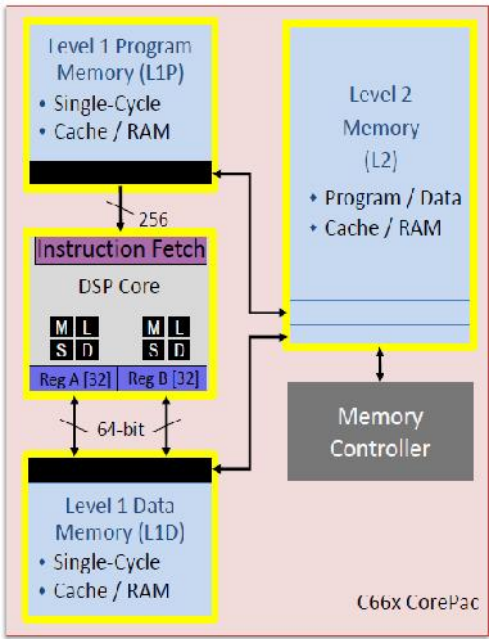


FIGURE 11 : ORGANISATION DE LA MEMOIRE DU DSP C6678

- Le niveau 1 (**L1**) : Chaque cœur possède sa propre mémoire L1. Il est plus proche du CPU et est constitué de 32 KB de mémoire L1P et L1D. Cette mémoire peut être configurée soit en SRAM soit en Cache.
- Le niveau 2 (**L2**) : Cette mémoire est partagée entre les huit cœurs. Sa taille est de 4 MB. Elle est un peu plus loin du CPU. Elle est divisée sur les huit cœurs d'une manière symétrique (C'est-à-dire 512 KB pour chaque cœur).
- Le niveau 3 (**L3**) : Cette mémoire se trouve en dehors du chip. L'accès à cette mémoire se fait à travers le contrôleur de mémoire DDR3 EMIF.

2.5. La carte d'évaluation

La TMDXEVM6678l EVM Board est une carte d'évaluation développée par la société Advantech permettant d'effectuer divers tests et développements sur les DSP de type C6678.

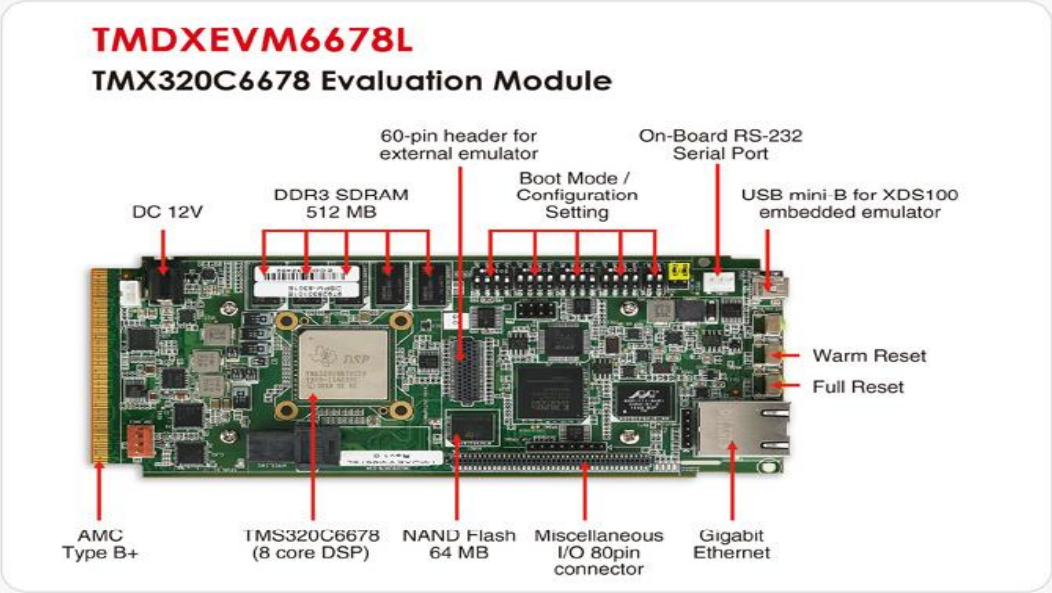


FIGURE 12 : CARTE D'EVALUATION TMDXEVM6678L EVM BOARD

3. Architecture multiboot Linux sur DSP C6678

Une des fonctionnalités les plus importantes d'un radar est le traitement de signaux. Un radar est confronté à d'énormes quantités de données et par conséquent une importante charge en calculs à effectuer. Les radars incorporent de plus en plus de processeurs, atteignant le nombre de quelques centaines, à un tel point qu'ils sont comparables à des machines massivement parallèles.

La présence d'un tel nombre de nœuds de calcul dans une machine induit naturellement le besoin de bien gérer l'ensemble. La mise en place d'un système de supervision permettrait principalement de relever périodiquement des informations sur le fonctionnement de chacun des nœuds de calcul, et ainsi de détecter anomalies, dysfonctionnements et pannes.

Cette partie détaille la réalisation et la mise en place de l'architecture de multiboot Linux que nous avons effectuée sur les huit cœurs du DSP. L'intérêt du multiboot est d'avoir un système d'exploitation Linux qui joue le rôle de superviseur dans la chaîne de traitement des signaux du système radar, tournant sur l'ensemble des cœurs permettant ainsi de tirer profit de la totalité des performances du DSP multicore.

3.1. L'atelier Logiciel

Le poste de développement de l'architecture du multiboot utilise la distribution Linux Ubuntu 10.04. La carte ne disposant pas d'espace de stockage suffisant pour contenir le système de fichiers et une quantité de mémoire limitée, il devient nécessaire d'utiliser un environnement de compilation croisée (ensemble d'outils permettant de compiler du code

source destiné à être exécuté sur une architecture différente de celle sur laquelle a été compilé le code).

Le noyau Linux sera exécuté sur la carte d'évaluation EVM6678, l'accès du noyau à son système de fichiers qui se trouve sur le PC sera assuré par un serveur NFS (Network File System).

Sur le PC, on configure l'adresse IP du noyau Linux chargée sur la carte, le chemin d'accès au système de fichiers, et l'adresse IP auquel il se connectera.

En ce qui concerne le PC, on configure le serveur NFS et l'adresse IP de la carte réseau. L'exécutable du noyau linux configuré sera chargé sur la carte par câble JTAG en utilisant CCS ou chargé via un serveur TFTP (Trivial File Transfer Protocol).

Le déploiement du noyau Linux sur les huit cœurs du DSP a été réalisé en utilisant les outils MAD (Multicore Application Deployment) de Texas Instruments.

La figure 13 donne une vue globale sur la maquette d'essai utilisée pour tester le noyau Linux sur la carte DSP.

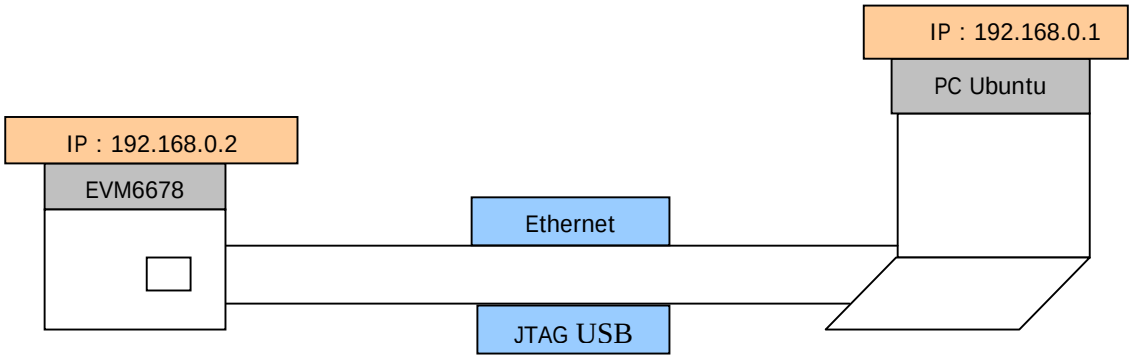


FIGURE 13 : DISPOSITIF DE LANCEMENT DE LINUX SUR EVM6678

La compilation, configuration et chargement du noyau Linux sur le DSP est décrite en détail dans l'**annexe 1**.

3.2. Multicore Application Deployment

Le MAD est un ensemble d'utilitaires fournit par Texas Instruments pour le déploiement d'applications sur des architectures multicœurs **[5]**.

Les principaux utilitaires inclus dans cet outil sont :

- **MAP** (Multicore Application Prelinker) : attribue une adresse virtuelle à chaque segment de l'application ELF (Executable and Linkable Format) en se basant sur le partitionnement mémoire fournit par l'utilisateur puis fait appel au « prelinker » qui permet à son tour de lier chaque segment du fichier ELF à l'adresse virtuelle qui lui a été assignée.

- **MAD Loader** (Multicore Application Deployment) : permet de démarrer une application sur un cœur donné.

- **IBL** (Intermediate Boot Loader) : chargeur d'amorçage (Bootloader) permettant de charger l'image créée par MAD Loader de la mémoire flash ou d'un serveur TFTP vers la mémoire DDR de la cible **[5]**.

3.3. Flux du multiboot du noyau Linux sur le DSP

L'architecture de multiboot Linux que nous avons réalisé pour le DSP C6678 et sa mise en place est illustrée dans la figure suivante :

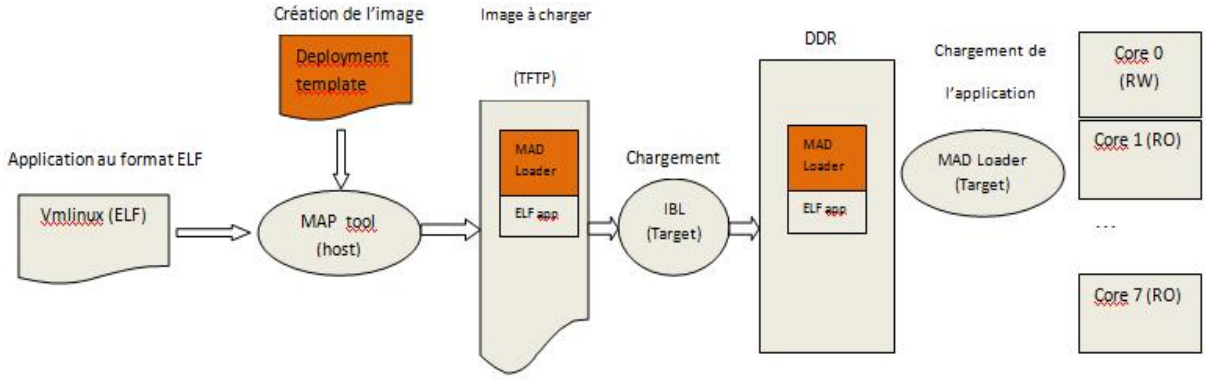


FIGURE 14 : ARCHITECTURE DU MULTIBOOT LINUX SUR LE DSP C6678

Pour qu'on puisse déployer le noyau Linux sur les différents cœurs du DSP, un ensemble d'étapes préliminaires ont été effectués.

3.3.1. Préparation de l'image à déployer

Dans la première étape, une image du noyau Linux que nous avons compilé sous format ELF (un fichier ELF est un format de fichier binaire pour l'enregistrement de code compilé) est passée à l'outil MAP qui lui attribue une adresse virtuelle dans la mémoire DDR du DSP à partir d'une stratégie de déploiement qu'on a définit dans un fichier de format JSON (Java Script Object Notation).

Un fichier JSON est un format de données textuel, générique qui permet de représenter de l'information structurée. Un tel document ne comprend que deux éléments structurels :

- des ensembles de paires nom/valeur;

- des listes ordonnées de valeurs.

Ce fichier informe l'outil MAP sur les partitions mémoires à utiliser dans le DSP, les permissions à attribuer à chaque partition (lecture seule ou lecture/écriture) puis l'application (noyau Linux) à déployer. Le fichier de configuration du déploiement contient les sections suivantes :

- **deviceName** : identifie le périphérique cible, le DSP C6678 dans notre cas.

- **partitions** : cette section identifie les partitions mémoires utilisables sur le DSP et les permissions attribuées à chaque partition.

- **applications** : spécifie l'application qui devra être chargé dans le périphérique, le noyau Linux dans notre cas.

L'**annexe 2** contient un exemple de fichier JSON que nous avons généré pour le DSP C6678.

3.3.2. Déploiement de l'image du noyau Linux

Le fichier de configuration JSON passé en argument à l'outil MAP permet de créer une image contenant l'ensemble des informations nécessaire au déploiement, l'image ainsi créée est placée dans un serveur TFTP pour qu'elle puisse être chargée par la suite sur le DSP **[6]**.

Le chargement de l'image créée par l'outil MAP, du serveur TFTP vers la mémoire DDR, est réalisé par le bootloader IBL, après le chargement, ce dernier passe la main au MAD Loader présent sur le DSP.

MAD Loader initialise les partitions mémoires sur le DSP en affectant à chaque partition les permissions prédéfinies dans le fichier JSON, puis déploie le noyau Linux sur l'ensemble des cœurs.

Le noyau Linux est déployé sur les huit cœurs suivant une architecture maitre-esclave. Le cœur 0 (core 0) est le cœur maitre (master) qui a accès à toutes les ressources du DSP. Les autres cœurs (core 1 à core 7) sont des cœurs esclaves (slaves) et n'ont pas accès à toutes les ressources.

4. Conclusion

La mise en place de plusieurs noyaux Linux sur le DSP permet de superviser des systèmes complexes. Le déploiement du noyau Linux sur l'ensemble des cœurs du C6678 a été réalisé en utilisant les utilitaires MAD de Texas Instruments qui permettent de définir et de mettre en place une architecture de déploiement d'applications sur des architectures multicoeurs.

Le chapitre suivant étudie le protocole de communication PCIe qui relie le PC à la chaîne de prétraitement inclus dans le système radar.

Chapitre 3 : Le protocole PCIe

Ce chapitre présente une vue détaillée sur le protocole PCI Express avec lequel nous allons réaliser la communication entre la carte d'évaluation à base du circuit FPGA Cyclone IV et l'ordinateur.

1. Introduction

Le protocole PCI Express, abrégé PCI-E, PCI-XP ou PCIe ou encore *3GIO (3rd Generation Input/Output)* est un standard de communication développé et introduit par *Intel Corporation* en 2002. Il spécifie un bus local (« bus PCI express ») et un connecteur qui sert à connecter des cartes d'extension sur la carte mère du système l'adoptant (un ordinateur par exemple).

C'est la troisième génération des bus I/O à hautes performances utilisée pour l'interconnexion des périphériques dans des applications telles que les plateformes informatiques ou de communication. La première génération de ces bus inclue les bus ISA, EISA, VESA et les bus « *Micro Channel* Architecture » (MCA), tandis que la seconde génération inclue les bus PCI (*Peripheral Component Interconnect*), AGP et PCI-X. PCIe est l'héritant direct des bus PCI et PCI-X **[15]**.

1.1. Bus PCI et PCI-X

Le bus PCI est un bus de 32 bits ou 64 bits à haute performance qui permet d'interconnecter les périphériques et le processeur/mémoire d'un système. Il a été développé par *Intel Corporation* en 1993 qui devait théoriquement permettre le transfert de données avec un débit quatre fois plus grand que les débits obtenus par bus disponibles ces temps-là. Cette architecture a été développée pour résoudre le problème des « *data bottlenecks* » (étranglement de données qui résulte entre un système qui nécessite un débit de transfert de données nettement supérieur à la bande passante offerte par le bus utilisé) causé par les nouveaux systèmes d'exploitation à interface graphique comme le Microsoft Windows. En plus, la bande passante du PCI était idéal pour les nouveaux processeurs pentium qu'Intel se préparait à lancer dans le marché. PCI avait pour objectif d'exploiter toutes les capacités de calcul de ces processeurs.

La première révision du PCI a été cadencée à 33 MHz, puis, en 1995, la révision 2.1 a été cadencée à 66MHz.

La figure suivante représente l'architecture générale d'un système basé sur le bus PCI. Ce system consiste en un pont entre le bus du processeur (*FSB : Front Side Bus*) et le bus PCI (le « North Bridge »). A ce pont sont associés le bus de mémoire (SDRAM), le bus graphique AGP et le bus PCI. Les périphériques IOs partagent le bus PCI et y sont liés en mode multipoint. Cette liaison est faite directement ou via les slots d'extensions. Le « South Bridge » fait la fonction de switching entre les périphériques fonctionnant a une fréquence plus basse comme le bus ISA et USB.

Le fonctionnement d'un tel système est basé sur l'échange des paquets de données entre un maître et un esclave sous le contrôle d'un arbitre (situé dans le North Bridge dans ce cas).

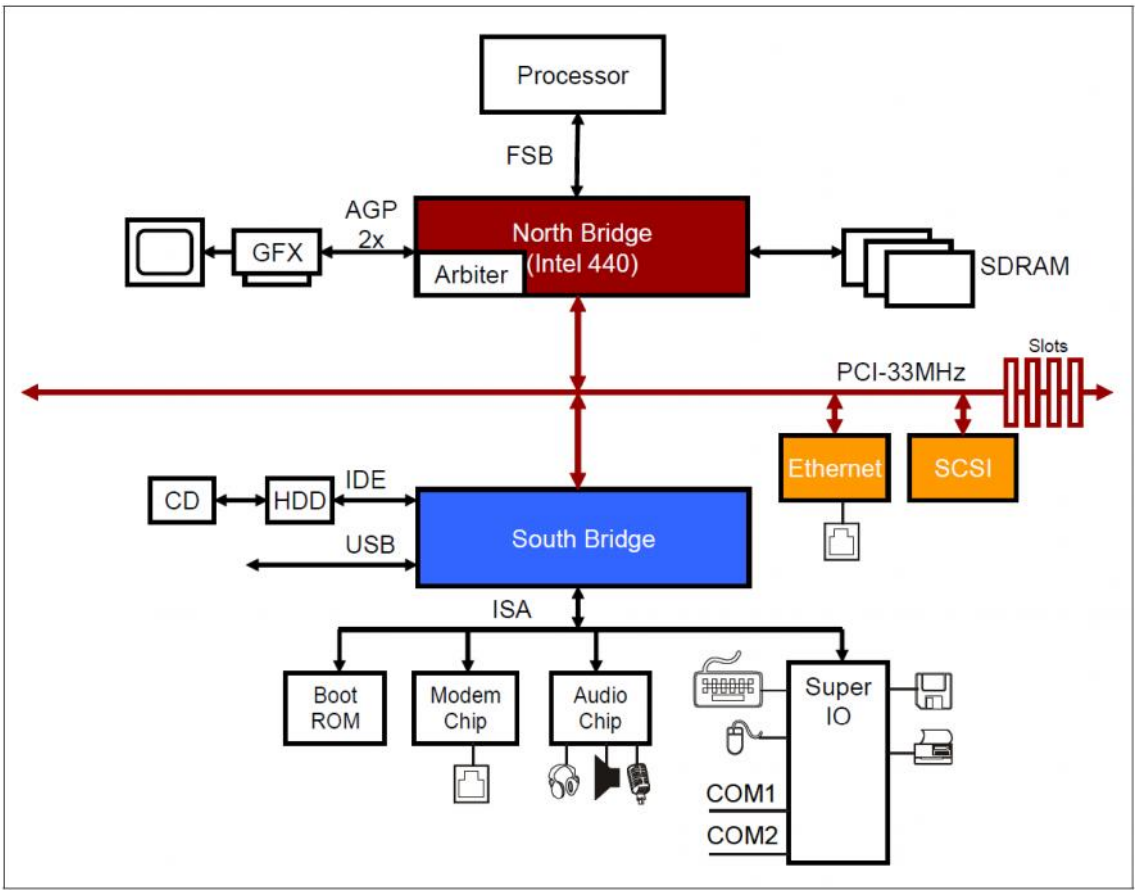


FIGURE 15: ARCHITECTURE GENERALE D'UN SYSTEME BASE SUR LE BUS PCI

1.2. Limitations du bus PCI

Malgré sa flexibilité et son adaptation avec l'évolution technologique, le bus PCI a montré des faiblesses qui ont ouvert la voie pour la définition d'une nouvelle architecture (PCI Express). Ces faiblesses sont la limitation en bande passante, l'encombrement en nombre de broches, l'incapacité à supporter des transferts en temps réel et d'adresser les futures périphériques IO.

Pour ces raisons PCI-SIG à introduit le PCI-Express. Le protocole PCI Express n'est pas une amélioration du PCI, comme l'a été le PCI-X, mais une évolution qui a changé carrément l'architecture matérielle mais qui a gardé une grande compatibilité avec l'architecture logicielle du PCI, ce qui a garanti la rétrocompatibilité avec les périphériques PCI.

2. Architecture du protocole PCI Express

Le PCI Express a introduit une très grande évolution dans les bus locaux par sa bande passante, son adaptation avec les nouveaux périphériques d'entrées/sorties, le support des applications temps-réel (isochrones), le support du multi-host, l'augmentation du nombre des périphériques qui peuvent se connecter au même système. Cette évolution revient entre autres à son mode de communication et à son architecture hardware.

2.1. Mode de communication

Contrairement au PCI qui est un bus parallèle, multipoint sur lequel les transferts se font par cycle de bus (phase d'adressage puis phases de données), la communication par PCI Express se caractérise par **[7]** :

- Une **interface série** basée sur deux liaisons simplex (dual-simplex), une pour la transmission et l'autre pour la réception. Chaque liaison PCI Express implémente une ou plusieurs voies pour la réception et la transmission, une liaison avec une seule voie est par convention notée **x1**, pour deux voies **x2**, pour quatre voies **x4** et ainsi de suite jusqu'à **x32**. Une voie est constituée de deux lignes, une pour la réception et l'autre pour la transmission, (dans la suite, la ligne de transmission sera notée TX et celle de réception RX).

- **Liaison « *point-to-point* »** : un périphérique PCI Express **A** doit être directement lié à un autre périphérique PCI Express **B** à travers ses voies de réception et de transmission (RX_**A**⇔TX_**B**, TX_**A**⇔RX_**B**), pour connecter plusieurs périphériques, on utilise des switches PCI Express.

- **Protocole de Communication par paquets** : PCI Express encode les transactions en des paquets qui contiennent les informations nécessaires pour l'acheminement du paquet; l'identité du périphérique cible, l'identité du périphérique initiateur, d'autres informations sur le type de la transaction, la taille,… et si elle existe, la donnée à transférer **[7]**.

La figure suivante montre une liaison PCIe entre deux endpoints.

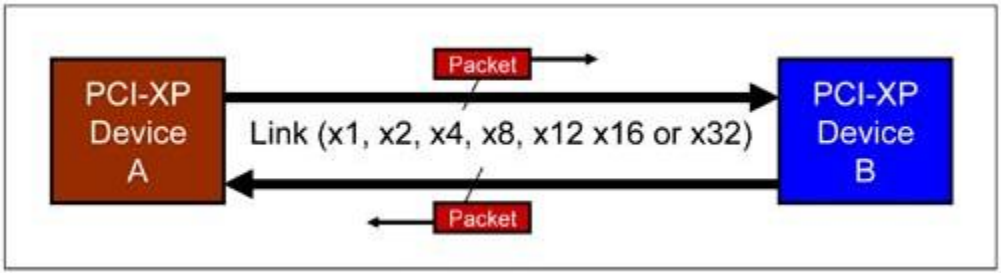


FIGURE 16 : LIAISON PCI EXPRESS

2.2. La bande passante de la liaison PCI Express :

La bande passante d'une liaison PCI Express entre deux périphériques dépend de la génération du PCI Express lui-même (appelée aussi «spécification ») et du nombre de voies implémentées entre les deux périphériques.

Chaque spécification se caractérise par une fréquence par ligne (la même fréquence pour la ligne de transmission et de réception), la fréquence de la voie est le double de la fréquence de la ligne sachant que la transmission et la réception peuvent se faire simultanément.

La spécification actuelle est la spécification 3.0 qui fonctionne à une fréquence de 8Gbits/sec, la spécification 2.0 fonctionne à 5Gbits/sec et la spécification 1.0 à 2.5 Gbits/sec.

La bande passante de la liaison est le produit des bandes passantes des voies qui la constituent **[7]**. Par exemple, pour une liaison PCI Express x1 entre deux périphériques qui implémentent la spécification 1.0 (2.5Gbits/sec); on aura la bande passante suivante :

Bande passante en GBytes/sec = 2.5 * 1 * 2 * (8/10) /8 [GBytes/s]

- La multiplication par 1 pour le nombre de vois : x1.
- La multiplication par 2 pour les deux directions (transmission/réception).
- La multiplication par (8/10) le codage 8b/10.
- La division par 8 pour le calcul de la bande passante en GBytes.

Ce qui va nous donner 0.500 GBytes/s (500 MBytes/s).

Le même calcul mais avec la spécification 2.0 (5 Gbits/s au lieu de 2.5 Gbits/s) nous donnera 1 GBytes/s.

Le tableau suivant représente la bande passante du PCI Express des liaisons de différentes largeurs pour la spécification 1.

PCI Express Link Width	x1	x2	x4	x8	x12	x16	x32
Aggregate Band-width (GBytes/sec)	0.5	1	2	4	6	8	16

Tableau 4: Bande passante des liaisons de différentes largeurs

A partir de ces deux exemples nous pouvons constater que la bande passante permise par PCI Express est nettement supérieure à celle du PCI et PCI-X. Le graphe sur la figure suivante représente une comparaison entre les performances par rapport au nombre totale des broches pour les bus PCI, PC-X et PCI Express.

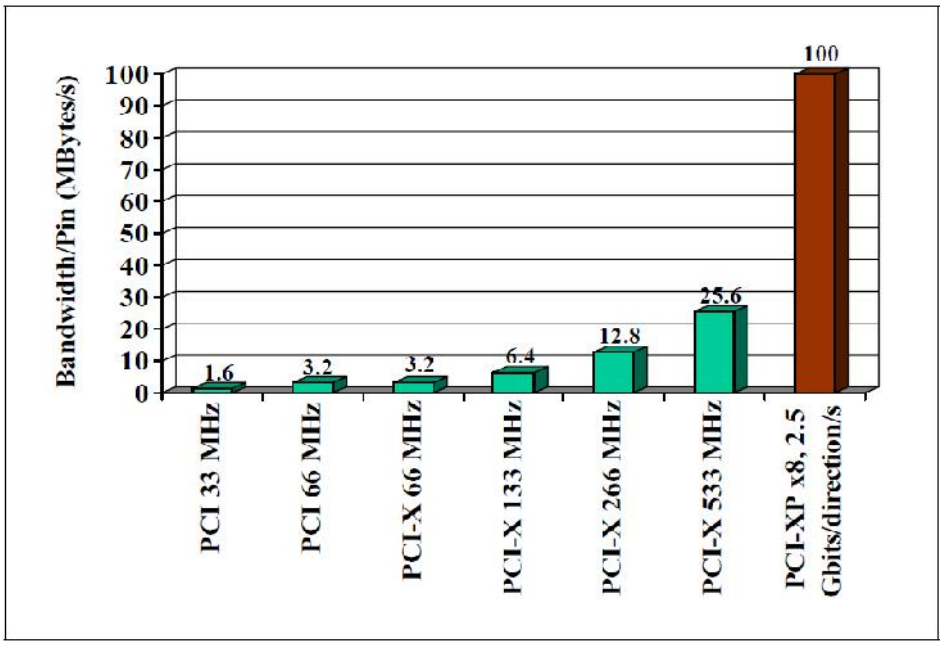


FIGURE 17: COMPARAISON DES PERFORMANCE PAR BROCHES DES BUS PCI, PCI-X ET PCI EXPRESS

2.3. Les transactions PCI Express

Les transactions sont la base de transfert d'informations entre les périphériques PCI Express. PCI Express utilise ce qu'on appelle un « split-transaction protocol ». Cela signifie qu'il y a deux phases de transaction, la phase de demande (request en anglais) et la phase de réponse (completion en anglais). L'initiateur de la transaction appelé « requester » émet le paquet de demande qui prend son chemin vers la cible de la demande appelé « completer ».

La phase de réponse n'est pas lancée obligatoirement directement après celle de la demande, le « completer » enregistre la demande avec les coordonnées du demandeur et décide quand il va y répondre. Ceci permet d'éviter les temps d'attente de la réponse et les temps de reprises des demandes qui n'ont eu aucune réponse, ce qui permet d'améliorer la bande passante.

2.3.1. Les types de transactions PCI Express

La spécification du PCI Express définie trois types de transactions : les transactions mémoire, les transactions IO et les transactions de configuration.

Les transactions mémoire sont des transactions ciblant des espaces mémoire, elles transfèrent les données de ou vers une zone mémoire. Il y a trois types de transactions mémoire : demande de lecture mémoire « Memory Read Request », réponse à une demande de lecture mémoire « Memory Read Completion » et demande d'écriture mémoire « Memory Write Request » **[7]**. Les transactions mémoire utilisent l'un des deux formats d'adresses : 32 bits ou 64 bits.

Les transactions IO sont des transactions ciblant un espace d'entrées/sorties. Il existe quatre types de transactions IO : demande de lecture IO « IO Read Request », réponse à une demande de lecture IO « IO Read Completion », demande d'écriture IO « Memory Write

Request » et réponse à une demande d'écriture IO « IO Write Completion ». Les transactions IO utilisent des adresses 32 bits.

Les transactions de configuration ciblent l'espace de configuration d'un périphérique utilisées pour la configuration. Il existe quatre types de transactions de configuration : demande de lecture de configuration « Configuration Read Completion », réponse à une demande de lecture de configuration « Configuration Read Completion », demande d'écriture de Configuration « Configuration Write Request » et réponse à une demande d'écriture de Configuration « Configuration Write Completion ».

2.4. Les couches PCI Express

Le protocole PCI Express est basé sur une architecture en couches (*Layered Architecture*) (comme le modèle OSI et le modèle TCP/IP).

La spécification définit 3 couches : la couche de transaction (*Transaction Layer*), la couche de liaison de données (*Data Link Layer*) et la couche physique (*Physical Layer*).

Chacune de ces couches effectue des fonctions et des traitements bien déterminées. Tout périphérique PCI Express implémente ces trois couches pour les deux parties, la réception RX et la transmission TX comme le montre la figure 18.

Le noyau du périphérique ou la couche d'application (*device core)* comporte l'implémentation des fonctions du périphérique (par exemple pour une carte graphique PCI Express, cette couche contient l'ensemble des fonctionnalités d'une carte graphique), c'est lui qui communique les données et les informations nécessaires pour la formation des paquets à la couche de transaction via l'interface « PCI Express Core Logic Interface ».

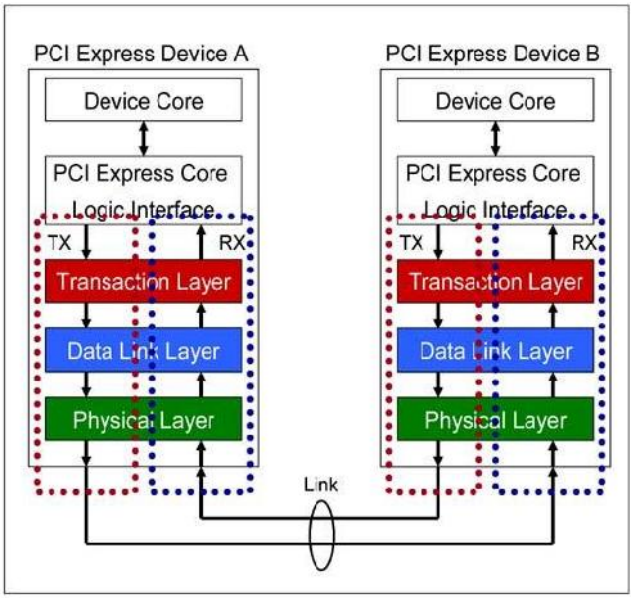


FIGURE 18: COUCHES D'UN PERIPHERIQUE PCI EXPRESS

Pour chaque couche, il y a un type de paquet qui lui est associé **[7]**:

- Les **TLPs** (*Transaction Layer Packets*) : ils sont générés par la partie TX de la couche de transaction du l'expéditeur et reçues par la partie RX de la couche de transaction du destinataire.

- Les **DLLPs** (*Data Link Layer Packets*) : ils sont générés par la partie TX de la couche de liaison de données du l'expéditeur et reçues parla partie RX de la couche de liaison de données du destinataire.

- Les **PLPs** (*Physical Layer Packets*) : ils sont générés par la partie TX de la couche physique du l'expéditeur et reçues par la partie RX de la couche physique du destinataire.

2.4.1. La couche de transaction

La couche de transaction se situe entre la couche d'application (*device core*) et la couche de liaison de données. C'est cette couche qui forme les TLPs (*request packets*) à l'issue de la demande de la couche d'application pour les passer à la couche de liaison de données. Côté réception, les paquets reçus par cette couche, sont passés à la couche d'application après le traitement nécessaire.

2.4.1.1. Les paquets TLPs

Les TLPs (Transaction Layer Packets) sont les paquets de transferts d'informations entre les périphériques PCI Express, ils sont initiés par la couche de transaction de la source (Device A dans la figure 19) et reçus par la même couche de la destination (Device B dans la figure 19) :

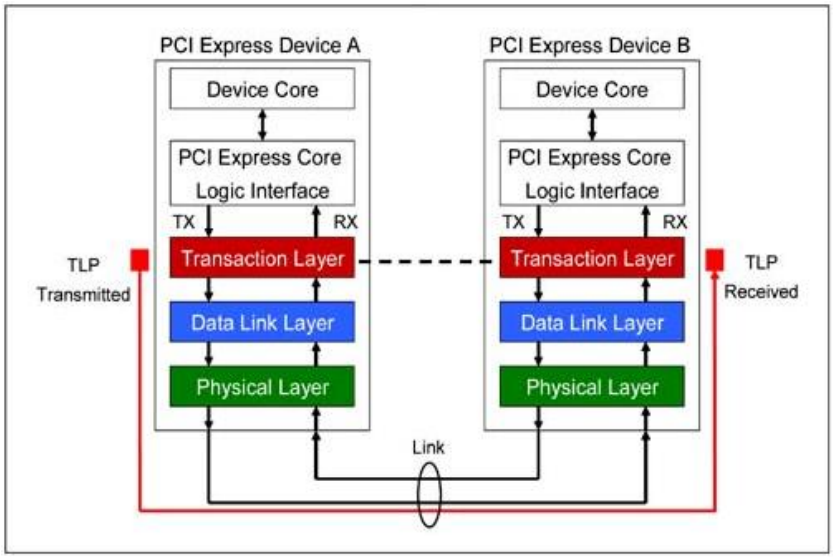


FIGURE 19 : SOURCE ET DESTINATION DES TLPS

Un TLP est formé d'un entête (Header) suivi de la donnée à transférer (Data Payload) puis un code d'erreur (TLP Digest) calculé sur l'entête et la donnée, tous ces champs sont alignés sur 4 Bytes (figure 20).

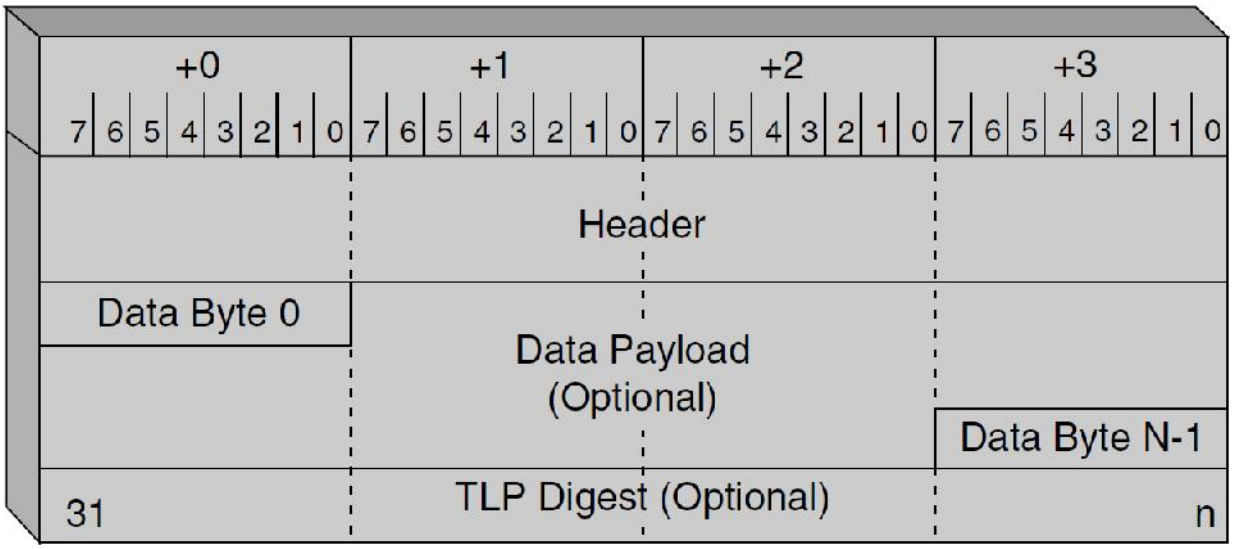


FIGURE 20: FORME GENERALE D'UN TLP

2.4.2. La couche de liaison de données

Cette couche se situe entre la couche de transaction et la couche physique, sa principale fonction est la détection et la correction d'erreur. Elle s'assure que la donnée émise dans le bus a été reçue correctement et dans l'ordre. En plus du traitement des TLPs issus de la couche de transaction ou reçus via la couche physique, cette couche est la source des paquets DLLPs qu'elle utilise pour le contrôle de la liaison et l'acquittement des TLPs.

2.4.2.1. Le processus de traitement des TLPs

La couche physique prend le TLP initial issu de la couche de transaction et lui ajoute un numéro de séquence qui sera utilisé lors de la réception par cette couche pour examiner l'ordre dans lequel sont reçus les TLPs et aussi pour l'acquittement. Pour garantir l'intégrité de données, un code CRC (« LCRC » pour Link-CRC) est calculé sur le TLP initial et le numéro de séquence.

Lors de la réception cette couche prend les TLPs reçus via la couche physique. Elle vérifie le code « LCRC » puis vérifie le numéro de séquence pour passer le TLP à la couche de transaction comme le montre la figure suivante.

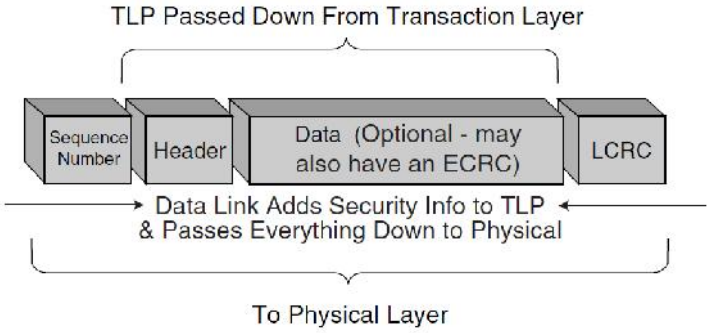


FIGURE 21 : PROCESSUS DE FORMATION DES TLPS

2.4.2.2. Les paquets de la couche de liaison de données (DLLPs)

Les paquets DLLPs reçus par la couche de liaison de données sont utilisés pour réaliser des fonctions de contrôle et de maintenance de la liaison PCIe. La spécification définie les types de DLLP suivant :

- Les **DLLPs d'acquittement** utilisés pour l'acquittement des TLPs échangés par les périphériques.
- Les **DLLPs du « Flow Control »** : le « Flow Control » est un protocole introduit par la spécification pour la bonne gestion des ressources. Cela consiste à ce que tout périphérique s'assure de la disponibilité des ressources (l'espace mémoire) chez le périphérique cible avant d'effectuer la transaction.

2.4.3. La couche physique

C'est la couche la plus basse d'un périphérique PCI Express, elle est connectée à la liaison PCIe d'un côté et à la couche de liaison de données de l'autre côté. Elle a trois principales fonctions :

- L'initialisation et la maintenance de la liaison.
- La préparation des paquets issus de la couche de liaison de données (TLPs et DLLPs) pour les mettre sur la ligne.
- Le traitement des données entrantes de la liaison et la reformation des paquets (TLPs et DLLPs) avant de les passer à la couche de liaison de données.

Elle contient deux sous-blocs ; le bloc logique et le bloc électrique. Chacun de ces deux blocs se compose d'une unité de transmission et d'une unité de réception, les deux unités peuvent fonctionner indépendamment l'une de l'autre (Figure 22).

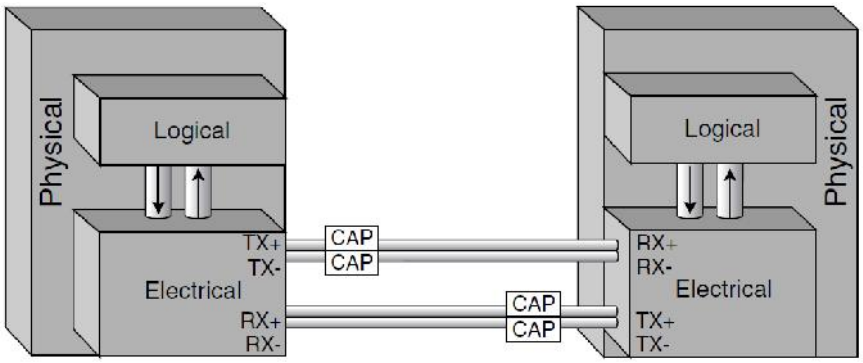


FIGURE 22 : ARCHITECTURE PHYSIQUE ENTRE DEUX PERIPHERIQUES PCI EXPRESS

3. Espace de configuration PCI Express

Chaque périphérique PCI Express implémente 4 Ko d'espace de configuration. Les premiers 256 octets sont compatibles avec l'espace de configuration PCI et PCI-X.

Les 64 premiers octets de l'espace de configuration d'une fonction sont appelés « l'entête de l'espace de configuration ». Il existe deux types d'entête de configuration, l'entête de type 0 pour les périphériques simple (End-point devices) et l'entête de type 1 pour les switches.

L'entête de type 0 représenté dans la figure 24 contient les champs suivant :

- « Vendor ID » : l'identifiant du fabriquant du périphérique, chaque fabriquant possède un identifiant propre à lui.
- « device ID » : l'identifiant du périphérique, chaque périphérique possède un identifiant qui le différencie des autres périphérique du même fabriquant.
- « Status » : registre de statut d'un ensemble de fonctionnalités de la fonction du périphérique telle que les interruptions.
- « Command » : registre d'activation d'un ensemble de fonctionnalités de la fonction du périphérique telle que les interruptions.
- « revision ID » : représente l'identifiant de la révision de la fonction au cas où le fabriquant a produit plusieurs version du périphérique.
- « Header Type » : identifie le type de l'entête (figure 23).

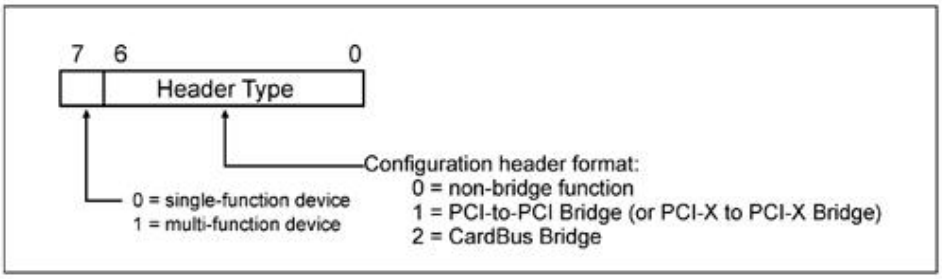


FIGURE 23 : CHAMP "HEADER TYPE" DE L'ENTETE

- « Base Address Registers » (BARs) : ces registres sont utilisés pour contenir les adresses de base des espaces d'adressage contenu dans la fonction. Par exemple, si la fonction implémente une mémoire alors le système va utiliser un BAR pour adresser cette mémoire.

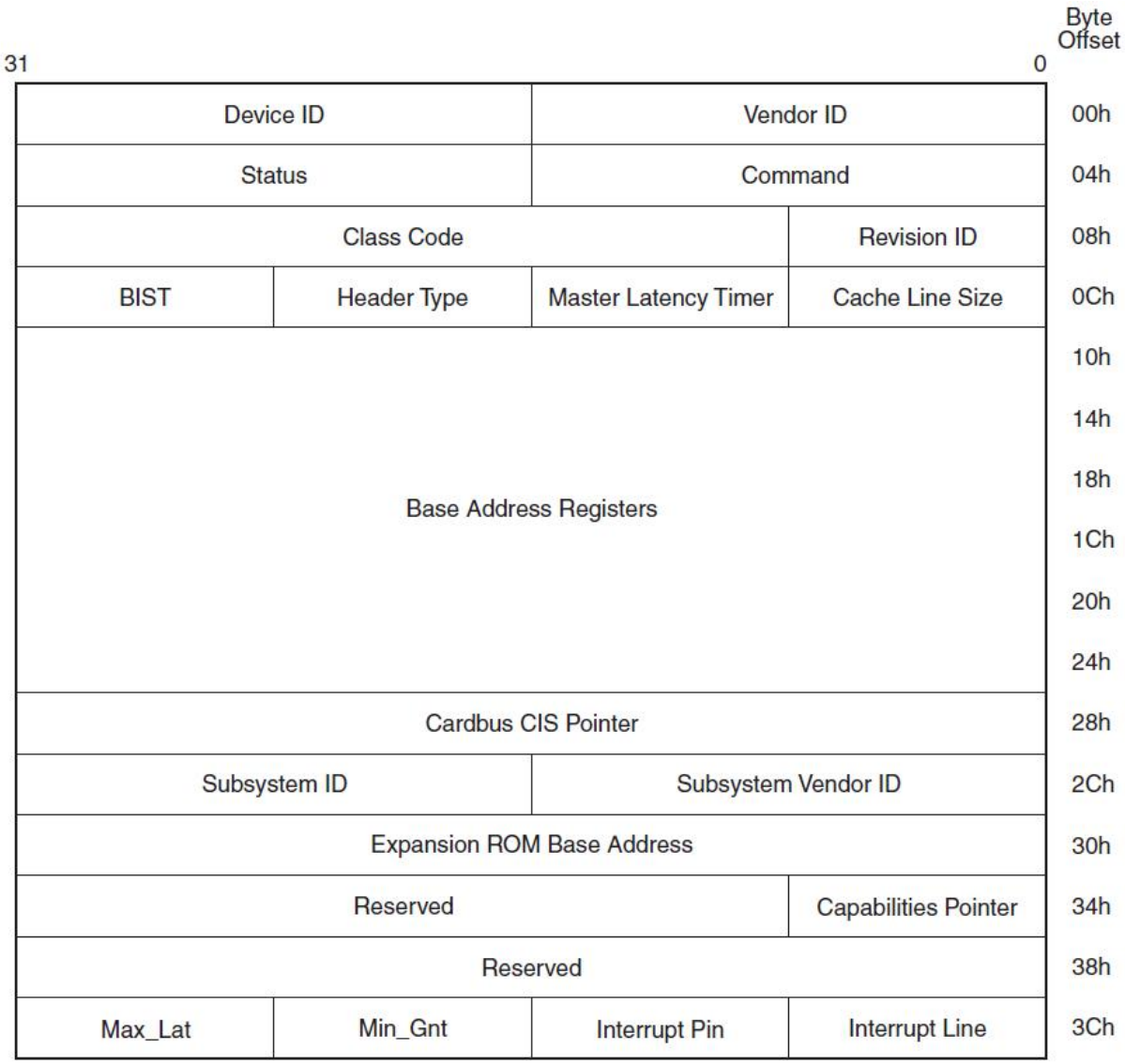


FIGURE 24: ENTETE DE L'ESPACE DE CONFIGURATION DE TYPE 0

L'entête de l'espace de configuration de type 1 (Figure 25) contient presque les mêmes registres que celui de type 0, la différence principale est que les switches ont des bus subordonné et qu'ils doivent dispatcher les paquets via ces bus pour les faire atteindre leurs cibles, pour cela cet entête contient les champs « secondary Bus Number » et « primary Bus Number ».

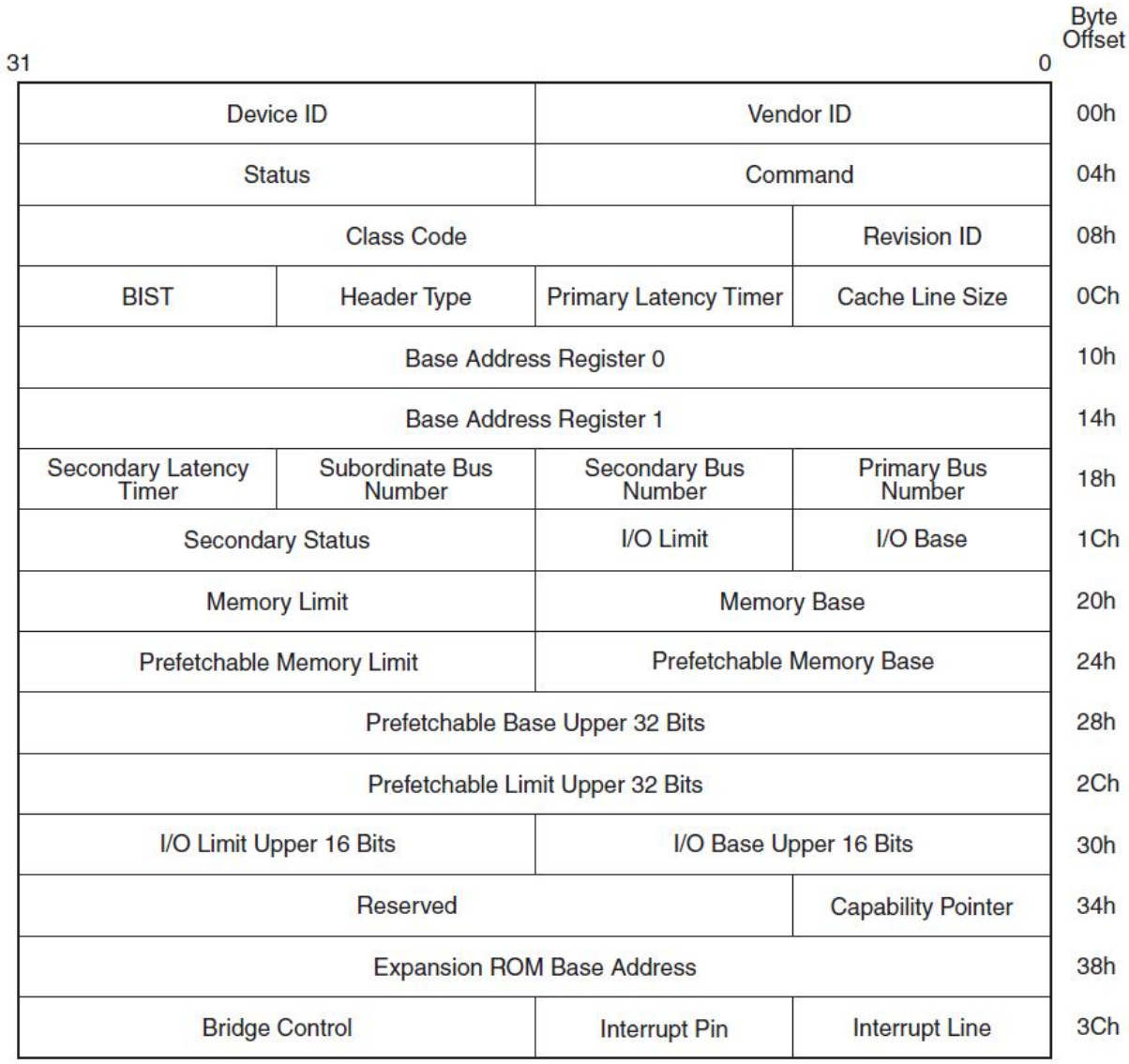


FIGURE 25 : ENTETE DE CONFIGURATION DE TYPE 1

4. Conclusion

Le PCIe est un protocole suffisamment souple pour couvrir la majorité des plateformes informatiques. Cette flexibilité est obtenue grâce à l'architecture en couches du PCIe et de la compatibilité avec les bus d'ancienne génération.

Le chapitre suivant décrit la couche application que nous avons développée pour tester les performances de la communication entre le circuit FPGA et le PC via le protocole PCIe.

Chapitre 4 : Développement du pilote PCIe et des scénarios d'analyse des performances

Pour connecter la carte de développement du FPGA au PC et tester les performances de la communication entre les deux architectures, le développement d'un pilote de périphérique ainsi qu'un ensemble de scenarios de tests est nécessaire.

L'objectif de ce chapitre est de décrire les étapes de développement du driver PCIe sous Linux et avec l'outil Windriver ainsi que les scenarios de tests développées.

1. Architecture matérielle

1.1. Carte de développement

Pour la réalisation et le test de la liaison PCI Express entre le FPGA et le PC, nous avons utilisé la carte « Cyclone IV GX FPGA Development Kit » d'Altera.

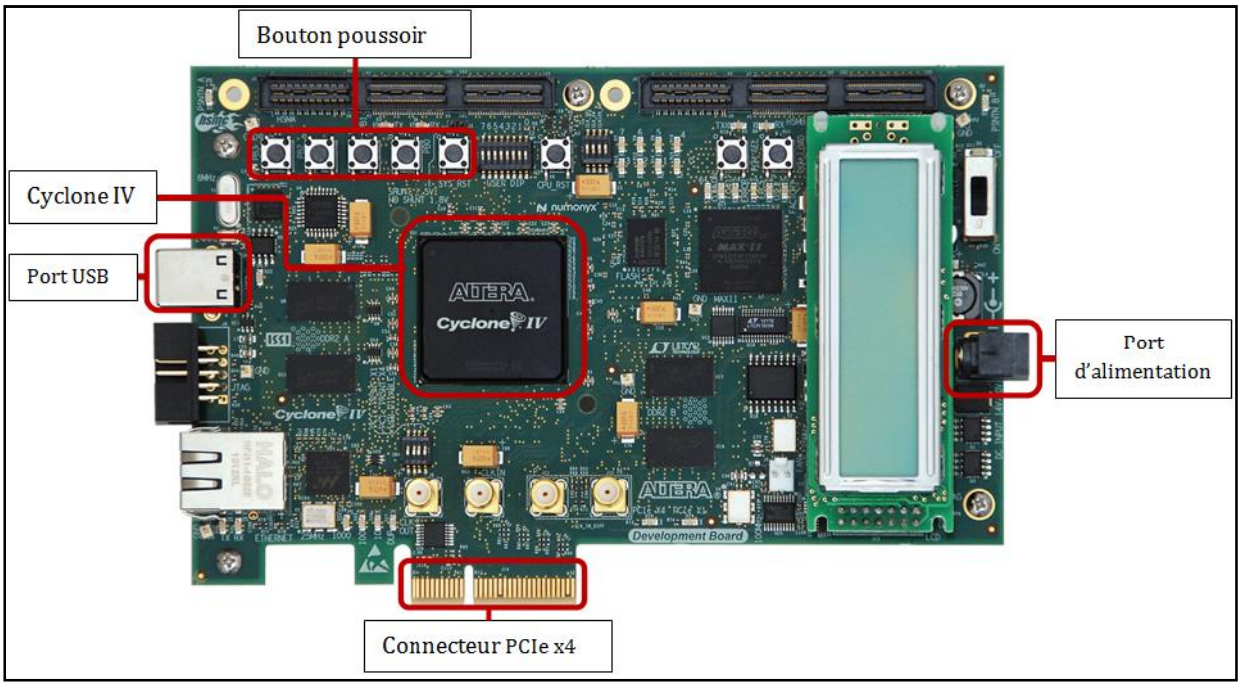


FIGURE 26 : CARTE DE DEVELOPPEMENT CYCLONE IV

Altera propose trois familles de FPGAs :

- La famille des FPGAs à hautes performances nommés STRATIX.
- La famille *« mid range »* appelée « ARRIA ».
- La famille des FPGA grande série appelés « Cyclone ».

Cette carte est basée sur le FPGA Cyclone IV GX EP4C150DF31C7. Le tableau suivant représente les caractéristiques des FPGAs Cyclone GX.

Resources	EP4CGX15	EP4CGX22	EP4CGX30	EP4CGX30	EP4CGX50	EP4CGX75	EP4CGX110	EP4CGX150
Logic elements (LEs)	14,400	21,280	29,440	29,440	49,888	73,920	109,424	149,760
Embedded memory (Kbits)	540	756	1,080	1,080	2,502	4,158	5,490	6,480
Embedded 18 × 18 multipliers	0	40	80	80	140	198	280	360
General purpose PLLs	1	2	2	4	4	4	4	4
Multipurpose PLLs	2	2	2	2	4	4	4	4
Global clock networks	20	20	20	30	30	30	30	30
High-speed transceivers	2	4	4	4	8	8	8	8
Transceiver maximum data rate (Gbps)	2.5	2.5	2.5	3.125	3.125	3.125	3.125	3.125
PCIe (PIPE) hard IP blocks	1	1	1	1	1	1	1	1
User I/O banks	9	9	9	11	11	11	11	11
Maximum user I/O	72	150	150	290	310	310	475	475

Tableau 5 : Caratéristiques des FPGAs Cyclone d'Altera

Cette carte permet d'implémenter l'IP de la génération 1.0 du protocole PCI Express avec un nombre de lignes allant jusqu'à 4 lignes (x1, x2, x4).

1.2. Blocs du système

La figure suivante représente le schéma général du système, il se compose de trois blocs :

- L'IP PCI Express.
- La mémoire « On-Chip Memory ».
- Le « Direct Memory Access » : DMA qui va effectuer les transactions de la mémoire du FPGA vers le PC et vice-versa.

Le DMA est un procédé informatique où des données, circulant de ou vers un périphérique (port de communication, disque dur), sont transférées directement par un contrôleur adapté vers la mémoire principale de la machine, sans intervention du microprocesseur si ce n'est pour lancer et conclure le transfert. La conclusion du transfert ou la disponibilité du périphérique peuvent être signalés par interruption.

L'interconnexion entre ces blocs est réalisée par le bus Avalon qui permet d'implémenter plusieurs interfaces **[13]**.

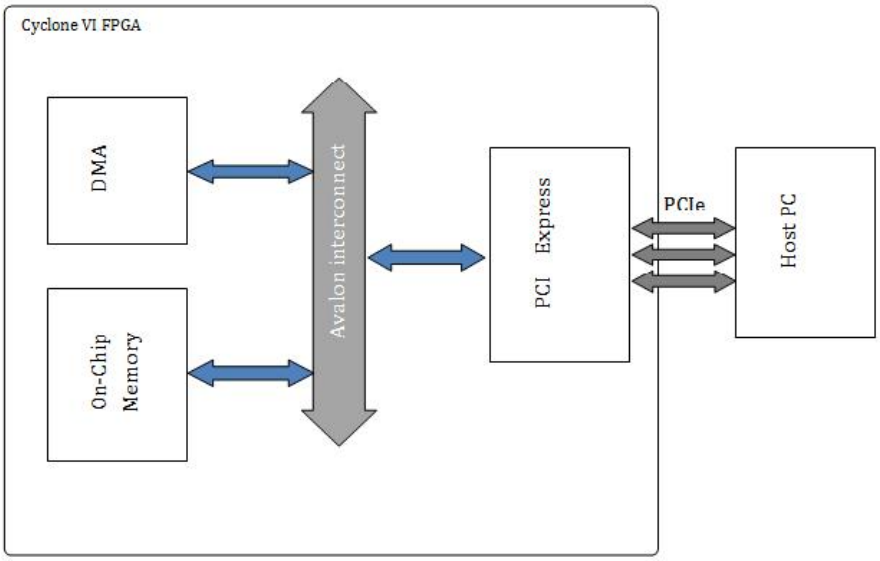


FIGURE 27 : SCHEMA DE L'ARCHITECTURE MATERIELLE

Cette conception réalisée dans la partie hardware du projet permettra au système de réaliser les opérations suivantes :

- Accès à l'espace de configuration de l'IP PCIe.
- Accès aux registres de contrôle de l'IP PCIe.
- Accès mémoire direct du root-complex.
- Accès mémoire DMA : de la mémoire du PC vers la mémoire du FPGA et vice-versa.

2. Développement du pilote PCIE

La carte de développement FPGA est connectée directement au bus PCIe sur la carte mère du PC. Pour que le système d'exploitation puisse reconnaitre ce périphérique et ainsi permettre aux applications externes (développées par l'utilisateur) de l'utiliser, un pilote PCIe doit être implémenté.

Deux solutions se sont offertes à nous, la première solution consiste à développer manuellement le pilote PCIe en utilisant l'API (Application Programming Interface) fournit dans les sources du noyau Linux. La deuxième solution est d'utiliser le logiciel propriétaire Windriver.

Dans la suite de ce chapitre seront développées les deux solutions proposées, les avantages et les inconvénients de chaque solution puis la solution retenue pour le développement des scénarios de test pour la mesure des débits de communication PCIe entre le FPGA et le PC.

2.1. Pilote PCIe sous Linux

2.1.1. Introduction aux pilotes de périphériques sous Linux

Un pilote de périphérique (device driver) permet à un programme utilisateur d'accéder à des ressources externes. Ces ressources peuvent être de type :

- matérielles: carte d'extension, périphérique sur la carte mère
- logicielles: mémoire physique

Il existe différents types de pilotes suivant le périphérique à contrôler **[8]**:

- les pilotes en **mode caractère** (char drivers): ce type de pilote est le plus simple à mettre en œuvre. Il permet de dialoguer avec le périphérique en échangeant un nombre variable d'informations binaires. Les ports séries et parallèles sont des exemples de périphériques contrôlés en mode caractère.
- les pilotes en **mode bloc** (block drivers): ces pilotes échangent des informations avec les périphériques uniquement par blocs de données. Un exemple d'un tel périphérique est le disque dur.
- les pilotes **réseau** (network drivers): ces pilotes sont destinés à contrôler des ressources réseau.

La figure suivante donne un aperçu sur le flux d'exécution d'un driver :

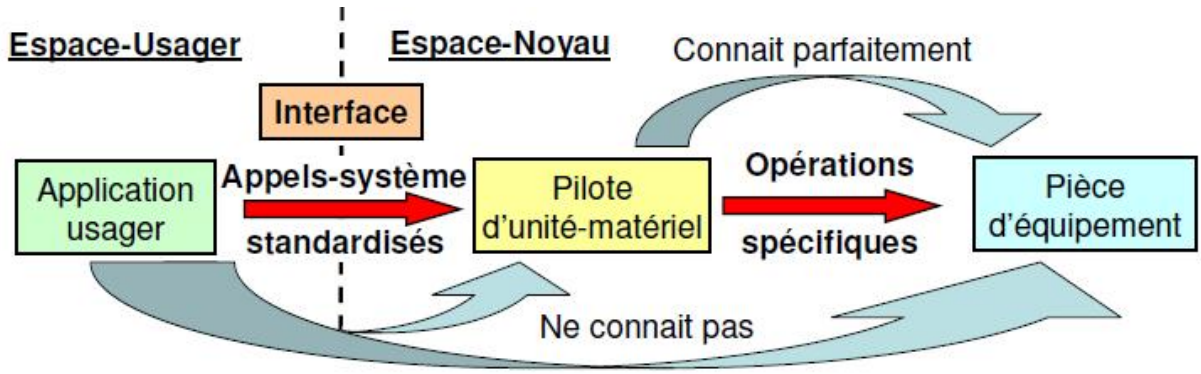


FIGURE 28 : FLUX D'EXECUTION D'UN PILOTE DE PERIPHERIQUE

Vu de l'extérieur du Noyau, les pilotes sont des "boites noires" qui permettent d'utiliser correctement des pièces d'équipement, au travers d'une interface de programmation bien définie.

Pour une application-usager, l'interface de programmation fait voir le matériel comme si c'était un fichier (ouvrir, lire, écrire, fermer, …).

La figure suivante donne une vue détaillée du noyau Linux **[11]** :

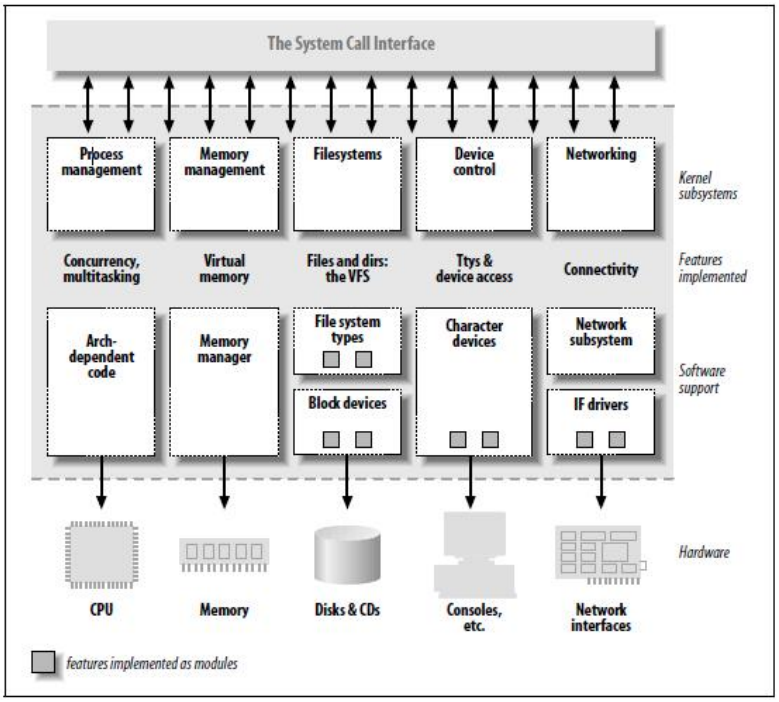


FIGURE 29 : VUE DETAILLEE DU NOYAU LINUX

Dans le cas de Linux, les pilotes de périphériques sont ajoutés au noyau de deux manières **[10]** :

- **Statique** : dans ce cas la, le code source du pilote est ajouté à l'arborescence des sources du noyau Linux. Le pilote sera chargé même en l'absence du périphérique à contrôler. Ce type de pilote était utilisé systématiquement jusqu'à la version 1.2 du noyau Linux.

- **Dynamique** : dans ce cas la, on peut ajouter dynamiquement à un noyau Linux existant un pilote dont le code source ne sera pas contenu dans l'arborescence du noyau. Le pilote constitue alors un module chargeable.

L'utilisation des modules chargeables permet aussi de limiter la mémoire utilisée par les pilotes car ceux-ci sont chargés uniquement lorsqu'un programme utilisateur ou un autre module en fait la demande.

2.1.2. Structure du pilote PCIe sous Linux

Les pilotes PCIe suivent les règles de développement des pilotes caractères mais l'API du noyau Linux inclut de nombreuses fonctions dédiées facilitant fortement la gestion de ce type de périphérique. Les prototypes de ces fonctions sont définis dans le fichier <linux/pci.h>.

Le flux d'exécution du pilote développé pour le périphérique PCI express est montré dans la figure suivante :

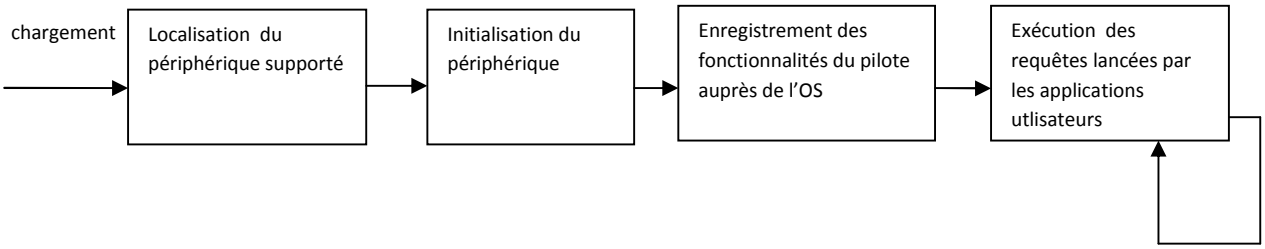


FIGURE 30 : FLUX D'EXECUTION DU DRIVER PCIE

Un pilote de périphérique a besoin d'une fonction d'entrée spécifique qui est exécutée lorsque le pilote est chargé. Cette fonction est alors responsable de faire en sorte que l'initialisation nécessaire soit effectuée.

Tout d'abord, le matériel pris en charge doit être localisé parmi les périphériques sur le bus PCI/PCI Express. Ceci est habituellement fait par la recherche d'identifiants spécifiques au périphérique : **Vendor** et **Device ID** (comme spécifié dans l'en-tête de configuration du périphérique PCI Express).

Si un ou plusieurs périphériques sont localisés, ceux-ci doivent être initialisés, et leurs ressources (BARs) doivent être mappées à la plage mémoire de l'OS hôte.

La troisième étape est d'informer le système d'exploitation hôte sur les fonctionnalités que ce pilote devra fournir aux applications. Cela se fait habituellement par le biais d'une structure particulière, dans laquelle un pointeur de fonction est stocké pour chaque type de requête que le pilote peut gérer (par exemple, lecture, écriture, le contrôle des périphériques, etc.)

Le pilote sera créé en tant que module qui sera chargé dynamiquement dans le noyau Linux, et aura besoin de trois fonctions principales, l'une pour l'initialisation, l'une pour gérer le déchargement du pilote, et l'autre pour traiter les requêtes.

2.1.2.1. Détection et initialisation du périphérique : fonction *init*

La méthode *init* (dans notre cas *c4pcie_init()*), effectue l'initialisation complète du périphérique sur le bus PCIe lorsque le module correspondant au pilote est chargée. Le prototype de la fonction est du type :

```
static int __init c4pcie_init (struct pci_dev *dev, const struct
pci_device_id *id);
```

Remarque : `__init` indique au noyau que la fonction *c4pcie_init()* est uniquement utilisée lors de la phase d'initialisation et peut donc être désallouée pour conserver la mémoire une fois que l'exécution de la fonction soit terminée.

La fonction *c4pcie_init()* permet en premier lieu d'informer le noyau sur le périphérique pour lequel le pilote a été écrit puis dans un second temps l'attribution au périphérique d'un numéro majeur et mineur.

Numéro majeur et mineur :

Le répertoire /dev *contient les pseudo-fichiers ou devices associés aux pilotes de périphériques. Les pilotes sont en effet accessibles à travers des fichiers spéciaux appelés également noeuds (ou nodes). Ces fichiers sont caractérisés par deux valeurs numériques :*

- *le majeur ou major qui identifie le pilote ;*
- *le mineur ou minor qui représente une sous-adresse en cas de présence de plusieurs périphériques identiques, contrôlés par un même pilote*

La suite de la fonction effectue la réservation de l'espace mémoire du périphérique sur le bus PCIe et l'accès aux registres à travers le jeu de fonctions suivant :

```
int pci_read_config_byte(struct pci_dev *dev, int addr, u8 *val);
int pci_read_config_word(struct pci_dev *dev, int addr, u16 *val);
int pci_read_config_dword(struct pci_dev *dev, int addr, u32 *val);
```

2.1.2.2. Libération du périphérique : fonction *exit*

Lorsque le pilote est déchargé, tout ce qui a été alloué dans la fonction *c4pcie_init()* doit être désalloué, ceci est réalisé dans la fonction *c4pcie_exit()* (voir annexe 2) dont le prototype est le suivant :

```
static void __exit c4pcie_exit(void);
```

Cette fonction permet de supprimer les fonctionnalités du pilote enregistrées dans le noyau, pour empêcher les applications dans l'espace utilisateur de faire appel au pilote. Ensuite, le périphérique PCIe est désactivé, et le noyau est informé que le périphérique n'est plus en usage. Enfin, les numéros mineur et majeur assignés au périphérique pendant la phase d'initialisation sont supprimés pour pouvoir être allouées à d'autres périphériques.

Espace noyau et espace utilisateur :

Les versions normales du noyau Linux utilisent deux espaces de mémoire :

- *L'espace dit "utilisateur" (ou user space). Cet espace mémoire correspond à l'espace d'exécution des programmes applicatifs.*
- *L'espace dit " noyau" (ou kernel space). Cet espace correspond à l'espace d'exécution du noyau Linux et de ses extensions (modules).*

L'architecture de Linux est décrite par le schéma ci-dessous **[12]** :

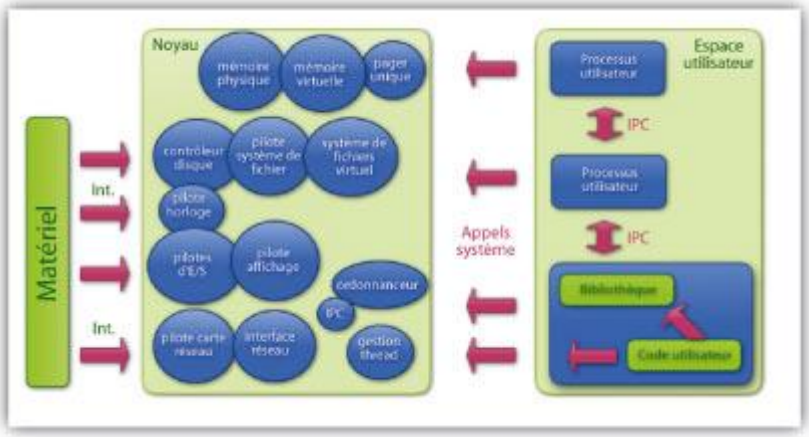


FIGURE 31 : ARCHITECTURE DU NOYAU LINUX

Depuis une application, on peut effectuer une requête au noyau en effectuant un appel système.

2.1.2.3. Lecture sur le périphérique: fonction *read*

La fonction *read()* permet de transmettre les données lues sur le périphérique vers l'espace utilisateur. Son prototype est le suivant :

```
ssize_t read(struct file *filp, char __user *buff, size_t count, loff_t
*offp);
```

L'argument `filp` est un pointeur sur le fichier représentant le périphérique et `count` est la taille de la donnée à transmettre.

Le transfert de données de l'espace noyau vers l'espace utilisateur est réalisé par la fonction noyau `copy_to_user()` dont le prototype est le suivant :

```
unsigned long copy_to_user(void __user *to, const void *from, unsigned long
count);
```

2.1.2.4. Ecriture sur le périphérique: fonction *write*

La fonction *write()* est très proche de la fonction *read()* sauf que l'on utilise la fonction *copy_from_user()* pour faire passer les données de l'espace utilisateur vers l'espace noyau.

2.1.2.5. Configuration du périphérique: fonction *ioctl*

La méthode *ioctl* permet d'effectuer une opération sur un périphérique, celle-ci n'étant pas réalisable par une lecture ou une écriture.

La fonction *ioctl* est appelée à chaque fois qu'une requête *ioctl* est reçue par le pilote. La fonction est déclarée comme ceci :

```
static int c4pcie_ioctl( struct inode* inode, struct file *filp, unsigned
int cmd, unsigned long arg);
```

Les paramètres `inode` et `filp` correspondent au descripteur de fichier ouvert par une application dans l'espace utilisateur. Le paramètre `cmd` contient l'indice du code de contrôle

de requêtes, et le paramètre `arg` (facultatif) représente la requête que l'application dans l'espace utilisateur souhaite passer au pilote.

2.1.3. Analyse de la solution

L'API PCIe fournit par Linux offre une couche d'abstraction très simple permettant une programmation de pilotes quasi-indépendante du matériel, un autre avantage majeur de la conception de pilotes sous Linux, est le fait de pouvoir les gérer sous forme de modules pouvant être dynamiquement chargés dans le noyau Linux en cas de besoin d'utilisation, ce qui permet d'économiser d'une manière performante les ressources mémoires du système utilisé.

Malheureusement, les fonctions inclus dans l'API PCIE sous Linux ne permettent pas d'adresser des registres de tailles supérieurs à 32 bits et n'intègre pas de fonctions pour la gestion des transferts DMA qui est l'élément clef de l'architecture matérielle utilisée.

2.2. Windriver

WinDriver est une solution propriétaire comprenant un ensemble d'outils pour le développement de pilotes qui simplifie grandement la création de pilotes de périphériques, elle comprend un environnement de développement graphique, une API PCIe pour le développement d'applications utilisateurs beaucoup plus riche que celle fournit par Linux et des utilitaires graphiques de diagnostic et de débogage.

2.2.1. Pilotes PCIe Windriver

Windriver permet de développer rapidement des pilotes performants sans nécessiter de connaissances préalables dans le développement des pilotes de périphériques et dans la programmation système et est conçu pour être indépendant du matériel. Cette souplesse offre aux développeurs la possibilité de concevoir des pilotes dans divers environnements et configurations sans besoin de réécriture du pilote. En outre, WinDriver, comprend des fonctions optimisées pour les chipsets d'Altera.

Contrairement aux pilotes sous Linux qui sont développés dans le mode noyau, les pilotes Windriver sont développés directement dans le mode utilisateur, ce qui permet de raccourcir considérablement le temps de développement et de faciliter le débogage du pilote par la suite puisque l'interaction entre l'espace noyau et l'espace utilisateur est éliminé et nous n'avons plus besoin de faire appel aux appels systèmes pour faire l'interface entre les deux modes.

Les sections critiques du code demandant des performances élevées comme la gestion des interruptions peuvent être exécutées dans le mode noyau. Le reste du code du code est toujours exécuté dans le mode utilisateur, les parties critiques sont développé sous forme de modules (Kernel Plugin). L'avantage de cette méthode est de pouvoir garder la facilité du développement dans le mode utilisateur tout en gardant les performances d'un pilote développé dans le mode noyau.

La figure suivante donne un aperçu de l'architecture d'un pilote développé avec Windriver **[17]**.

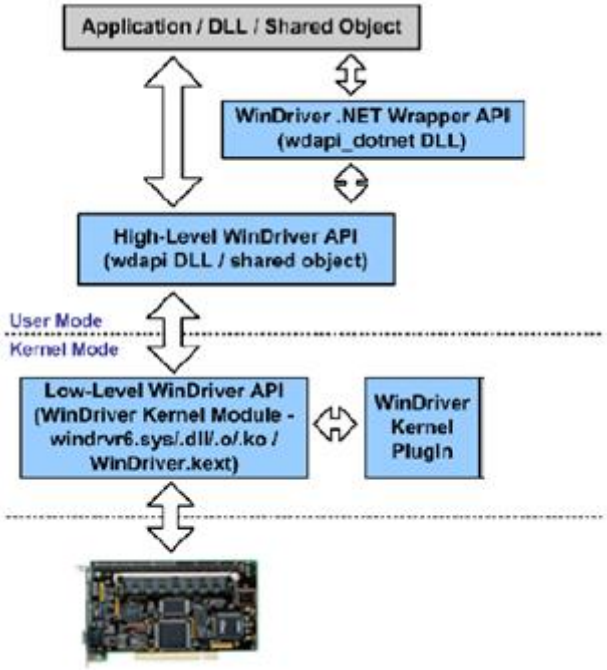


FIGURE 32 : DIAGRAMME DE L'ARCHITECTURE DE WINDRIVER

La génération d'un pilote PCIe à l'aide de Windriver est détaillée dans l'**annexe 3**.

2.2.1. Analyse de la solution

Windriver est une solution qui permet de générer facilement et rapidement des pilotes PCIe fiables et performant tout en offrant une API PCIe très riche dotée de fonctions de haut niveau permettant le transfert de données de différentes tailles ainsi que la gestion des transactions DMA.

Comme cité plus haut, Windriver est une solution propriétaire de la société Jungo dont la licence est très chère (6000 $), mais une version de démonstration est librement téléchargeable à partir du site web de la société éditrice, cette version intègre toutes les fonctionnalités de la version payante mais elle est limitée à une période d'utilisation de 30 jours.

Nous avons opté pour cette solution car elle intègre déjà les fonctions dont on aura besoin pour créer les scenarios de test ce qui va nous permettre de diminuer le temps de développement et ainsi mener à bien notre projet

3. Scénarios de test

Les scénarios ont été développés suivant l'architecture matérielle définit précédemment, un transfert de données du PC vers le FPGA via DMA et vice-versa. Deux fonctions ont été développées pour tester la lecture et l'écriture en utilisant la DMA de la carte FPGA. La fonction qui réalise la lecture (PC vers FPGA) est appelée *DMA_Read()* alors que la fonction réalisant l'écriture (FPGA vers PC) est appelée *DMA_Write()*.

Les deux fonctions possèdent les prototypes suivants :

- `void DMA_Read(WDC_DEVICE_HANDLE hDev, unsigned int target_addr, int length);`
- `void DMA_Write(WDC_DEVICE_HANDLE hDev, unsigned int target_addr, int length);`

Les deux fonctions font appel aux mêmes arguments :

- `hDev` représente le périphérique PCIe cible.
- `target_addr` est l'adresse mémoire adressable dans le périphérique PCIe.
- `Length` est la taille de la donnée à transférer.

Le flux d'exécution des deux fonctions est le même et est définit dans l'organigramme suivant :

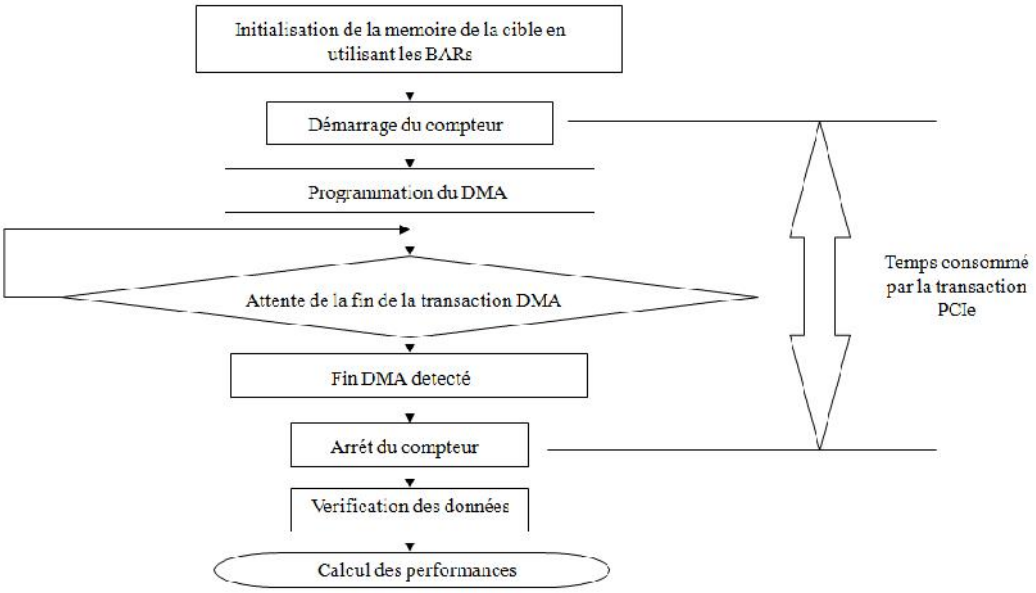


FIGURE 33 : ORGANIGRAMME D'EXECUTION DES FONCTIONS DE TRANSFERTS DMA

La première tâche consiste à utiliser les BARs (voir chapitre 3) pour définir les partitions de la mémoire interne du FPGA (On chip memory) à adresser durant les transferts DMA.

La programmation du DMA permet de définir l'adresse dans laquelle seront écrites respectivement lues les données ainsi que la taille de chaque transfert. Le flux de données est envoyé en boucle vers la mémoire désirée (mémoire du PC pour l'écriture ou mémoire du FPGA pour la lecture) jusqu'à ce que la fin de la transaction DMA soit détectée ce qui déclenche l'arrêt du compteur. Les données sont ensuite vérifiées pour valider le transfert.

Un compteur est utilisé pour calculer le temps consommé par la transaction PCIe pour chaque taille de données envoyée, le temps calculé permettra par la suite de calculer les débits atteints pour chaque transfert.

Plus de détails sur le fonctionnement de chaque fonction est donné dans le chapitre suivant.

4. Conclusion

La solution du pilote PCIe sous Linux offre une flexibilité importante dans l'écriture du pilote, le fait aussi d'utiliser une solution libre permet de s'affranchir des licences payantes, mais demande des connaissances accrues en termes de programmation système et de développement de pilotes.

Vu le temps imparti au projet, nous nous sommes dirigés vers la solution propriétaire Windriver dont les performances sont équivalentes à ceux sous Linux, mais le coût de la licence d'un tel produit n'est pas abordable dans notre cas.

Chapitre 5 : Résultats et Analyse des Performances de la communication PCIe

Dans ce chapitre nous allons tout d'abord représenter les résultats obtenus pour les différents scénarios qui ont été réalisés pour faire ensuite une analyse de ces résultats en se référant aux performances théoriques citées par la spécification du protocole et la documentation du fournisseur de FPGA Altera.

1. Performances théoriques de la liaison PCI Express

La bande passante d'une liaison PCI Express dépends de plusieurs facteurs : la génération du protocole implémentée par les périphériques, l'overhead du protocole, la taille de donnée utile (payload) et le temps de latence de la réponse. Elle dépend aussi des caractéristiques des périphériques connectés sur la liaison et du nombre des lignes de la liaison (x1, x2, …, x32).

1.1. Influence de la génération du protocole

Comme on a déjà cité dans le troisième chapitre, jusqu'à présent, il y a trois générations de protocole PCIe. La génération 1.0 fonctionne à un débit de liaison de 2.5 Gbit/s, la deuxième à 5 Gbit/s tandis que la troisième à 8 Gbit/s.

Le débit de la communication entre deux périphérique est celui du plus bas des deux, la négociation de ce débit se fait par la couche physique au début lorsque liaison est établie. C'est-à-dire que si l'un des deux périphériques fonctionne à 8 Gbit/s et l'autre à 2.5 Gbit/s, alors la communication sera maintenue à un débit de 2.5 Gbit/s.

La carte Cyclone GX d'Altera implémente la génération 1.0 (2.5 Gbit/s). Les tests effectués ont été réalisés en utilisant une liaison x1 et un débit théorique de 2.5 Gbit/s.

1.2. Overhead du protocole

La génération 1.0 utilise le codage 8b/10b, ce qui donne un overhead de 20%. Cela signifie que le débit effectif sera réduit à 2 Gbit/s (250 MBytes/s).

Une liaison active transmet aussi des paquets de la couche de liaison de données (DLLPs) qui ont une taille de 8 octets et des paquets de la couche physique (PLPs) qui ont une taille de 4 octets, ce qui va encore réduire la bande passante de cette liaison. En plus, chaque TLP contient de 20 à 30 octets d'overhead (figure 32).

Start	SequenceID	TLP Header	Data Payload	ECRC	LCRC	End
1 Byte	2 Bytes	3-4 DW	0-1024 DW	1 DW	1 DW	1 Byte

FIGURE 34 : FORME D'UN TLP

La figure suivante représente la bande passante théorique maximale sans prendre en compte les DLLPS et les PLPs pour une opération d'écriture mémoire pour une entête de 3 Double-Words et 4 Double-Words avec et sans le champ du code ECRC.

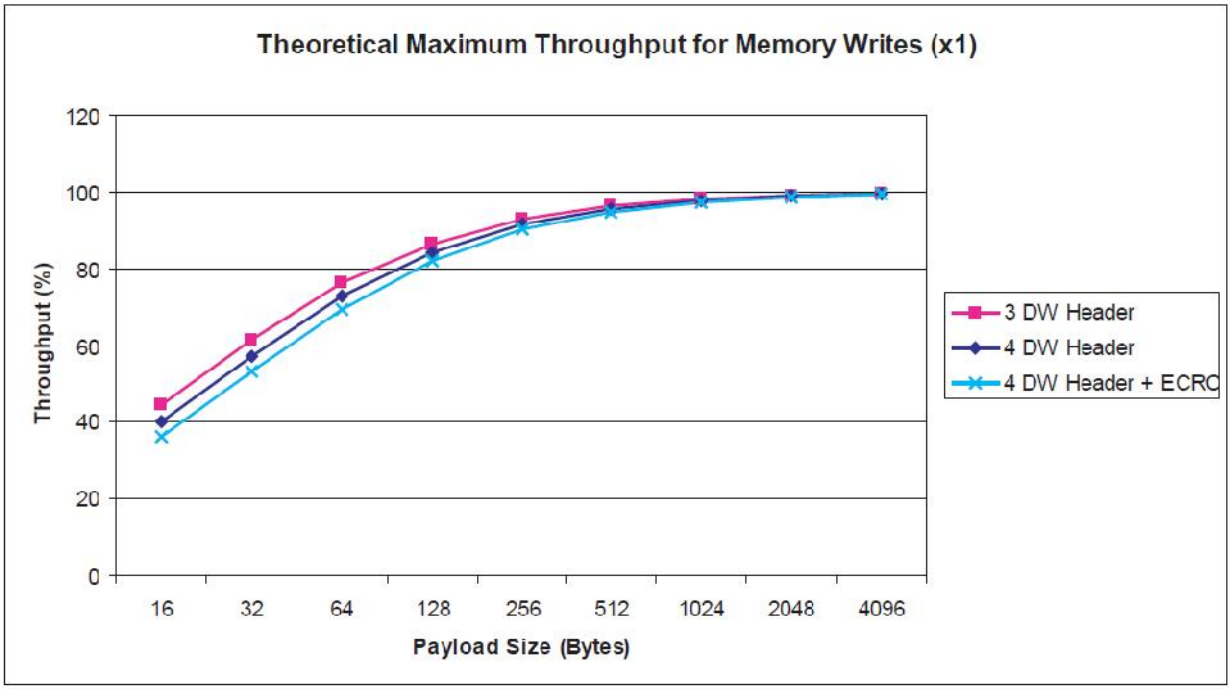


FIGURE 35 : DEBIT THEORIQUE MAXIMAL DU PCIE X1

Le débit théorique maximal calculé par Altera avec les mêmes contraintes des scenarios que nous avons menés est de 230 MBytes/s.

2. Résultats obtenus

Nous avons réalisé deux scénarios en utilisant le DMA implémenté dans le FPGA :

Le premier consiste à effectuer des transactions DMA pour transférer des données de taille différentes de la mémoire physique du PC vers la mémoire implémentée dans le FPGA. Une transaction DMA se fait en deux phases, une phase de lecture de la mémoire source puis la phase d'écriture vers la mémoire destination. Dans ce scénario la phase de lecture est faite à travers le PCI Express (puisque la mémoire source est celle du PC) tandis que l'écriture se fait via le bus Avalon (puisque la mémoire destination est connectée au DMA par le bus Avalon). Pour cette raison nous avons appelé ces transactions « DMA Read », la fonction développée qui permet de lancer ce scénario est la fonction *DMA-Read()* (voir chapitre précédent).

Le second consiste à effectuer des transactions DMA dans le sens inverse (de la mémoire du FPGA vers la mémoire physique du PC). La phase de lecture se fait via l'Avalon (la mémoire source est celle du FPGA) et la phase de l'écriture à travers le bus PCI Express (la mémoire destination est celle du PC). Ces transactions sont donc des transactions d'écriture côté PCI Express (DMA Write), c'est la fonction *DMA_Write()* qui initialise ces transactions et calcule le débit (voir chapitre précédent).

Nous avons effectué ces deux scénarios pour des données de tailles différentes, de 32 octets à 256 Ko. Les résultats que nous avons obtenus pour les deux scénarios sont représentés dans le tableau suivant.

taille de la donnée en Octet	Débit en Mo/s	
	DMA Write	DMA Read
32	4	4
64	8	7
128	16	14
256	31	28
512	61	48
1024	99	76
2048	136	107
4096	163	133
6144	176	146
8192	184	153
12288	192	161
16384	195	165
24576	199	170
32768	202	173
49152	204	175
65536	205	176
98304	206	177
131072	207	178
196608	207	179
262144	208	179

Tableau 6 : Débits obtenus en R/W par transferts DMA

Sur la figure suivante sont représentées les deux courbes de l'évolution du débit par rapport à la taille de données.

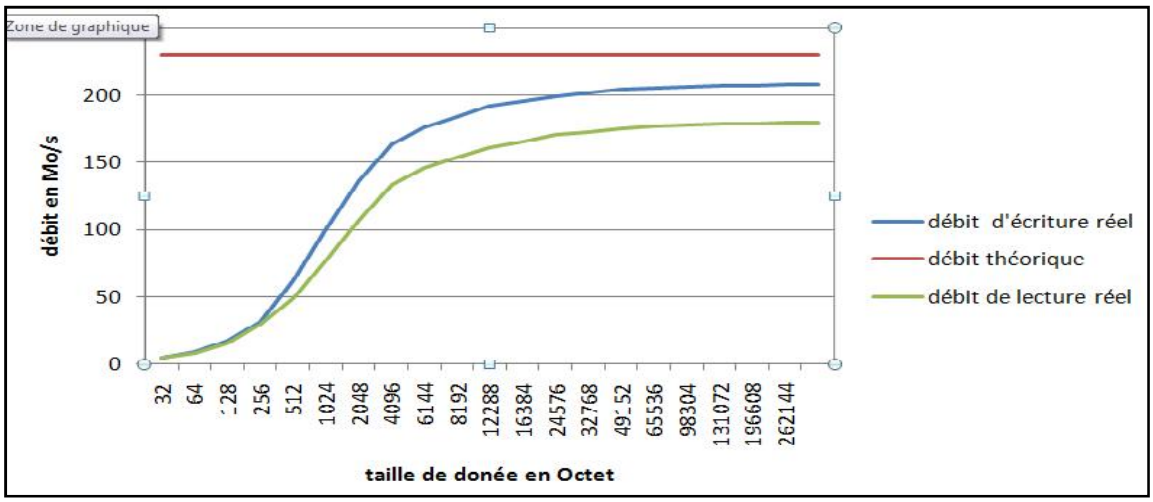


FIGURE 36 : REPRESENTATION DU DEBIT DE TRANSFERT EN FONCTION DE LA TAILLE DE DONNEES POUR LA LECTURE ET L'ECRITURE DMA

Le débit maximal que nous avons obtenu est 208 Mo/s pour l'écriture et 179 Mo/s pour la lecture.

3. Analyse des courbes obtenues

3.1. Evolution des deux courbes

Pour les données de petites tailles le débit est nettement inférieur au débit théorique (230 MBytes/s) pour les deux scénarios, ceci est dû principalement à l'overhead du protocole. Le débit continue à croître d'une manière importante avec la taille de donnée jusqu'à ce qu'il atteigne 163 Mo/s pour l'écriture et 133 Mo/s pour la lecture pour 4 ko de données.

A partir de 4 ko qui est la taille maximale du paquet pouvant être envoyée, la croissance du débit diminue d'une manière remarquable jusqu'à ce qu'il se stabilise à partir de 32 ko (le débit dépasse 200 Mo/s) puisque tous les paquets dépassant les 4 Ko sont découpés en paquets de 4 Ko et envoyés en file.

3.2. Comparaison des deux scénarios

Le débit atteint par l'écriture est plus important que celui de la lecture, ceci revient au fait que la lecture se fait en deux étapes, l'étape de demande puis l'étape de réponse alors que l'écriture se réalise en une seule étape qui est l'étape de la demande.

La figure suivante représente la différence du débit entre l'écriture et la lecture par rapport à la taille de données.

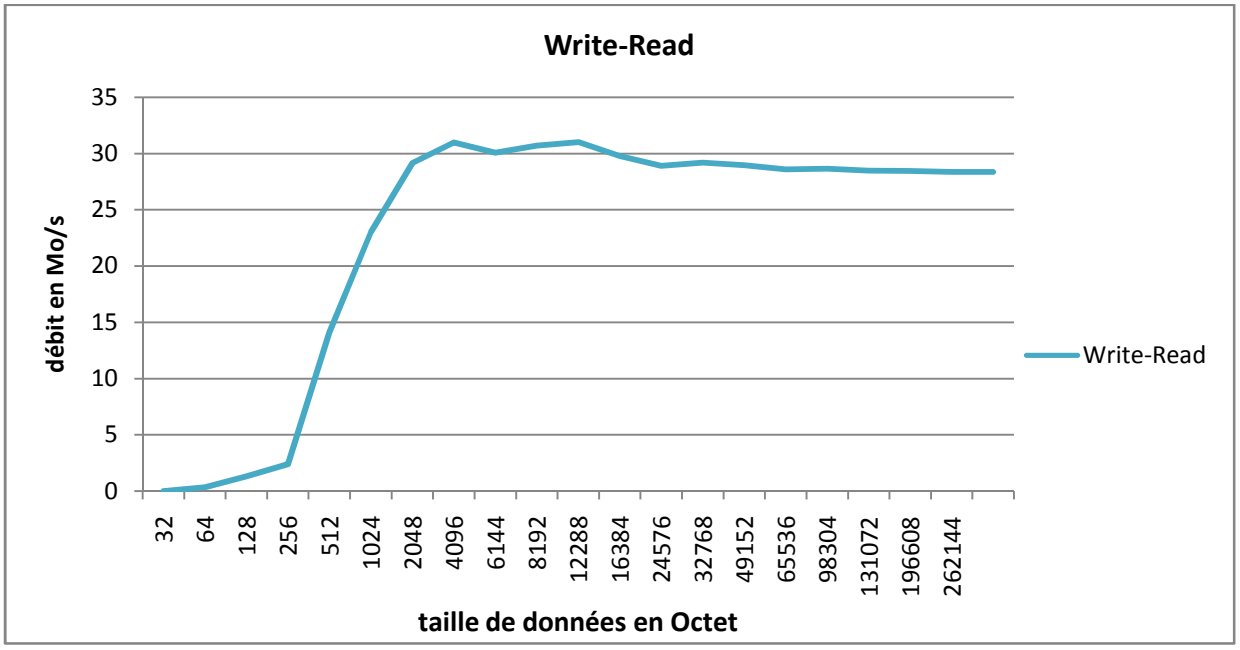


FIGURE 37 : DIFFERENCE ENTRE LA LECTURE ET L'ECRITURE DMA

4. Perspectives

4.1. Test multi-lignes

La carte de développement Cyclone IV sur laquelle nous avons travaillé permet une liaison PCI Express multi-lignes (x1, x2, x4). Côté architecture nous avons fait les trois architectures. Mais nous n'avons pu activer que l'architecture x1 qui nous a donné ces résultats, le test des deux autres architectures (x2 et x4) pourra donner un débit meilleur qui peut dépasser 800 Mo/s pour x4.

4.2. Test du scenario full-duplex

PCI Express possède une architecture qui permet de faire l'émission et la réception en même temps, c'est ce qu'on appelle le full-duplex. Ce mode doit être testé pour voir l'interaction qu'il y a entre les deux sens de transmission.

5. Recommandations

Dans cette partie nous allons émettre quelques recommandations concernant la communication entre un FPGA d'Altera et un PC via le protocole PCI Express pour avoir de meilleures performances. Ces recommandations sont basées sur l'étude des ressources matérielles fournies par les FPGA d'Altera (Xilinx aussi offre des FPGA et des Hard IP pour le PCI Express, mais puisque nous avons travaillé avec un produit d'Altera, on a plus de connaissances sur ses produits que sur ceux de Xilinx).

5.1. La génération 2 du PCI Express avec le FPGA Stratix IV GX

La carte de développement Stratix IV GX d'Altera permet d'implémenter la génération 2 du PCI Express (5Gbit/s) avec un nombre de ligne de 8 (x1, x2, x4, x8). Avec ces conditions la bande passante sera doublée puis multipliée par le nombre de lignes utilisé.

Une architecture avec x8 permet d'avoir le débit théorique suivant :

$$\text{Débit [GBytes/s]} = 5*(8/10)*8$$

$$= 4 \text{ GBytes/s}$$

Le tableau suivant représente les résultats des tests réels effectués par Altera sur cette carte pour une architecture matérielle semblable à celle qu'on a utilisée (une mémoire interne et un DMA) pour différentes implémentations de l'IP PCIe.

Configuration	DMA Reads (MB/s)	DMA Writes (MB/s)	Simultaneous DMA Read/Writes (MB/s)	Theoretical Maximum Throughput (MB/s)	
				Read	Write
Hard IP Implementation—Stratix IV GX					
Gen2 ×8, 128-bit	3304	3434	2956/2955	3710	3710
Gen2 ×4, 64-bit	1727	1775	1691/1631	1855	1855
Gen2 ×4, 128-bit	1708	1783	1684/1484	1855	1855
Gen2 ×1	448	450	438/425	463	463
Gen1 ×8, 64-bit	1706	1778	1680/1628	1855	1855
Gen1 ×8, 128-bit	1694	1778	1678/1480	1855	1855
Gen1 ×4	875	890	855/815	927	927
Gen1 ×1	224	225	219/211	231	231

Tableau 7 : Performances PCIe sur FPGA Stratix IV GX

6. Conclusion

Les tests de la communication PCIe entre le FPGA et le PC que nous avons menés ont montré une légère différence dans le débit avec les tests menés par Altera, cette différence est due principalement à la non transparence du système que nous avons utilisés.

Le bus PCIe n'est pas dédié entièrement à notre carte PCIe, d'autres périphériques PCIe sont déjà connectés à ce bus (24 périphériques en total) ce qui influence directement sur le transfert de données du PC vers le FPGA et vice-versa.

Conclusion générale

Ces quatre mois passés au sein de la fondation MAScIR ont été une expérience enrichissante.

Ce stage m'a permis d'approfondir mes connaissances acquises durant ma formation en répondant aux différentes tâches définies dans le cahier des charges. Dans la première phase du stage, j'ai commencé par étudier l'architecture du DSP C6678 puis de maitriser l'ensemble des techniques nécessaires au portage du noyau Linux sur le DSP. Dans un second temps j'ai été amené à apporter des solutions à la problématique du multiboot sous Linux sur le DSP C6678. Ces solutions doivent être intégrées dans le système de supervison de la chaîne RADAR.

Dans la seconde phase du stage, j'ai été amené à étudier les communications via le protocole PCIe entre le circuit FPGA Cyclone IV et un PC. Ce protocole permettra de relier la chaîne de prétraitement des signaux dans le système RADAR à un ordinateur ayant pour but le contrôle des circuits FPGAs intégrés dans cette chaîne. Pour cela j'ai été amené à développer un pilote PCIe permettant le transfert de données entre le deux architectures puis a développer un ensemble de fonctions et de scénarios de test des transferts DMA du FPGA vers le PC et vice-versa.

Documents annexes

Annexe 1 : Chargement du noyau Linux sur le DSP C6678

1. Téléchargement et installation des logiciels

1.1. Téléchargement du code source du noyau Linux

La version Linux-c6x utilisée dans ce projet est la version 2.0 GA basée sur le noyau 2.6.34. Cette version utilise la chaîne de compilation (toolchain) GCC 4.5-124 de Code Sourcery pour la compilation croisée pour des cibles DSP de la famille C6000. Cette toolchain est téléchargée en lançant la commande **./prj config** et ne doit pas être téléchargée manuellement **[12]**.

Plusieurs dépendances doivent être compilées avant de passer à la compilation du noyau Linux. Ces dépendances sont résumées dans le tableau suivant :

Dépendance	Utilisation
CCSv5.0.3.00028	JTAG debug
TI CGT 7.2.2	Compilation bootloader, Syslink
BIOS 6.32.01.38	Compilation des exemples Syslink
IPC 1.23.01.26	Compilation Syslink
XDC tools 3.22.01.21	Compilation Syslink

Toutes ces dépendances sont rassemblées dans le DVD fournit avec la carte d'évaluation.

Les fichiers source du noyau Linux sont téléchargés à l'aide du script bootstrap qui utilise le logiciel de gestion de versions git pour cloner les sources.

Pour cloner les sources de Linux-c6x 2.0, exécuter les commandes suivantes dans un terminal :

```
mkdir ~/my-linux-c6x
cd~/my-linux-c6x
wget http://linux-c6x.org/bootstrap
chmod +x bootstrap
./bootstrap linux-c6x-2.0
```

Le dossier my-linux-c6x aura la structure suivante:

```
my-linux-c6x/
|-- busybox
|-- linux-c6x
|-- linux-c6x-project
|-- projects
|   |--mtd-utils
|   |--packages
|   |--rio-utils
|   |--c6x-linux-mcsdk-demo
|   |--bootloader-support
|   |--c64-eepromwriter
|   |--ibl
|   |--ltp
|   |--package-downloads
```

```
|   |--syslink
|-- gcc-c6x-uclibc
`-- release-reference-src
```

1.2. Intégration des extra-paquets

Les paquets additionnels nécessaires au benchmarking du DSP c6678 doivent être ajoutés dans le dossier ~/my-linux-c6x/projects/packages.

2. Compilation du noyau et du root file system

2.1. Ligne de commande du noyau Linux

Les arguments passés au noyau Linux lors de la phase de compilation sont enregistrés dans le fichier ~/my-linux-c6x/linux-c6x-project/kbuilds/evmc6678.mk

Ce fichier dot être modifié pour y renseigner l'adresse IP de la carte d'évaluation, du PC auquel elle sera connectée, du type de console utilisé pour l'affichage ainsi que du dossier contenant le système de fichiers dans le cas de l'utilisation d'un serveur NFS.

Le contenu du fichier evmc6678.mk modifié est le suivant :

```
# these are mandatory
DEFCONFIG = ti_evmc6678_defconfig
LOCALVERSION = -evmc6678$(ENDIAN_SUFFIX)$(BUILD_SUFFIX)

# these are optional
CONFIGPATCH =
ifeq ($(ENDIAN),little)
CMDLINE=console=cio ip=192.168.0.2:::255.255.255.0::eth0
root=/dev/nfsnfsroot=192.168.0.1:/opt/full-root-c6x-le-
netcp,v3,tcp rw
else
CMDLINE=console=cioip=dhcproot=/dev/nfs
nfsroot=158.218.100.25:/opt/min-root-c6x-le-netcp,v3,tcp rw
endif
```

console=cio : les messages au moment du boot du noyau seront affichés sur la console CIO de CCS.

192.168.0.1 : est l'adresse IP du host (PC Ubuntu)

192.168.0.2 : est l'adresse IP du Target (la carte EVMC6678)

/opt/full-root-c6x-le-netcp : est le dossier contenant le root file system sur le PC Ubuntu.

Les arguments passés au noyau peuvent être modifiés après la compilation du noyau en utilisant l'outil bootblob qui est généré avec le noyau après la fin de la compilation.

```
./bootblob set-cmdline vmlinux-2.6.34-evmc6678.el-linux-c6x-
2.0.0.63.bin "console=cio
ip=192.168.0.2:::255.255.255.0::eth0 root=/dev/nfs
nfsroot=192.168.0.1:/opt/full-root-c6x-le-netcp,v3,tcp rw"
```

Les arguments passés au noyau peuvent être vérifiés par la commande suivante :

```
./bootblob get-cmdline vmlinux-2.6.34-evmc6678.el-linux-c6x-
2.0.0.63.bin
```

2.2. Configuration compilation du noyau Linux

L'exécution de la commande ./prjconfig permet de vérifier et de configurer le PC sur lequel sera compilé le noyau Linux. La commande vérifie toutes les dépendances et génère le fichier setenv qui permet de spécifier et de configurer les différents éléments à compiler.

```
cd  ~/my-linux-c6x/linux-c6x-project
./prj config
[Please edit setenv to specify what to build, or leave it
untouched to build the example configuration]
```

Les variables suivantes doivent être modifiées dans le fichier setenv pour correspondre à la cible evmc6678.

Cible

Permet de spécifier la cible pour laquelle le noyau sera compilé.

```
export KERNELS_TO_BUILD="evmc6678"
```

File system

Le système de fichiers utilisé est full-root qui permet d'intégrer dans le système de fichiers des paquets additionnels spécifiés dans la variable PKG_LIST.

```
export ROOTFS="full-root"
export PKG_LIST="zlib net snmp polarssl ttcp dhrystone nbench-byte
tcpdump iperf openssl ethtool lmbench"
```

Big/little endian

Permet de choisir le type d'endian utilisé.

```
export ENDIAN=little
```

Syslink

Compilation de Syslink

```
export BUILD_SYSLINK=yes
```

Bootloader

Compilation du bootloader IBL.

```
export BUILD_BOOTLOADERS=yes
```

Après avoir modifié le fichier setenv, les variables d'environnement définies doivent être importées dans l'espace de travail courant.

```
Source setenv
```

La compilation du noyau Linux est lancée par la commande suivante :

```
./prj build
```

2.3. Fichiers générés

Si la compilation se termine avec succès, les fichiers générés se trouveront dans le répertoire ~/my-linux-c6x/product.

Le dossier product contient les fichiers suivant :

```
$ cd ~/my-c6x-linux/product
$ ls -l
total 98396
-rwxr-xr-x 1 rachid rachid    11018 Dec 10 06:42 bootblob
drwxr-xr-x 4 rachid rachid     4096 Dec 10 06:42 bootblob-
templates
-rwxr-xr-x 1 rachid rachid  3957248 Feb  4 16:16 evmc6678-
jffs2.el-hf-dev-rachid-20120204.bin
-rw-r--r-- 1 rachid rachid 43821116 Feb  4 16:16 evmc6678-
jffs2.el-hf-dev-rachid-20120204.jffs2
-rwxr-xr-x 1 rachid rachid  3957248 Feb  4 16:16 evmc6678-nfs.el-
hf-dev-rachid-20120204.bin
-rw-r--r-- 1 rachid rachid 20069172 Feb  4 16:16 evmc6678-nfs.el-
hf-dev-rachid-20120204.cpio.gz
-rw-r--r-- 1 rachid rachid 17139896 Feb  4 16:16 full-root-c6x-
hf.cpio.gz
-rw-r--r-- 1 rachid rachid   145045 Feb  4 16:16 gplv3-devtools-
c6x-hf.cpio.gz
-rw-r--r-- 1 rachid rachid    52064 Feb  4 16:16
i2crom_0x51_c6678_le.bin
-rwxr-xr-x 1 rachid rachid     7792 Dec 10 06:42 make-filesystem
-rw-r--r-- 1 rachid rachid   540606 Feb  4 16:16 modules-2.6.34-
evmc6678.el-dev-rachid-20120204.tar.gz
-rw-r--r-- 1 rachid rachid  2387165 Feb  4 16:16 syslink-demo-
evmc6678.el-hf-dev-rachid-20120204.tar.gz
-rwxr-xr-x 1 rachid rachid  4676407 Feb  4 15:56 vmlinux-2.6.34-
evmc6678.el-dev-rachid-20120204
-rwxr-xr-x 1 rachid rachid  3957248 Feb  4 15:56 vmlinux-2.6.34-
evmc6678.el-dev-rachid-20120204.bin
```

Le noyau Linux en format binaire (avec une extension .bin) et en format ELF (sans extension).

Le root file system sous format cpio.gz.

L'outil bootblob pour pouvoir changer le kernel command line de l'image binaire du noyau.

3. Configuration du NFS filesystem

1- Copier le fichier full-root-c6x-hf.cpio.gz dans /opt/min-root-c6x-le

```
cd ~/my-linux-c6x/product/
sudo cp full-root-c6x-hf.cpio.gz /opt/full-root-c6x-le-netcp
```

2- Décompresser l'archivefull-root-c6x-hf.cpio.gz.

```
cd /opt/full-root-c6x-le-netcp
sudo su
gunzip full-root-c6x-hf.cpio.gz
cpio –idmv < full-root-c6x-hf.cpio
```

Le système de fichier a la structure suivante :

```
cd /opt/full-root-c6x-le-netcp/
ls -l
total 41036
drwxr-xr-x  2 rachid rachid     4096 Feb  5 21:23 bin
drwxr-xr-x  2 rachid rachid     4096 Feb  5 21:11 boot
drwxr-xr-x  6 rachid rachid     4096 Feb  5 21:11 dev
drwxr-xr-x 10 rachid rachid     4096 Feb  5 21:23 etc
-rw-r--r--  1 root   root   41944576 Feb  5 21:22 full-root-c6x-
hf.cpio
drwxr-xr-x  2 rachid rachid     4096 Feb  5 21:11 home
lrwxrwxrwx  1 rachid rachid       12 Feb  5 21:23 init ->
/bin/busybox
drwxr-xr-x  2 rachid rachid     4096 Feb  5 21:11 initrd
drwxr-xr-x  3 rachid rachid     4096 Feb  5 21:23 lib
drwxr-xr-x  2 rachid rachid     4096 Feb  5 21:11 mnt
drwxr-xr-x  4 rachid rachid     4096 Feb  5 21:23 opt
dr-xr-xr-x  2 rachid rachid     4096 Feb  5 21:11 proc
drwxr-x---  2 rachid rachid     4096 Feb  5 21:11 root
drwxr-xr-x  2 rachid rachid     4096 Feb  5 21:23 sbin
drwxr-xr-x  2 rachid rachid     4096 Feb  5 21:11 sys
drwxrwxrwt  2 rachid rachid     4096 Feb  5 21:11 tmp
drwxr-xr-x 14 rachid rachid     4096 Feb  5 21:23 usr
drwxr-xr-x 16 rachid rachid     4096 Feb  5 21:23 var
drwxr-xr-x  4 rachid rachid     4096 Feb  5 21:23 web
```

3- Modification du fichier /etc/exports

Dans le fichier /etc/exports, il faut ajouter la ligne suivante :

```
/opt/full-root-c6x-le-netcp
*(rw,no_root_squash,no_all_squash,sync)
```

4- Redémarrer le service NFS

```
sudo service nfs-kernel-server restart
* Stopping NFS kernel daemon [ OK ]
* Unexporting directories for NFS kernel daemon...
[ OK ]
* Exporting directories for NFS kernel daemon...
exportfs: /etc/exports [2]: Neither 'subtree_check' or
```

```
'no_subtree_check' specified for export "*:/opt/full-root-c6x-le-
netcp".
Assuming default behaviour ('no_subtree_check').
NOTE: this default has changed since nfs-utils version 1.0.x
[ OK ]
* Starting NFS kernel daemon [ OK ]
```

4. Boot du noyau Linux

La carte TMX320C6678L doit être en mode no-boot pour pouvoir la connecter au CCS via JTAG.

Pour cela, les switchs DIPs doivent être sous la forme suivante :

Switch	Pin1	Pin2	Pin3	Pin4
SW3	OFF	ON	ON	ON
SW4	ON	ON	ON	ON
SW5	ON	ON	ON	ON
SW6	ON	ON	ON	ON

4.1. Préparation du Host (PC Ubuntu)

Avant de charger le noyau Linux sur la carte :

1- Le pare-feu doit être désactivé :

```
sudo service ufw stop
```

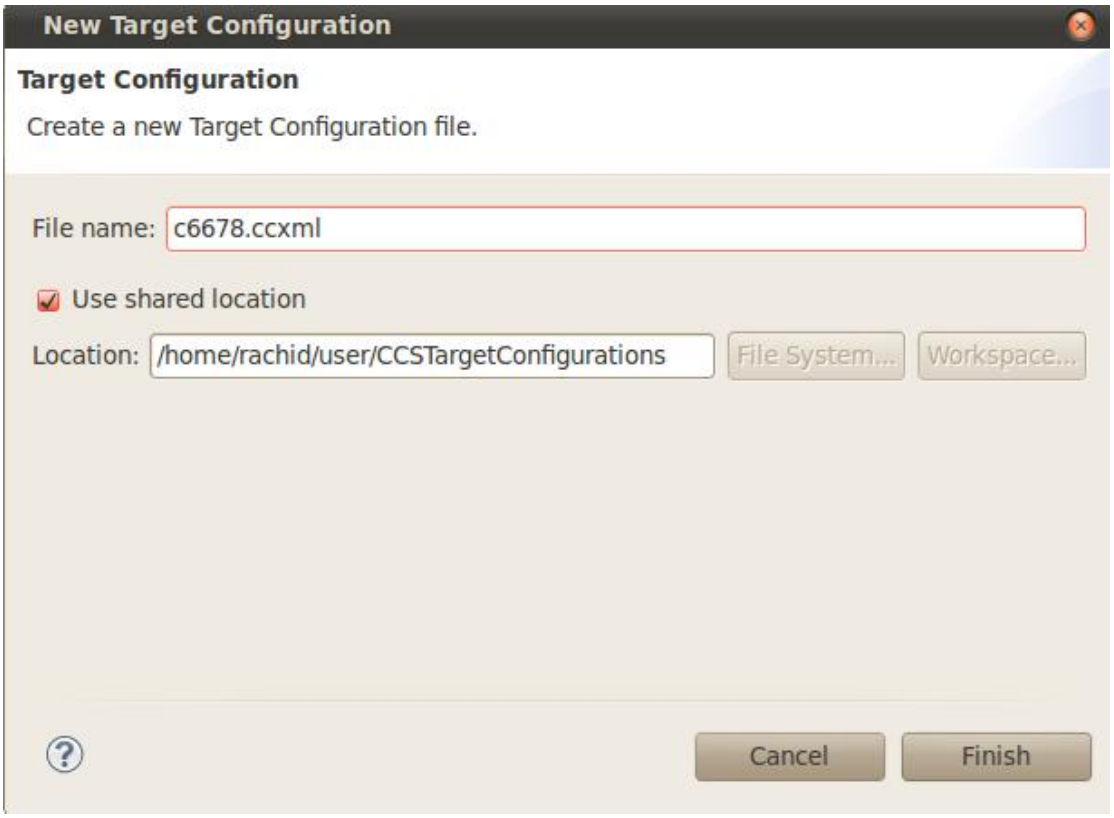
2- Affecter une adresse IP au Host :

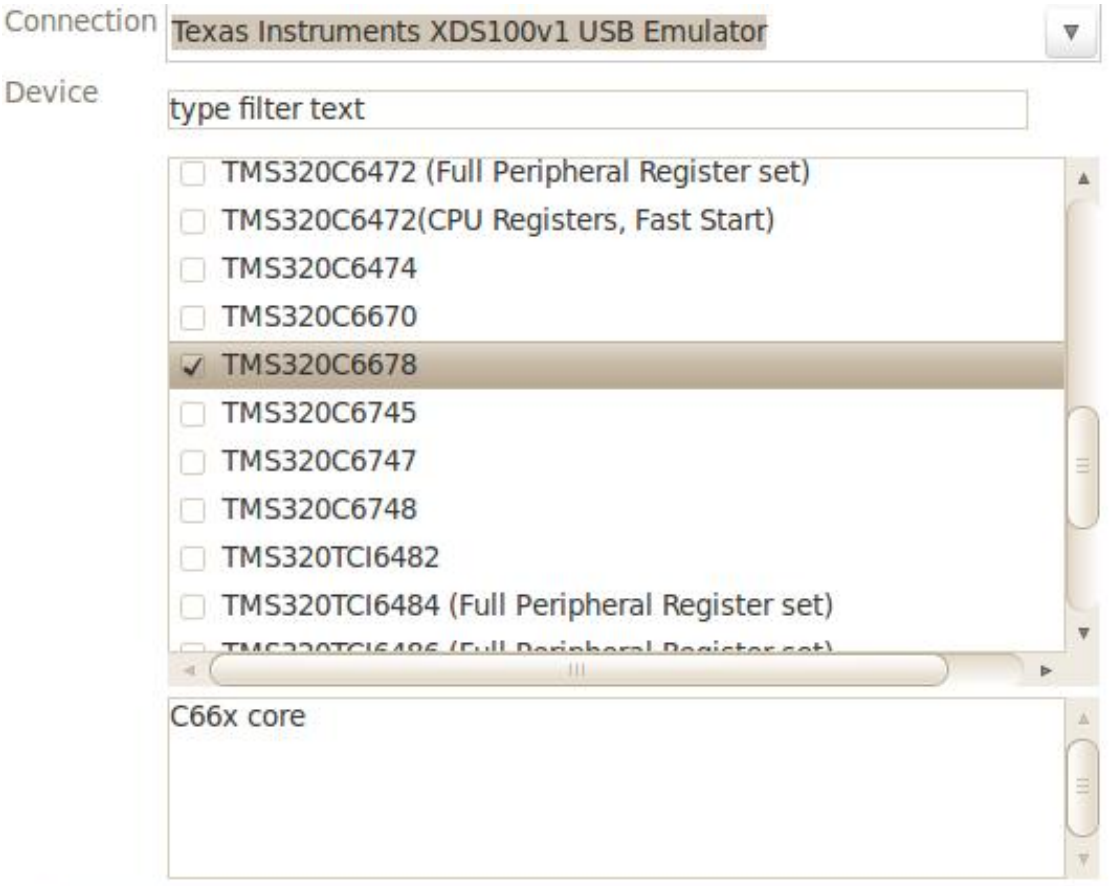
```
ifconfig eth0 192.168.0.1
```

4.2. Chargement et exécution

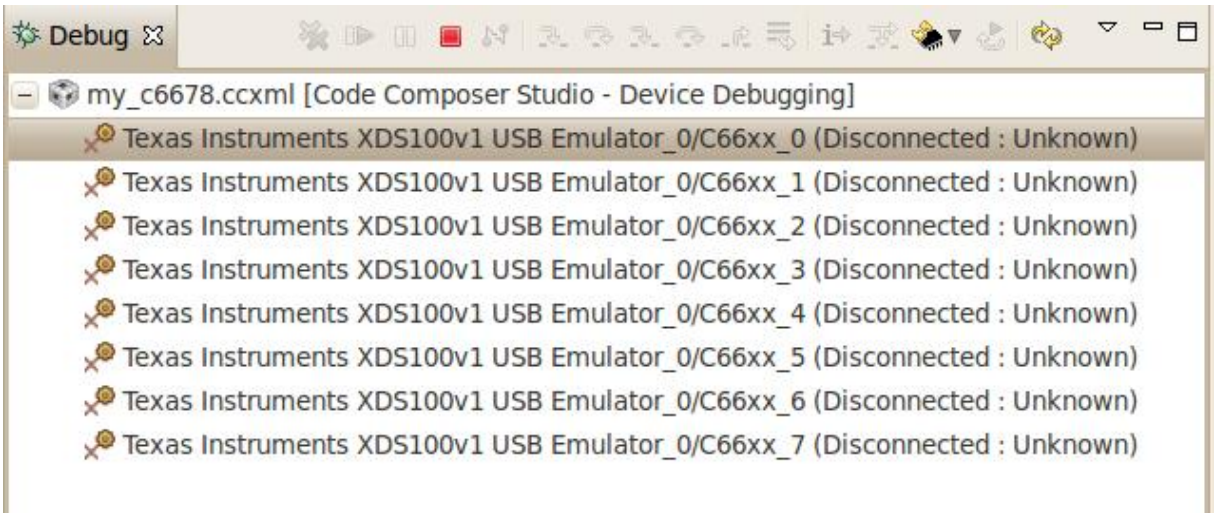
1- Ouvrir le CCS après avoir mis sous tension la carte et branché le câble USB JTAG.

2- Définir une nouvelle configuration pour la carte (View> Target Configuration)

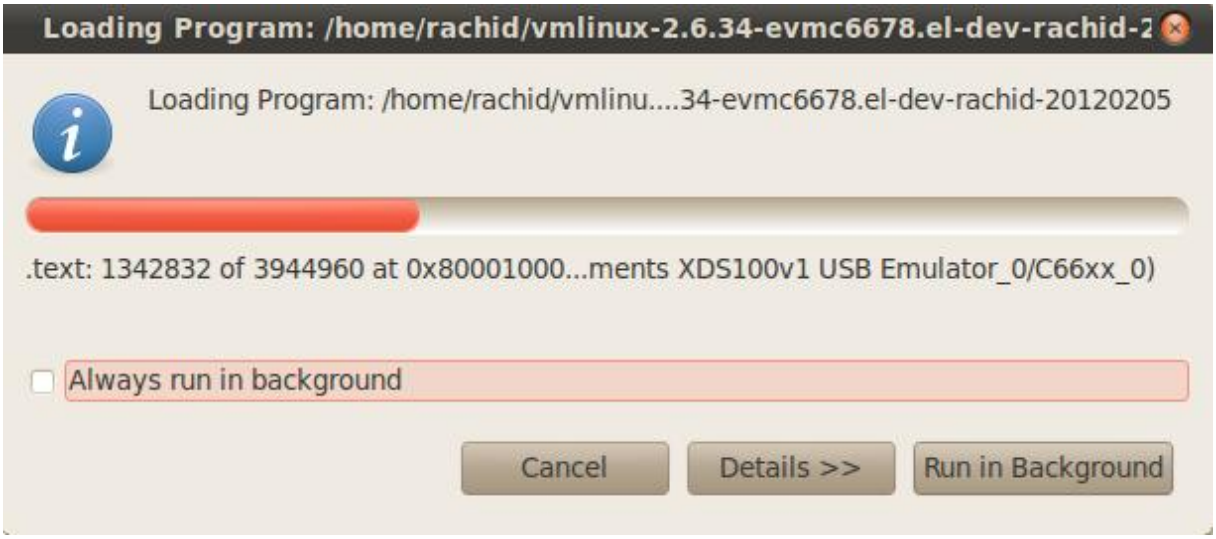

New Target Configuration
Target Configuration
Create a new Target Configuration file.
File name: c6678.ccxml
Use shared location
Location: /home/rachid/user/CCSTargetConfigurations
File System...
Workspace...
Cancel
Finish


Connection
Texas Instruments XDS100v1 USB Emulator
Device
type filter text
TMS320C6472 (Full Peripheral Register set)
TMS320C6472(CPU Registers, Fast Start)
TMS320C6474
TMS320C6670
TMS320C6678
TMS320C6745
TMS320C6747
TMS320C6748
TMS320TCI6482
TMS320TCI6484 (Full Peripheral Register set)
C66x core

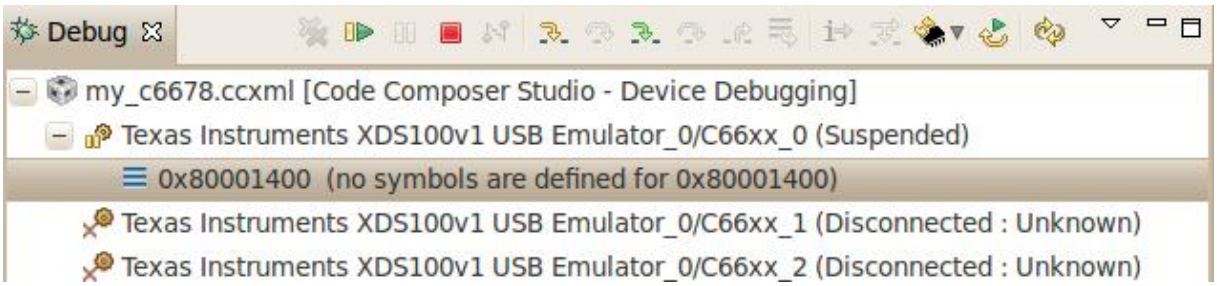
3- Connecter le cœur 0 DSP (clic droit sur le core 0 > connect target)



4- Charger le noyau Linux (Run > Load > Load Program)



5- Lancer l'exécution



La console cio de CCS affiche les messages de boot du noyau Linux :

```
[C66xx_0] Linux version 2.6.34-evmc6678.el-dev-rachid-20120204
(rachid@rachid-laptop) (gcc version 4.5.1 (Sourcery CodeBench Lite
4.5-124) ) #5 Sat Feb 4 15:56:07 WET 2012
[C66xx_0] Designed for the EVMC6678 board, Texas Instruments.
[C66xx_0] CPU0: C66x rev 0x0, 1.2 volts, 1000MHz
[C66xx_0] Initializing kernel
[C66xx_0] no initrd specified
[C66xx_0] Built 1 zonelists in Zone order, mobility grouping on.
Total pages: 130048
[C66xx_0] Kernel command line: console=cio
ip=192.168.0.2:::255.255.255.0::eth0 root=/dev/nfs
nfsroot=192.168.0.1:/opt/full-root-c6x-le-netcp,v3,tcp rw

[….]

[C66xx_0] IP-Config: Complete:[C66xx_0]
[C66xx_0]      device=eth0[C66xx_0] , addr=192.168.0.2[C66xx_0] ,
mask=255.255.255.0[C66xx_0] , gw=255.255.255.255[C66xx_0] ,
[C66xx_0]      host=192.168.0.2, domain=, nis-
domain=(none)[C66xx_0] ,
[C66xx_0]      bootserver=255.255.255.255[C66xx_0] ,
rootserver=192.168.0.1[C66xx_0] , rootpath=[C66xx_0]
[C66xx_0] Looking up port of RPC 100003/3 on 192.168.0.1
[C66xx_0] Looking up port of RPC 100005/3 on 192.168.0.1
[C66xx_0] VFS: Mounted root (nfs filesystem) on device 0:11.
```

Si le noyau démarre avec succès, le message suivant sera affiché pour donner la main à l'utilisateur :

```
[C66xx_0] System started.[C66xx_0]
[C66xx_0]
[C66xx_0]
starting pid 66, tty '/dev/console': '/bin/sh'[C66xx_0]
[C66xx_0] / #
```

4.3. Connexion à la carte

Ouvrir un terminal sur le host (PC Ubuntu) et lancer telnet en lui passant l'adresse IP de la carte.

Si un login est demandé, se sera root.

```
$ telnet 192.168.0.2
Trying 192.168.0.2...
Connected to 192.168.0.2.
Escape character is '^]'.
192.168.0.2 login: root
/root #
```

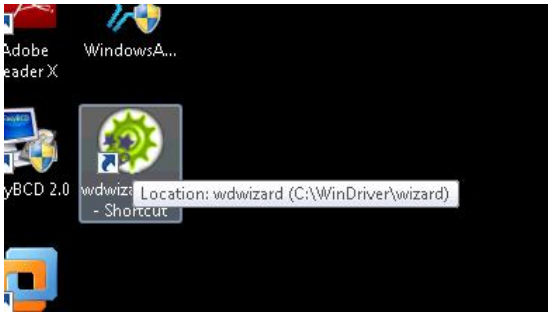
Annexe 2 : fichier de configuration JSON

```
{
    "deviceName" : "C6678",

     "partitions" : [
                {
                     "name"     : "ddr-code",
                    "vaddr"     :  "0x9e000000",
                    "paddr"     : [ "0x81e000000", "0x81e000000",
"0x81e000000", "0x81e000000","0x81e000000", "0x81e000000", "0x81e000000",
"0x81e000000" ],
                    "size"      : "0x1000000",
                    "secNamePat": ["^.text", "const", "switch"],
                    "cores"     : [0,1,2,3,4,5,6,7],
                    "permissions": ["UR", "UX", "SR", "SX"],
                    "cacheEnable": true,
                    "prefetch"   : true,
                    "priority"  : 0,
                     "shared"   : true,
                     "loadPartition" : true
                },
                {
                    "name"      : "ddr-data",
                    "vaddr"     : "0xD0000000",
                    "paddr"     : [ "0x800000000", "0x801000000",
"0x802000000", "0x803000000", "0x804000000", "0x805000000", "0x806000000",
"0x807000000" ],
                    "size"      : "0x1000000",
                    "secNamePat": ["stack", "^.far$", "args", "neardata",
"fardata", "rodata"],
                    "cores"     : [0,1,2,3,4,5,6,7],
                    "permissions": ["UR", "UW", "SR", "SW"],
                    "cacheEnable": true,
                    "prefetch"   : true,
                    "priority"   : 0,
                    "shared"     : false
                }
        ],

        "applications" : [
                {
                    "name"         : "app1",
                    "fileName"     : "../mad-
loader/examples/app_1/build/app_1.exe",
                    "libPath"      : "../mad-loader/examples/shlibs/build",
                    "allowedCores" : [0,1,2,3,4,5,6,7]
                },
                {
                    "name"         : "app2",
                    "fileName"     : "../mad-
loader/examples/app_2/build/app_2.exe",
                     "libPath"      : "../mad-loader/examples/shlibs/build",
                    "allowedCores" : [0,1,2,3,4,5,6,7]
                }
        ],

        "appDeployment" : [
```

```
                "app1",
                "app2",
                "app1",
                "app2",
                "app1",
                "app2",
                "app1",
                "app2"
        ]
}
```

Annexe 3 : génération du pilote PCIe à l'aide de Windriver

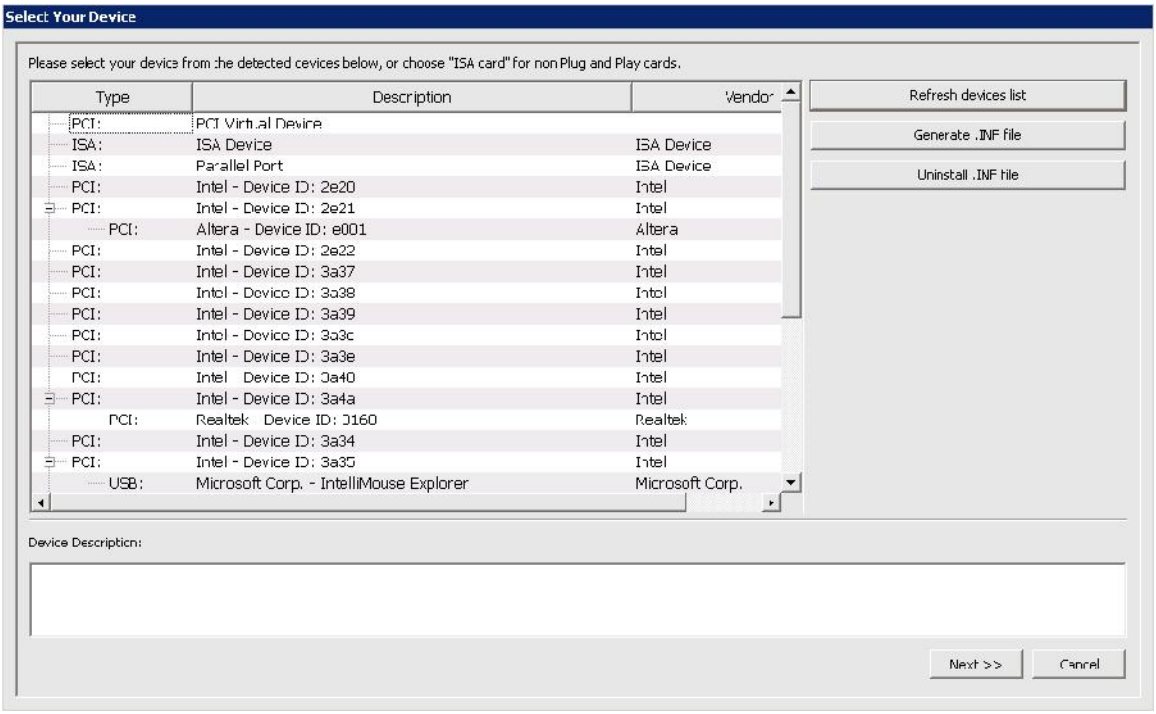
Pour lancer Windriver, double cliquer sur son icone



Puis sélectionner "New host driver project"

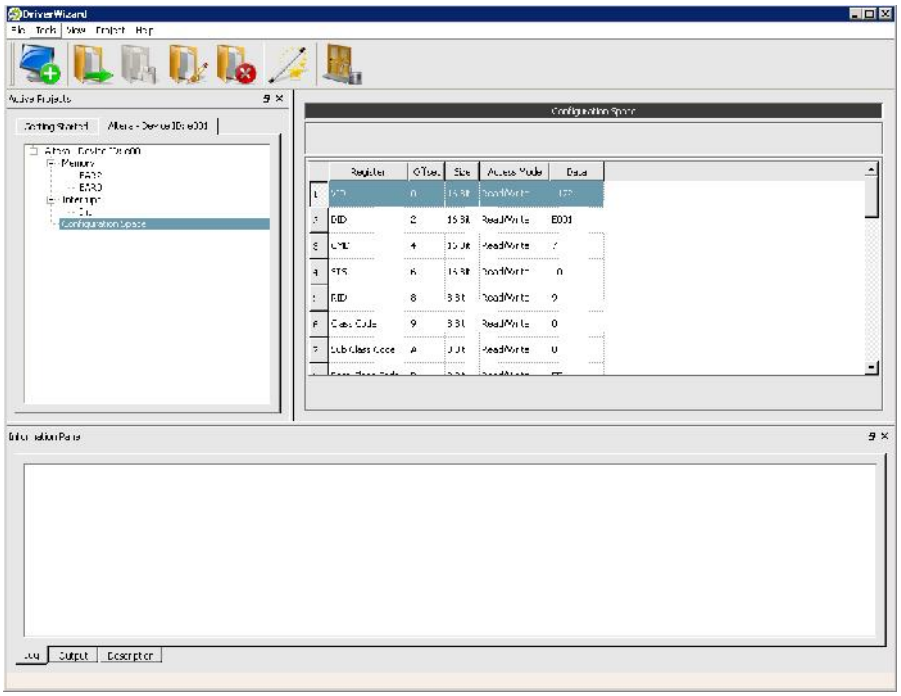
Dans la liste des périphériques affichée par Windriver, la carte Cyclone IV est reconnue par la ligne suivante :

PCI: Altera – Device ID:e001

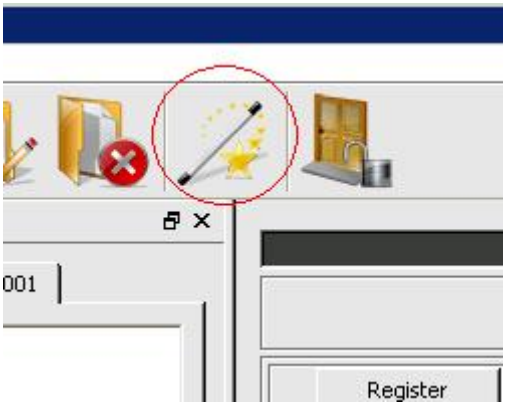


Pour voir l'espace de configuration PCIe du périphérique, il suffit de cliquer sur la ligne correspondante au Cyclone IV.

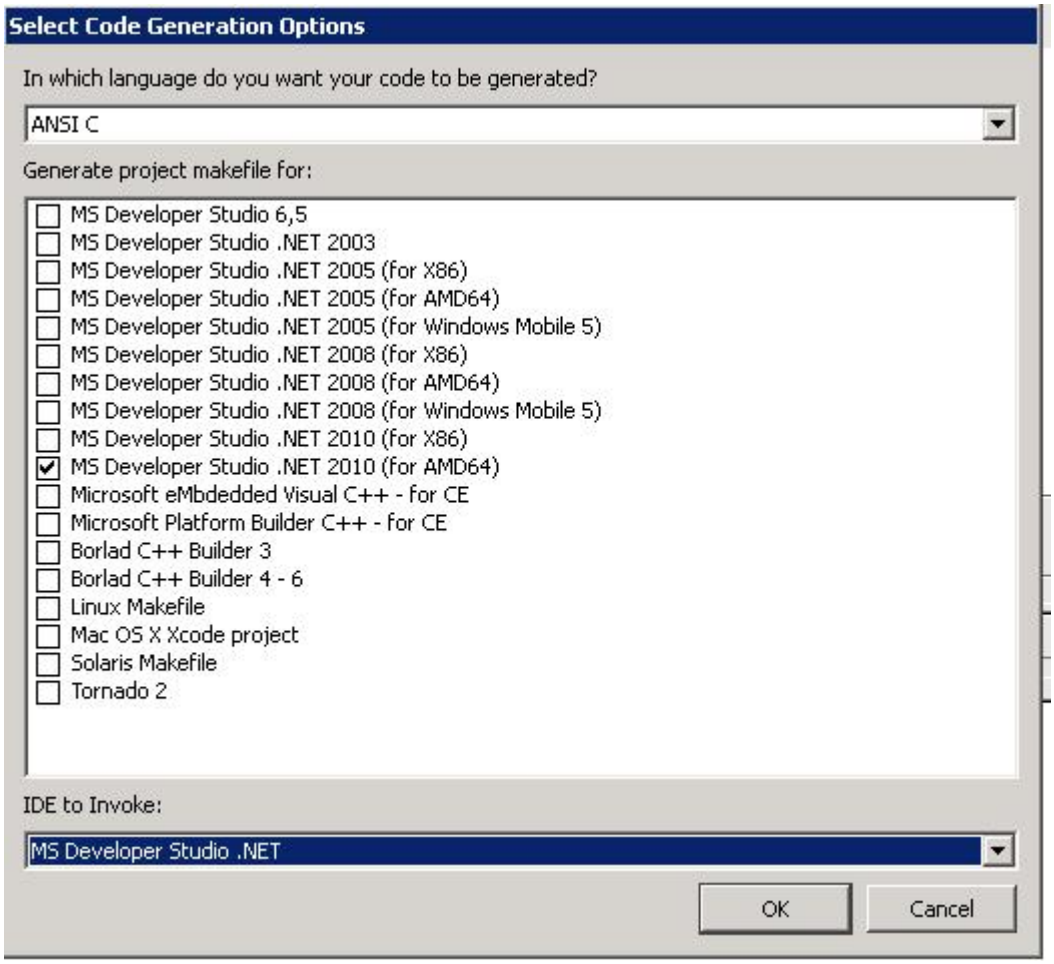
Les valeurs des registres de configuration sont affichées alors dans une fenêtre représentée dans la figure suivante :



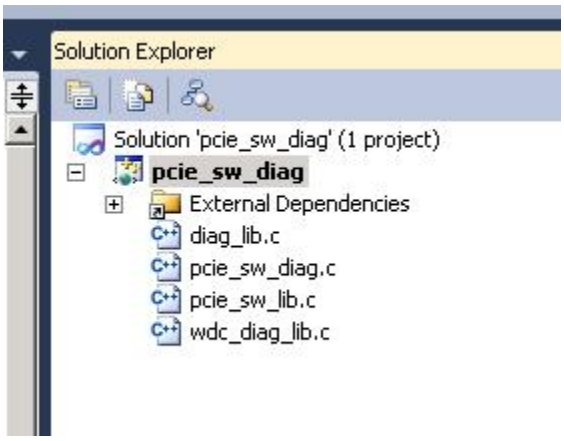
Pour générer le code correspondant au driver PCIe, il faut sélectionner l'icône montrée dans la figure suivante :



Puis sélectionner pour quel type d'environnement le code approprié devra être généré



Les fichiers suivants seront générés (Le nom du projet dans cet exemple est pcie_sw)



Ces fichiers là correspondent au code du pilote PCIe et peuvent être personnalisés par l'utilisateur du pilote.

Références

Bibliographie :

[1] Introduction to Microprocessors and Microcontrollers by John Crisp. Second edition 2004. 287 pages.

[2] C66x CorePac: Achieving High Performance :Texas Instruments

[3] KeyStone C66x Multicore SoC Overview : Texas Instruments

[4]TMS320C6678 Multicore Digital Signal Processor: Texas Instruments.

[5] Multicore Application Deployment : Texas Instruments

[6] IBL : Texas Instruments

[7] PCI Express System Architecture. by Ravi Budruk. Don Anderson. Tom Shanley. 1ère édition.

[8] Embedded Linux primer: a practical real-world approach by Christopher Hallinan.2nd edition 2011. 652 pages.

[9] Pro Linux Embedded Systems by Gene Sally. First edition 2010. 445 pages.

[10] Linux Device Drivers by Rubini Alessandro, Corbet Jonathan, Kroah-Hartman Greg. 3ème édition. 602 pages.

[11] Essential Linux device drivers by Sreekrishnan Venkateswaran**.** 1ère édition. 832 pages.

[12] Linux embarqué par Pierre Ficheux. 3ème édition. 357 pages.

Webographie :

[13] http://linux-c6x.org/wiki/index.php/Main_Page

[14] http://www.altera.com/literature/manual/mnl_avalon_spec.pdf

[15] http://fr.wikipedia.org/wiki/PCI_Express

[16]http://www.jungo.com/st/windriver_driver_development_pci_express.html