TP de programmation DSP pour le traitement d'image

ATEME[1] - Groupe ESIEE

Introduction

Objectifs

Le TP sur carte IEKC6x11 s'étend sur 3 séances de 3 heures chacune. Vous aurez l'occasion durant ces séances d'expérimenter :

- l'utilisation d'une carte DSP pour le traitement d'image basée sur le processeur Texas Instruments TMS320C6211 ou TMS320C6711,
- de coder et d'optimiser sur cette plate-forme différents algorithmes de traitement d'images étudiés en cours,
- d'étudier le contrôleur DMA du DSP C6211 afin d'effectuer des opérations mémoires simples,
- de coder et d'optimiser une portion de code utilisé dans un décodeur MPEG4 Video.

Documentation

Les documents auxquels vous aurez à vous référez durant le TP sont notamment :

- le polycopié du cours ATEME sur l'IEKC6211,
- le polycopié du TD ATEME sur l'Optimisation sur DSP,
- les cours ESIEE de traitement d'images,
- la documentation Texas Instruments notamment :
  Spru190.pdf "TMS320C6000 Peripherals Reference Guide" : ce document décrit le fonctionnement interne du DSP, notamment le contrôleur EDMA/QDMA (chap. 6 et Annexe A),
  Spru401.pdf "TMS320C6000 Chip Support Library API User's Guide" : ce document décrit l'utilisation de la bibliothèque CSL (Chip Support Library) utilisée dans la 2nde partie,
  Spru400.pdf "TMS320C62x Image/Video Library Programmer's Reference" : ce document décrit l'utilisation de la bibliothèque IMGLIB de fonctions de traitement d'images optimisées en assembleur.

ATTENTION :
NE JAMAIS CONNECTER OU DÉCONNECTER L'ALIMENTATION OU LE CONNECTEUR
JTAG OU FAIRE UN RESET **DE LA CARTE TANT QUE CODE COMPOSER EST LANCÉ :**
IL Y A RISQUE DE DESTRUCTION DU DSP
AVANT CES MANIPULATIONS VOUS DEVEZ TOUJOURS VERIFIER
DANS LE GESTIONNAIRE DE PROGRAMME QUE CCS_APP.EXE N'EST PAS EN COURS D'EXECUTION

[1] www.ateme.com

1 Première Partie - Traitement d'images

1.1 Programme de base

1.1.1 Description de la plate-forme d'expérimentation

La carte IEKC6211 peut être connectée à 4 sources vidéo analogiques en entrées et possède une sortie vidéo analogique. Elle repose sur un DSP Texas Instruments TMS320C6211 ou TMS320C6711, un FPGA 50k cellules et plusieurs bancs mémoires RAM (Cf. figure 1).

Cette carte peut fonctionner, selon la configuration du FPGA, soit en acquisition d'images, soit en restitution.

En mode acquisition le signal d'une des 4 entrées analogiques est convertit en un flot numérique par le convertisseur A/N correspondant. Ce flot est ensuite stocké dans la RAM du FPGA par le FPGA. Une fois qu'une image entière est stockée dans la RAM FPGA il faut la transférer dans celle du DSP pour qu'il puisse y faire des traitements (filtrage, compression, etc). Pour cela le FPGA et le DSP sont connectés ensembles par une mémoire RAM double ports (DPRAM) de 16Ko. Cette taille étant insuffisante pour transférer une image entière en une fois ce sont des blocs de 8 lignes qui sont échangés. Côté DSP, on utilise plutôt le DMA pour effectuer rapidement les copies des blocs de 8 lignes vers la RAM du DSP sans gaspiller de puissance cpu.

En mode restitution le processus est totalement symétrique : l'image à afficher est d'abord contenue dans la RAM du DSP. Il s'agit donc de la découper en blocs de 8 lignes pour la transférer dans la RAM du FPGA par le biais de la petite DPRAM. Quand une image entière a été transférée, le FPGA la transmet à l'unique convertisseur numérique-analogique qui est lui même connecté au moniteur vidéo.

Comme tous les processeurs récents, le DSP est équipé d'une interface JTAG qui permet, depuis Code Composer Studio exécuté sur un PC, de contrôler très précisément le DSP : mis en place de points d'arrêts, lecture-écriture des registres internes, accès en lecture-écriture à tous l'espace mémoire. Cette dernière fonctionnalité sera d'ailleurs utilisée pour charger la mémoire du DSP avec un programme et des données.

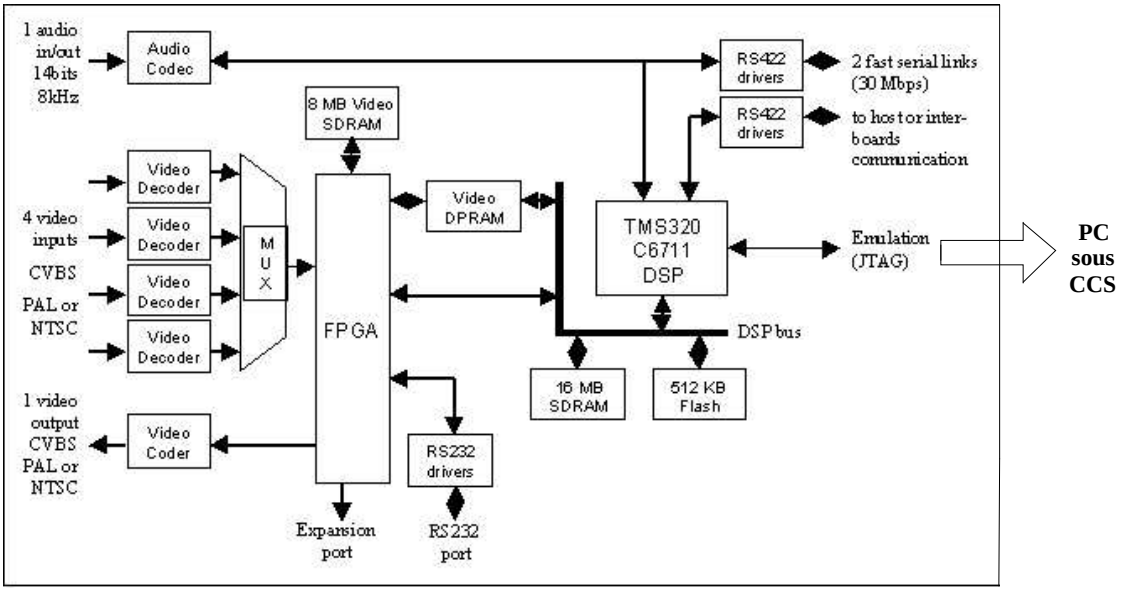

FIG. 1 – Architecture fonctionnelle du kit IEK

1.1.2 Prise en main de Code Composer Studio

Avant de commencer les expérimentations sur des images, nous allons commencer par vous initier à l'utilisation de l'environnement de programmation *Code Composer Studio* (version 2.12). Pour cela vous effectuerez les exercices du tutorial que vous exécuterez à partir du menu "HELP → Tutorial → Code Composer Studio" : tous les exercices de "Developing a simple program" à "Profil Code execution" inclut.

Notes :

– ne pas créer le projet dans le repertoire `c :/ti/Myproject` comme indiqué dans le tutorial en ligne mais créez le dans un sous repertoire de `c :/temp`. N'oubliez de recopiez ce dernier dans votre compte à la fin du TP,
– pour vos carte vous devez lire `target = dsk6711` au moment d'allez chercher le repertoire `volume1`,
– vous devez utiliser la librairie `rts6701.lib`.

1.1.3 Démarrage de l'expérimentation

Avant de commencer il faut copier le contenu du repertoire "c :/Ateme/TP IEK/" dans le repertoire "c :/Ateme/tmp/" dans lequel vous avez les droits d'écriture suffisants. Vous n'oublierez pas de nettoyer ce repertoire à la fin de chaque TP (après en avoir effectué une sauvegarde).

1. Lancer CCS,
2. Charger le projet se trouvant dans le répertoire
   "c :/Ateme/tmp/1ere_Partie_TrtImage/AppliSquelette/video.pjt"

Le programme exemple contient un ensemble de fichiers sources :

– le module main,
– un module d'utilitaire pour l'utilisation de la carte IEKC6211 qui simplifie l'utilisation de la carte dans le contexte du TP,
– les sources du kit de développement de la carte IEKC6211, c'est à dire la bibliothèque de fonctions qui permet de faire fonctionner la carte.

Dans le cadre de ce TP nous nous intéresserons uniquement au module main. Le coeur de ce module est la fonction C `main()` dont le squelette est le suivant : (Ouvrir le source du fichier `main.c` à l'aide de l'explorateur de projet de Code Composer Studio)

```
#define WIDTH    352
#define HEIGHT   280

const far unsigned char ImageRef[] = { #include "image.bmp.Y.txt"};

void main( void )
{

  CSL_init(); board_init();  //init la carte

  image_display(ImageRef);
}
```

Les fonctions utilisée dans `main()` sont externes et implémentées dans le module utilitaire. Leur rôle est le suivant :

– `board_init()` : cette fonction à pour but d'initialiser la carte. Sur une carte telle que celle-ci, il y a toujours un ensemble d'opération à effectuer afin de rendre la carte fonctionnelle. Parmi ces opérations il y a :
  – initialisation de la mémoire
  – initialisation des périphériques internes de la carte (FPGA, interface vidéo, .)
– `image_load()` : cette fonction à pour but de charge dans le tableau passé en 2nd paramètre l'image contenue dans le fichier dont le nom est passé en 1er paramètre. Pour des soucis de simplification, cette fonction ne sait charger qu'un fichier de type BMP contenant une image CIF (352*280) en niveau de gris sur 8 bits,

– `image_display()` : cette fonction permet de transférer le contenu du tableau Image passé en paramètre vers le Fpga et la mémoire SDRAM Vidéo afin de faire afficher cette image sur la sortie vidéo composite.

Pour ce TP, nous n'utiliserons qu'une sous partie des fonctionnalités de la carte IEKC6211 : images **CIF en niveau de gris**. La carte est toutefois capable de travailler aussi en plein format vidéo (720x576) et en couleur (YUV422).

Le fichier "image.bmp.Y.txt" contient la sequence de tous les pixels qui forment l'image encodé sous forme ASCI. Ce fichier est créé avec l'utilitaire de conversion rgbbmp_to-ytxt (dans le même répertoire) qui prend en entrée sur la ligne de commande le nom d'un fichier BMP au format RGB 24 bits et qui crée un fichier texte comportant la composante Y correspondante.

1.1.4 Essai

1. recompiler le programme,
2. "resetter" la carte,
3. charger le programme et l'exécuter : l'image doit alors s'afficher sur la télévision.

1.2 Traitement d'image

1.3 Codage de fonctions

On se propose de coder quelques fonctions de traitement d'images sur la carte DSP. Implémentez au moins 3 fonctions parmi les algorithmes suivants ou ceux étudiés en cours :

– Traitements linéaire : convolution 3x3, 5x5,
– Traitements non-linéaires : filtre médian, seuillage
– Morphologie mathématique : erosion, dilatation (en binaire ou niveau de gris)

Codez au moins la convolution 3x3.

Pour effectuer le codage, créer un nouveau fichier C pour chaque fonction, donner un nom adapté à la fonction qui prendra comme paramètre des pointeurs vers les images d'entrée et de sortie :

Exemple : `convol.c`

```
void convol(  unsigned char *pImageSource, unsigned char *pImageDest, unsigned char *
{
      // coder la fonction
}
```

Puis modifier le fichier `main.c` pour faire appel à la nouvelle fonction de traitement afin d'effectuer le traitement sur l'image préalablement chargée puis afficher le résultat.

Le résultat obtenu à l'affichage est-il bien celui attendu ?

1.3.1 Mesure des performances

A l'aide de la fonction de mesure des performances de de CodeComposerStudio (profiling), déterminez le temps d'exécution en nombre de cycles d'une des fonctions implémentées préalablement. Notez les résultats.

Il s'agit maintenant d'optimiser l'exécution. A l'aide des méthodes étudiées lors du TD Optimisation, optimiser cette fonction. Analysez l'amélioration en étudiant le source assembleur généré par CCS. Mesurez les performances et notez les résultats.

A chaque étape d'optimisation, vérifiez par l'affichage que le résultat obtenu est toujours celui attendu.

1.3.2 Comparaison avec des fonctions optimisées en assembleur

Le document *spru400.pdf* documente une bibliothèque de fonctions d'imagerie optimisées en assembleur et fournie par Texas Instruments : `IMGLIB`.

Étudier la liste des fonctions fournies.

Si la fonction que vous avez choisie d'optimiser fait partie des fonctions proposées (la convolution 3x3 en fait partie), étudiez sa documentation :
- estimez le nombre de cycles qu'elle devrait prendre pour son exécution à partir de la documentation,
- implémentez la fonction de traitement à l'aide de la bibliothèque de Texas Instruments,
- vérifiez que cela fonctionne et confirmez les performances estimées par la mesure.

2 Seconde Partie - Utilisations des DMAs

Comme cela a été abordé dans le cours de présentation de l'IEKC6211, les DMA sur DSP sont très utilisées car leurs possibilités sont extrêmement puissantes.

De la même façon que dans le 2.2, vous allez programmer le contrôleur DMA du DSP TMS320C6211 pour effectuer des opérations simples en créant des fonctions que vous pourrez intercaler dans la chaîne de chargement et de visualisation.

Vous devrez implémenter au moins 3 fonctions parmi les suivantes :

- recopie simple (identique à memcpy)
- retournement horizontal de l'image
- retournement vertical de l'image
- rotation de 180° de l'image
- rotation de 90° du portion d'image

Pour implémentez ces fonctions, vous utiliserez la bibliothèque CSL fournie avec CodeComposerStudio et les fonctions :

CSL_init()	Initialisation de la bibliothèque
EDMA_qdmaConfigArgs()	Pour utiliser la QDMA
EDMA_open()	Pour utiliser la EDMA
EDMA_configArgs()	
EDMA_close()	

Implémentez et visualisez le résultat.

3 Troisième Partie - Optimisation d'une fonction dans le MPEG4

3.1 Mesure de performance de la fonction assembleur

Cette partie du TP à pour but d'étudier une fonction utilisée dans de nombreux algorithmes de compression vidéo : la transformée en cosinus discrète inverse ou IDCT.

Fermez le projet de la partie précédente et ouvrez le projet du décodeur MPEG4. Compilez le programme et exécutez le. Le DSP va décompresser en temps réel un flux MPEG4 embarqué dans sa mémoire et afficher la vidéo correspondante.

Ouvrez le fichier `idct.c` et regardez son contenu. Il fait appel à une fonction `idct_asm()` implémentée en assembleur dans le module `idctasm.asm`. Ouvrez le module `idctasm.asm` et regardez le.

A l'aide de l'outil de mesure de performance (profiler) de CCS, mesurez le temps d'exécution de la fonction assembleur lors de la décompression du flux MPEG4.

Notez les résultats.

3.2 Codage en C de l'IDCT

Dans la fonction C `idct()`, supprimez l'appel à la fonction assembleur et codez vous même en C l'équivalent de la fonction.

- compilez et vérifiez que la décompression MPEG4 fonctionne toujours,
- mesurez les performances de la fonction C,
- comparez avec les résultats de la fonction assembleur.

3.3 Optimisation de l'IDCT

- Essayez d'optimiser votre code pour vous rapprocher des performances de la version assembleur,
- mesurez les performances,
- commentaires.