TP IF4-ARCH

DSP - carte DM6437EVM

Un RAPPORT sera à rendre en Janvier

Téléchargez le fichier TP_DSP_6437.zip sur la page www.esiee.fr/~grandpit et décompressez le dans le repertoire c ;\Temp

Dans cette archive se trouvent plusieurs sous-répertoires :

Tests : contient des fichiers binaires exécutable sur la cible permettant de tester le fonctionnement de la carte

Tp_DSP_Exo1 : ce répertoire contient une application qui exécute en boucle : l'acquisition d'une image sur l'entrée vidéo puis son affichage sur la sortie vidéo

TP_DSP_loop_DMA : ce fichier contient un programme qui en plus de la version précédente effectue une recopie d'image par DMA

Travail à effectuer :

1. Démarrage : boucle « acquisition → Affichage »

Compilez et testez l'application « TP_DSP_exo1 / TP_DSP.pjt »:

- Connectez la carte à l'écran, à la caméra vidéo (tout deux doivent être alimenté), puis alimentez là : l'écran doit alors afficher l'image de la caméra
- Branchez la carte avec le connecteur « mini USB »

- Lancez CCS (Code Composer Studio) à l'aide du bouton droit : « exécutez en tant que » : choisir le login CCS dans le menu déroulant et le password CCS

- Sélectionnez « connect » dans le menu « Debug ». En bas à gauche de la fenetre, CCS doit passer de « Disconnected » à « connected » et l'application de démonstration doit cesser de tourner en boucle.

- Dans CCS faites « Project → Open » pour charger le projet TP_DSP1.pjt

  La fenêtre de gauche présente les fichiers sources du projet. La boucle principale d'acquisition-traitement-affichage se trouve dans le fichier fonctions.c dans la fonction «**start_boucle** ». Cette fonction est lancée par la fonction **tachevideo** elle-même lancée par l'ordonnanceur de DSP/BIOS (le micro noyau DSP) comme indiqué dans le DSP/BIOSConfig/TP_DSP.tcf→Scheduling→TSK

- Compilez le programme
- La fenêtre du bas affiche les messages liés à la compilation. Si tout se passe bien un fichier binaire « TP_DSP.out » à été construit dans le sous-répertoire DEBUG.
- Télécharger ce programme sur le kit à l'aide du menu « File→ Load Program »
- Lancez l'exécution du programme avec la commande « Debug→ Run » (ou à l'aide du personnage qui court sur la gauche de l'écran).

- Vous devez alors voir l'image de la caméra affichée en couleurs à l'écran.
- Vous pouvez ouvrir une fenêtre de log en sélectionnant « DSP/BIOS→Message log ». Cette fenêtre affiche le contenu de tous les appels à la fonction « LOG_printf »

Question :

Double cliquez sur le fichier TP_DSP.tcf du dossier « DSP/BIOSConfig », puis dans la fenêtre de droite qui va s'ouvrir, bouton droit « propriétés » sur « MEM », voir aussi le contenu du fichier « TP_DSP.cmd » :

Rappeler le nombre de mémoire disponible, leur type et leur taille
Donnez la taille et emplacement de la pile, du tas
Pendant combien de temps va durer la boucle acquistion-affichage ?

2. Première transformation YUV en niveaux de gris

Dans cet exercice nous allons transformer l'image couleur en niveaux de gris. Pour cela nous allons procéder en 2 étapes :

Etape 1 - Copie d'une image

L'appel suivant alloue de la mémoire pour stocker une image qui sera remplie par le circuit d'acquisition :
FVID_alloc(ccdcHandle, &FBAddr);
avec PSP_VPSSSurfaceParams *FBAddr = NULL;
et *ccdcHandle* un handler associé au driver d'acquisition vidéo
La structure PSP_VPSSSurfaceParams possède un champ frameBufferPtr qui pointe sur l'image. L'image couleur est stockée au format YUV 422 dans ce tableau de type uint 16 (unsigned short) de taille 720*480 (taille de l'image

Il s'agit d'allouer une nouvelle structure d'image et d'y recopier intégralement l'image acquise avant de l'afficher :

- Allouez un pointeur vers une seconde structure PSP_VPSSSurfaceParams que vous appelez FBAddrImageOut
- Avec FVID_alloc allouez la mémoire pour que cette structure pointe vers la zone allouée sans oubliez de libérer la mémoire après la boucle
- Ecrire une boucle qui parcours les 720*480 élèments du tableau de FBaddr et les recopie dans FBaddrImageOut (pour cela déclarer un pointeur source et un pointeur destination qui serviront à recopier un à un les pixels de l'image de FBaddr vers FBaddrImageOut).
- Modifier la ligne qui fait l'affichage pour qu'elle affiche FBaddrImageOut (et non FBaddr)

Etape 2 - Transfromation en niveau de gris

Le convertisseur vidéo de la carte fournit une image codée au format YUV 422.
Dans ce format, Y est la composante de luminance, U et V les composantes de chrominance. L'information de luminance Y permet seule d'afficher une image en niveaux de gris. En effet, pour convertir une image couleur où chaque pixel est codée en RGB (composante de couleurs primaires) il faut utiliser les équations suivante :

Y= c1*R +c2*G+c3*B (Y entre 0 et 1) avec C1=0.299, C3=0.114 et C2=1-C1-C2=0.587
U= Umax * (B-Y) / (1-C3) (U entre –Umax et +Umax, Umax=0.436)
V= Vmax * (R-Y) / (1-C1) (V entre –Vmax et Vmax, Vmax = 0.615)

Pour transformer l'image couleur en image en niveau de gris il faut savoir que dans chaque case du tableau pointé par frameBufferPtr correspond à un pixel de l'image. Les 8 bits de poids fort contiennent la composante Y du pixel alors que les 8 bits de poids faible contiennent la composante U ou la composante V. En YUV 422 la composante Y est donc unique pour chaque pixel, mais U et V sont mis à jour une fois sur deux car l'œil est moins sensible aux variations de couleurs que de luminances.

Si la composante U et V sont remplacés par la valeur 0, l'image sera transformée en niveaux de gris. Attention, U et V étant codé sur 8 bits , il faut donc mettre 128 dans U et V pour obtenir 0.

Dans un premier temps, pour effectuer la conversion couleur → niveaux de gris, il faudra modifier la boucle du programme précédent de façon à modifier U et V lors de la copie de l'image stockée dans FBAddr vers FBaddrImageOut.

Une fois que cela fonctionne, faites 2 boucles : la première conserve l'image couleur pour la moitié supérieur, la seconde transforme la moitié inférieur de l'image en niveaux de gris.

3. Démarrage : préparation pour traitements d'images en niveaux de gris

Les traitements que l'on peut faire « à la volée » comme précédement sont limités. Il s'agit donc d'extraire la composante Y de l'image acquise et de la stocker dans un tableau d'octèts (unsigned char) sur lequel nous ferons ensuite des traitements.

Pour cela il faut :

- Allouez statiquement dans la mémoire externe (DDR) un tableau *imagein* de taille LARGEUR*HAUTEUR octets non signés (avec #define LARGEUR valant 300 pixels HAUTEUR 200 pixels) Allouez de même un tableau identique *imageout*,

- Pour forcer l'allocation de ce tableau dans une zone mémoire spécifique il faut utiliser la directive de compilation suivante juste avant l'allocation mémoire :
    - #pragma DATA_SECTION(nomVariable, ".NomSectionMémoire")
    - (Le nom de la section mémoire a été trouvée dans la question 1)

- Ecrire une boucle qui copie la composante Y de l'image dans le tableau *imagein*
- Ecrire une boucle qui copie *imagein* dans *imageout*
- Ecrire une boucle qui copie *imageout* dans FBaddrImageOut
- Affichez comme précédement l'image

Seuillage de l'image en niveaux de gris :

Remplacez la simple recopie de imagein dans imageOut par une fonction *Seuillage* ; tous pixel du tableau d'entrée dont la valeur est supérieure à un seuil vaudra 255 dans le tableau de sortie imageout qui sera affiché. Les autres pixels passent à zero.
Le prototype sera *void Seuillage(char unsigned *in, char unsigned *out, int Size_L, int Size_H, unsigned char seuil) ;*

4 Filtre Sobel

Ecrire une fonction MySobel. Pour cela il faut appliquer séparément le sobel horizontal et le sobel vertical, puis faire la somme des valeurs absolue (approximation moins couteuse que la racine des sommes des carrés) des 2 valeurs pour obtenir une image du module (attention aux débordements!) .Le prototype sera *void MySobel(char unsigned *in, char unsigned *out, int Size_L, int Size_H,) ;*

Mesurer la durée d'exécution du Sobel complet suivi d'un seuillage. Le résultat sera donné en « nombre de cycles » et en déduire une approximation du nombre de cycles par pixels. Attention : ne pas mesurer autre chose (par exemple copie de *FBaddr* dans *imagein* et copie de *imageout* dans *FBaddrImageOut*)

La mesure du nombre de cycle peut se faire avec C64P_getltime() :

```
ts = C64P_getltime();
        PARTIE DU PROGRAMME à CHRONOMETRER
te = C64P_getltime();
LOG_printf( &trace, "    Nbcycles = %u", te - ts );
```

La mesure effectuée correspond à un Sobel appliqué à des tableaux stockés en mémoire externe (puisque imagein et imageout étaient dans la DDR). Il s'agit de faire la même mesure mais en appliquant le sobel sur une image stockée en mémoire interne.
Faire attention aux tailles des mémoires internes disponibles…

Récapituler les résultats sous la forme d'un tableau, en donnant les valeurs mesurées, les tailles images et le gain en pourcentages.

Modifiez les options de compilations et refaites toutes les mesures (mémoires internes et mémoires externes)

4 Filtre Sobel optimisé

Utilisez la fonction IMG_sobel de la bibliothèque IMAGLIB de Texas Instrument : c'est une fonction codée et optimisée en assembleur. Pour cela il faut lire le fichier C:\CCStudio_v3.3\c6400\imglib\README64.XT dont voici un extrait :

Using the Imaging Library in Code Composer Studio

1. First you have to specify to the linker to link the Imaging Library into your application. You do this by going to "Project->Options->Linker->Include Libraries" and then specify the Imaging Library, img64x.lib.

2. For each of your C files that you plan to use an Imaging Library kernel in, you must include the appropriate header file for each kernel. For instance, to use the kernel "IMG_fdct_8x8()", you must

#include <img_fdct_8x8.h> in that file.

C'est à dire : dans build option : linker → libraries →
path : C:\CCStudio_v3.3\c6400\imglib\lib;"
Et include libraries ajouter : "img64x.lib"

Refaites les mesures comme en 5) en commençant par un sobel de grande taille pour vérifier qualitativement les résultats.

Faire un nouveau tableau récapitulatif incluant les résultats de 5) et ceux de cette partie.

- ***Transferts DMA :***

Ouvrez le projet TP_DSP_DMA

Ce projet utilise l'EDMA (dans la fonction transfert_DMA) pour transférer les pixels du tableau imagein dans le tableau imageout.

1)Compilez et executez. Etudiez la fonction "transfert_DMA" et notamment les paramètres des fonctions dont le nom commence par "EDMA3_DRV_" c'est grâce à ces fonctions que l'on paramètre le transfert DMA.

2) Mesurer la durée (en nombre de cycles) du transfert d'une image complète.

3) Mettre en commentaires l'appel à la fonction transfert_DMA et remplacer par une fonction transfert_CPU qui réalisera le transfert à l'aide d'une simple boucle for.

Calculer le gain qu'apporte le transfert par DMA comparé au transfert par CPU.
Notez bien qu'en plus du gain de temps, on libère le CPU qui peut être utilisé pour du calcul.

4) Modifier l'exemple précédent de façon à ce que le transfert DMA réalise une inversion Haut <-> Bas de l'image : il faut pour cela changer l'adresse destination et la valeur de l'incrément (qui peut être négatif).

5) Donnez les paramètres DMA pour qu'il diminue par 2 la taille horizontale et verticale de l'image (saut de 2 en 2 dans les pixels sources)

6) Vous avez mesuré dans le TP précédent que le sobel était beaucoup plus rapide quand les données étaient stockées en mémoire interne mais que celle-ci ne permettaient pas de stocker l'image complète : proposer une solution qui utilise le DMA pour transférer les données entre mémoire externe et mémoire interne pendant que le CPU effectue le Sobel sur une autre partie de l'image stockée en mémoire interne.