PROJET:

Traitement d'images par Convolution sur plateforme DSP

I. Convolution:

La convolution est une fonction très utiliser en traitement d'image, parmi ces applications (filtrage, détection de contour, effet 3D...)voici quelques exemples d'opérateurs.

-1/8	-1/8	-1/8
-1/8	1	-1/8
-1/8	-1/8	-1/8

détection de contour

0	0	0
0	1	0
0	0	-1

effet 3D

-k/8	-k/8	-k/8
-k/8	k	-k/8
-k/8	-k/8	-k/8

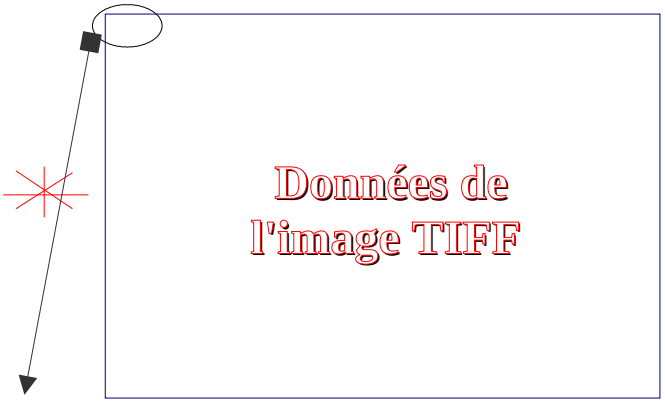
ajustement de contraste

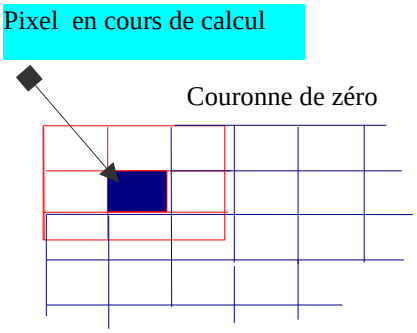
formule mathématique:

$$Pout(i,j) = \sum_{k=0}^{3} \sum_{l=0}^{3} (H(k,l) . Pin[(i-k),(j-l)])$$

Masque de convolution

-1/8	-1/8	-1/8
-1/8	1	-1/8
-1/8	-1/8	-1/8

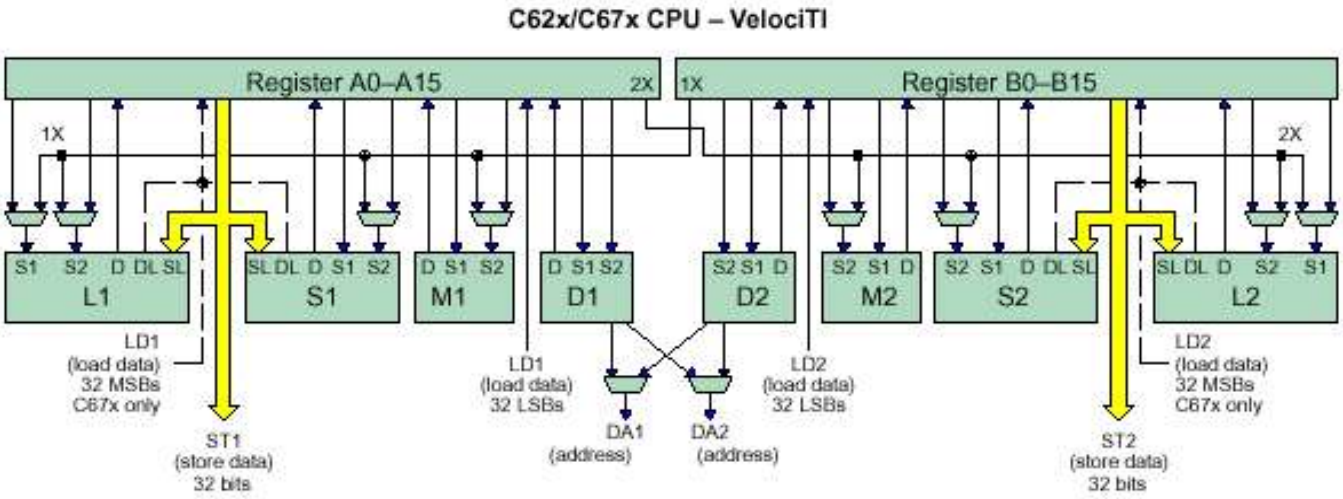




...

II.Architecture du C67:

Le DSP C67 fait partie de la dernière famille de DSP de Texas Instruments, apparue en 1998, ce DSP possède huit unités de calcul et 32 registres. Il fonctionne à des fréquences allant de 200 à 300 Mhz, ce qui lui donne une puissance crête de 1600 à 2400 MIPS, soit plusieurs fois celle d'un processeur RISC actuel.



Sa puissance crête n'est atteignable que lorsque toutes les unités de calculs fonctionnent. Notre but sera donc de maximiser leur utilisation, afin de minimiser les temps de calcul. Cependant il existe des contraintes de programmation:

-deux multiplications par cycle (un par côté),

-deux accès mémoire par cycle (un par côté),

-huit instructions par cycle (quatre par côté),

-six opérations par cycle, autre que multiplication (trois par côté) telle que addition, décalage, valeur absolue,

-zéro à deux croisements des chemins de donnée (cross-path).

Ces contraintes sont liées au fait que les unités M (Multiplication) et D (Data) sont spécialisées et ne peuvent exécuter qu'un seul type d'instruction. A côté de cela, nous pouvons voir qu'il est possible de réaliser jusqu'à six additions/soustractions par cycle.

III.STRUCTURE DU CODE ASSEMBLEUR:

Le programme assembleur doit être un fichier texte en ASCII, chaque ligne peut être composer de 7 élément.

-Label
-Parallèle bars
-Conditions
-Instruction
-Functionnal unit
-Operands
-comment

Labels:

Permet d'identifier une ligne du code.
Le label doit respecter les conditions suivantes:

-le premier caractère doit être une lettre ou un underscore.
-le premier caractère doit être dans la première colonne du texte.
-le label peut contenir jusqu'à 32 caractères alphanumériques.

Parallèle bars:

Les bars parallèle indique que l'instruction est exécutée en parallèle avec celle qui la precede.

Condition:

-Cinq registres du C67xx sont valable pour les conditions A1, A2, B0, B1 et B2.

*si la condition n'est pas spécifiée, l'instruction est toujours exécutée.

*si la condition existe et que celle ci est vrai, l'instruction est exécutée.

exemple:

avec cette condition	l'instruction est exécutée si
[A1]	A1 != 0
[!A1]	A1 = 0

-si la condition existe et que celle ci est fausse, l'instruction n'est exécutée.

exemple:

avec cette condition	l'instruction est exécutée si
[A1]	A1 = 0
[!A1]	A1 != 0

Instructions:

label:	parallel bars	[condition]	**instruction**	unit	operands	; comments

Les instructions assembleurs peuvent être des directives ou des mnémoniques:

-Directives: les directives assembleur sont des commande pour l'assembleur(asm6x), ces directives contrôlent le processus d'assemblage ou défini une structure de données (constantes et variables).

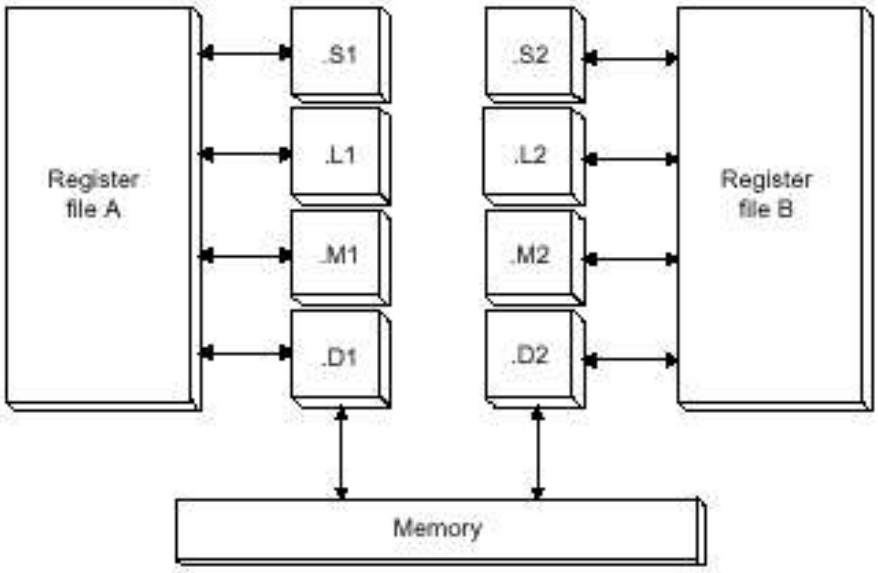
Directives	Description
.sect "*name*"	Creates section of information (data or code)
.double *value*	Reserve two consecutive 32 bits (64 bits) in memory and fill with double-precision (64-bit) IEEE floating-point representation of specified value
.float *value*	Reserve 32 bits in memory and fill with single-precision (32-bit) IEEE floating-point representation of specified value
.int *value* .long *value* .word *value*	Reserve 32 bits in memory and fill with specified value
.short *value* .half *value*	Reserve 16 bits in memory and fill with specified value
.byte *value*	Reserve 8 bits in memory and fill with specified value

-Mnémoniques:sont les instructions qui peuvent être exécuter sur le processeur,il doivent être placer à partir de la deuxième colonne.

Functionnal units:

label:	parallel bars	[condition]	instruction	unit	operands	; comments

Le C6000 contient 8 unités d'exécution comme décrit sur la figure suivante:



Functional Unit	Fixed–Point Operations	Floating–Point Operations
.L unit (.L1, .L2)	32/40-bit arithmetic and compare operations 32-bit logical operations Leftmost 1 or 0 counting for 32 bits Normalization count for 32 and 40 bits Byte shifts Data packing/unpacking 5-bit constant generation Dual 16-bit arithmetic operations Quad 8-bit arithmetic operations Dual 16-bit min/max operations Quad 8-bit min/max operations	Arithmetic operations DP → SP, INT → DP, INT → SP conversion operations
.S unit (.S1, .S2)	32-bit arithmetic operations 32/40-bit shifts and 32-bit bit-field operations 32-bit logical operations Branches Constant generation Register transfers to/from control register file (.S2 only) Byte shifts Data packing/unpacking Dual 16-bit compare operations Quad 8-bit compare operations Dual 16-bit shift operations Dual 16-bit saturated arithmetic operations Quad 8-bit saturated arithmetic operations	Compare Reciprocal and reciprocal square-root operations Absolute value operations SP → DP conversion operations

Functional Unit	Fixed–Point Operations	Floating–Point Operations
.M units (.M1, .M2)	16 x 16 multiply operations 16 x 32 multiply operations Quad 8 x 8 multiply operations Dual 16 x 16 multiply operations Dual 16 x 16 multiply with add/subtract operations Quad 8 x 8 multiply with add operation Bit expansion Bit interleaving/de-interleaving Variable shift operations Rotation Galois Field Multiply	32 x 32 bit fixed–point multiply operations Floating–point multiply operations
.D units (.D1, .D2)	32–bit add, subtract, linear and circular address calculation Loads and stores with 5–bit constant offset Loads and stores with 15–bit constant offset (.D2 only) Dual 16–bit arithmetic operations Load and store double words with 5–bit constant Load and store non–aligned words and double words 5–bit constant generation 32–bit logical operations	Load doublewords with 5–bit constant offset

Operands:

label:	parallel bars	[condition]	instruction	unit	**operands**	; comments

*Toutes les instructions on besoin d'opérande destination.
*Chaque instruction nécessite une ou deux opérandes source.
*L'opérande peut être la même que la source.
*Si l'opérande destination est un registre du chemin A, l'opérande source peut être un registre du chemin B, dans ce cas le champs unités s'écrit (.L1X).

```
ADD   .L1     A0, A1, A3
ADD   .L1X    A0, B1, A3
```

(tous les registre sauf B1 sont du même coté du CPU)

-Les instructions du C6000 utilise trois types d'opérandes pour accéder à la donnée:
*registre contenant une donnée.
*constante.
*pointeur contenant l'adresse de la donnée.

Seul le load et le store requièrent et utilise un pointeur pour charger des données dans la mémoire ou dans un registre.

Comments:

*Une ligne est mise en commentaire si elle est précédée de (;).
*Le commentaires n'est pas obligatoire mais recommander.

IV.PIPELINE TMS320C67X:

1. Pipeline:

Le pipeline est constituer de trois phases:
-Fetch
-Décode
-Execute

1.1Fetch:

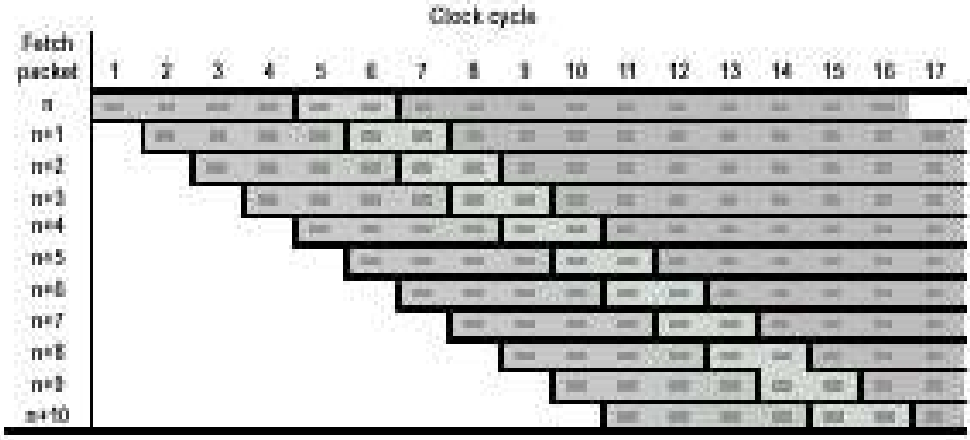
Le fetch instruction prend 4 cycles:
-PG : génération d'adresse.
-PS : envoi d'adresse.
-PW : accès à la lecture mémoire.
-PR : réception du paquet d'instruction.

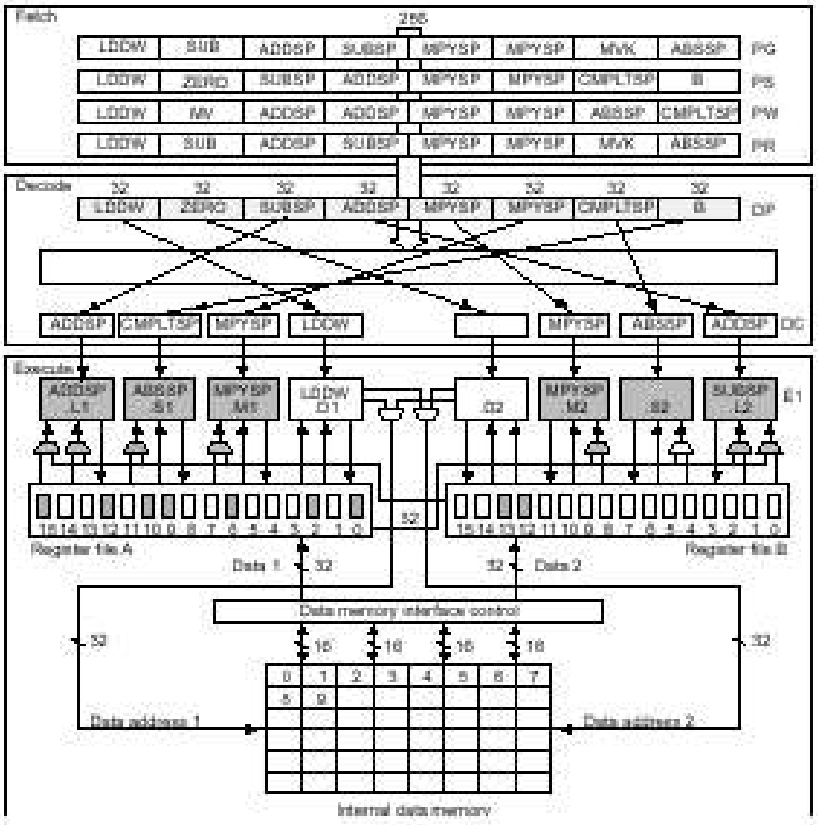
1.2Décode:

Les phases de décodage du pipeline sont:
-DP : attribution des instructions au unités fonctionnelles approprié.
-DC : Décodage d'instructions.

1.3Exécute:

Le nombre de cycles d'exécution varie en fonction de l'instruction, et peut allez jusqu'à 10 cycles pour certaine instructions pour flottant.





exemple d'exécution

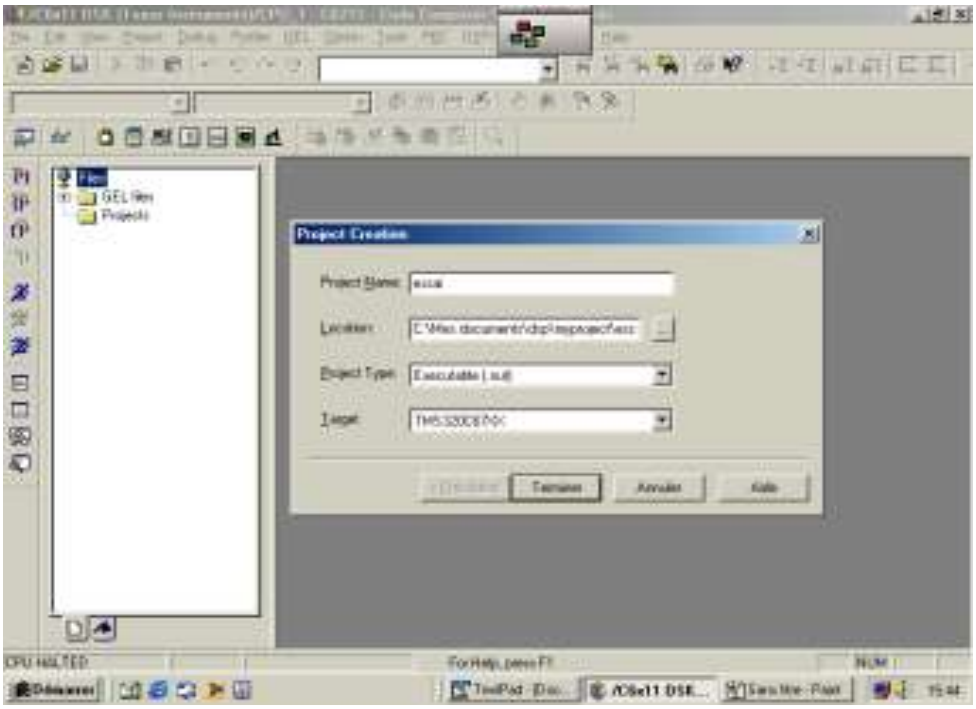
V.Introduction à l'usage de CCS(Code Composer Studio):

CCS est un environnement de développement pour les DSP de Texas Instrument qui fournit plusieurs outils pour faciliter la construction et la mise au point des programmes.
En outre, il permet l'analyse en temps réel de l'exécution des programmes de DSP. CCS inclut un éditeur de code source en (C et assembleur), un compilateur, un assembleur et un éditeur de liens, et un debogueur. Il fournit aussi un environnement de gestion de fichiers qui facilite la construction et la mise au point des programmes.

L'exemple qui suit montre comment un algorithme multifilaire simple peut être compilé,assemblé et lié en utilisant CCS. D'abord, plusieurs valeurs de données sont écrites à la memoire. Ensuite un pointeur est assigné au début des données de sorte qu'elles puissent être traitées comme elles sont présentées . Finalement des fonctions simples sont ajoutées en C et en assembleur pour illustrer la façon dont celles-ci sont exécutées .

1. CRÉER UN PROJET

Le processus de développement du code de CCS commence par la création d'un fichier projet(extension .pjt) qui facilite l'édition et l'intégration des fichiers requis pour produire un fichier exécutable . Le fichier projet contient des références aux fichiers source(.c , .asm), aux fichiers d'en-tète (.h), au fichier de commande pour l'éditeur de liens (.cmd),et aux fichiers de bibliothèque (.lib). Le fichier-projet permet aussi de spécifier les paramètres du compilateur, de l'assembleur et de l'éditeur de liens afin de produire le fichier exécutable .
Pour créer un projet, vous choisissez l'élément de menu « Project->New » de la barre d'outils du CCS. Cela fait apparaître la fenêtre de dialogue « Save New Project As » comme illustré ci-dessous.



Dans celle-ci, vous introduisez un nom de projet (essai, par exemple) dans le domaine « Filename». Ensuite, vous cliquez sur « Save » pour que le CCS crée un fichier-projet nommé essai.pjt Les fichiers nécessaires pour établir une application devront être ajoutés au projet par la suite.

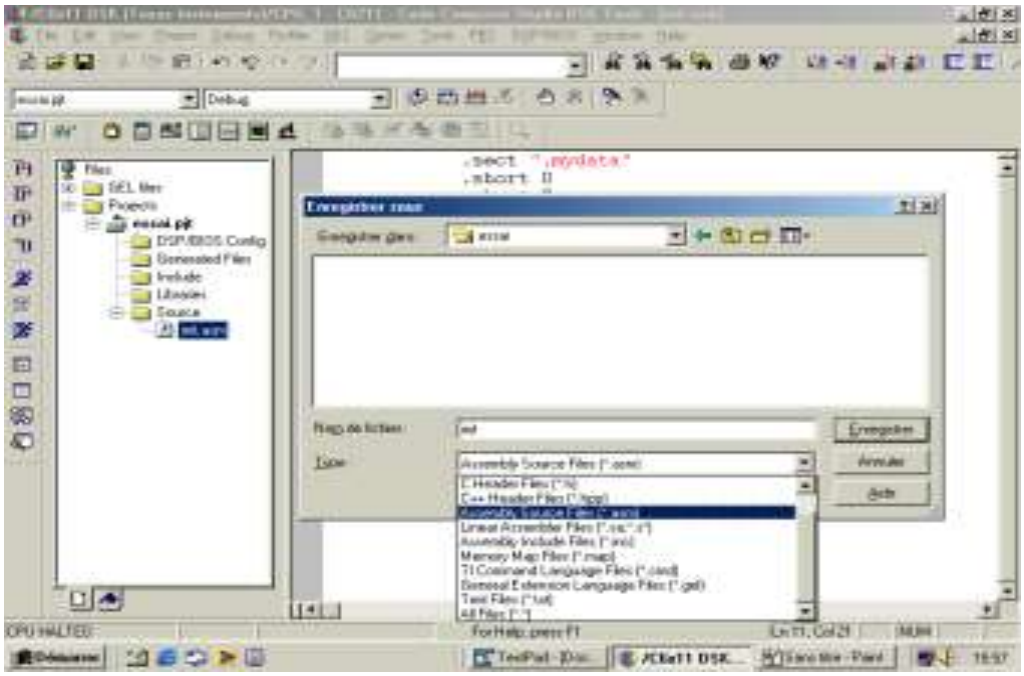
CCS fournit un éditeur intégré qui permet la création des fichiers source. Une fenêtre d'éditeur apparaît vers le haut en choisissant l'élément de menu « File -> New ->Source » Pour notre exemple, écrivons le code suivant dans cette fenêtre d'éditeur :

```
.sect  ".mydata"
.short  0
.short  7
.short  10
.short  7
.short  0
.short  -7
.short  -10
.short  -7
.short  0
.short  7
```

Ce code déclare 10 valeurs numériques en utilisant la directive .short. La directive .sect spécifie que les 10 valeurs doivent résider dans une section de la mémoire du DSP appelée .mydata : les adresses

physiques qui correspondent à cette section seront définies plus tard dans un fichier .cmd.

Vous sauvegardez le fichier source créé en choisissant l'élément de menu « File -> Save » ce qui fait apparaître la fenêtre de dialogue « Save as » comme illustré ci-dessous. Dans cette fenêtre allez à la zone « Save as type » et choisissez « Assembly Source Files (*.asm)» à partir de la liste déroulante. Ensuite ,allez à la zone « Filename » et écrivez le nom du programme: initmem.asm pour l'exemple. Finalement, en cliquant sur « Save », le code est sauvegardé dans le fichier source assembleur spécifié.



En plus du fichier source, un fichier de commande d'éditeur de liens doit être indiqué pour créer un fichier exécutable et pour se conformer aux spécifications de mémoire du DSP et de la cible sur laquelle le fichier exécutable va être compilé. Un fichier de commande d'éditeur de liens peut être créé en utilisant la même procédure que pour le fichier précèdent(en choisissant « File -> New ->Source ») Pour l'exemple en cours, nous allons considérer le fichier de commande suivant :

```
MEMORY
{
  SDRAM      : origin = 0x80000000,  len = 0x100000
  DATA       : origin = 0x80700000,  len = 0x300000
}
```

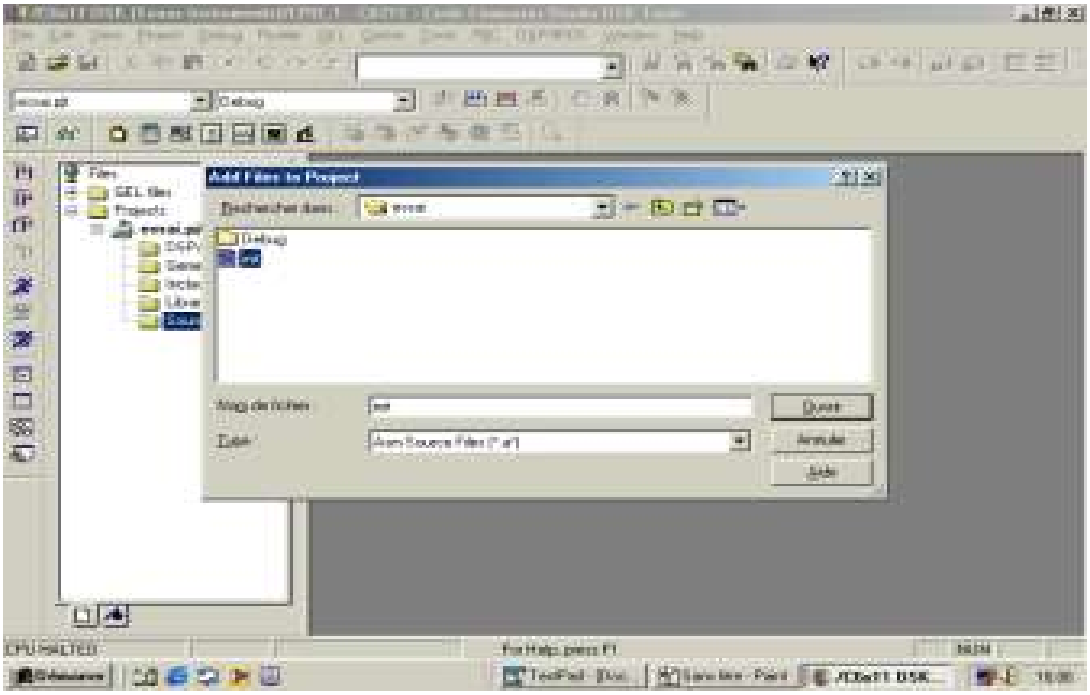
```
SECTIONS
{
      .vectors > SDRAM
      .text    > SDRAM
      .bss     > SDRAM
      .cinit   > SDRAM
      .mydata > DATA
      .const   > SDRAM
      .far     > SDRAM
      .stack   > SDRAM
      .cio     > SDRAM
      .sysmem  > SDRAM
}
```

Puisque notre intention est de placer les valeurs définies dans le fichier initmem.asm dans la mémoire, un espace qui ne sera pas recouvert par le compilateur devrait être choisi.L'espace de donnés extérieure DATA peuvent être utiliser à cette fin. Commençons par assembler les données à l'adresse de mémoire 0x80700000, situé au début de DATA.

Pour faire ceci, assignez la section nommée .mydata à DATA en ajoutant « mydata ->DATA » dans la partie SECTIONS du fichier de commande, comme déjà indiqué dans le programme ci-haut. La fenêtre d'édition devrait être sauvegarder dans un fichier de commande d'éditeur de liens en (essai.cmd).

1-Ajout d'un fichier au projet

Une fois que le fichier source initmem.asm et le fichier de commande d'éditeur de liens essai.cmd sont créés, ils devraient être ajoutés au projet. Pour ce faire, choisissez l'élément de menu « Projet-> Add Files to Project » Cela évoque la fenêtre de dialogue « Add Files to Project» . Sélectionner les deux fichiers et cliquer sur « open ».



En plus des fichiers initasm.asm et essai.cmd, le fichier d'exécution de bibliothèque de support devrait être ajouté au projet (ti/c6000/cgtools/lib/rts6700.lib).

2-La mise au point du projet

Pour la mise au point, utilisons un programme interpréteur de commandes interactif vide. Créez un fichier source en c qu'on appellera main.c, écrivez les lignes suivantes et ajoutez main.c au projet de la même manière que juste décrit.

main.c:

```
#include        <stdio.h>

void   main()
{       printf("BEGIN\n");
        printf("END\n"); }
```

Apres avoir ajouté tous les fichiers source, fichier de commande et fichier de bibliothèque au projet,vous pouvez construire le projet et créer un fichier exécutable pour le DSP-cible. Pour faire ceci, choisissez l'élément de menu « Project->Build ». Cette fonction permet au CCS de compiler, assembler, et joindre tous les fichiers dans le projet.

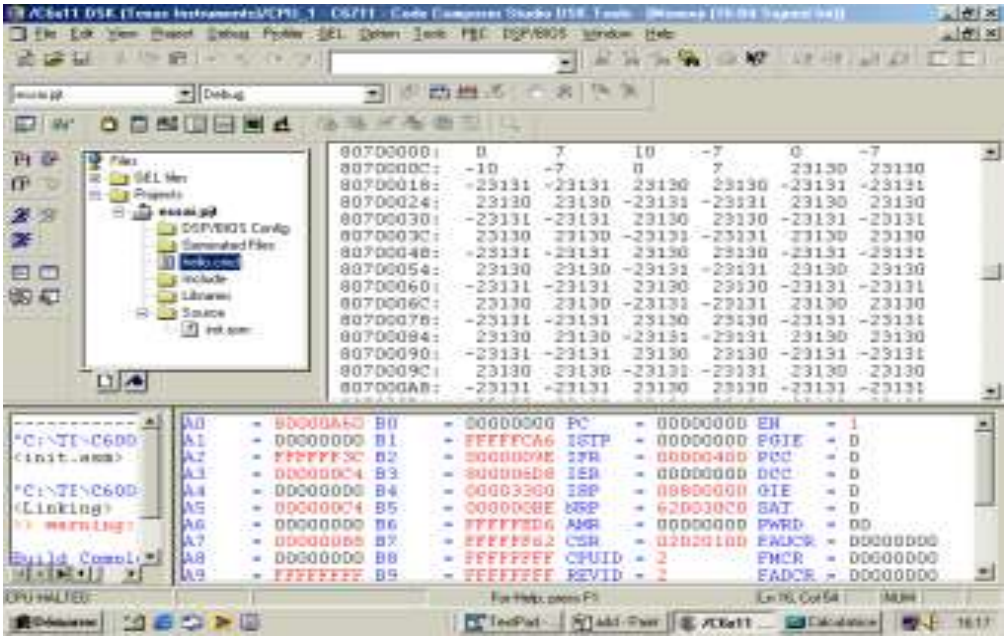
Si le processus de construction est complété sans erreur, le fichier exécutable essai.out est produit. Il est également possible de compléter les constructions par incréments ; c'est-à-dire en recompilant ou en assemblant seulement des fichiers changés depuis la dernière construction , en

choisissant l'élément de menu « Project->Rebuild ».

2. OUTILS DE SIMULATION

Si le processus de construction est complété sans aucune erreur, le programme peut être chargé et exécuté sur le DSP-cible. Pour charger le programme de l'exemple choisissez «File->Load Program» sélectionnez le fichier essai.out et cliquez sur « Open ».
Pour exécuter le programme, choisissez l'élément « Debug->Run » du menu. Vous devriez voir "'BEGIN" et "END" apparaître dans le " Stdout window"(La fenêtre en bas de l'écran).

Maintenant allons vérifier si l'ensemble des valeurs est assemblé dans l'emplacement de mémoire indiqué. CCS permet de visualiser le contenu de la mémoire à un emplacement spécifique. Pour visualiser le contenu de la mémoire à l'adresse 0x80800000. Sélectionnez alors, «16bit Signed Int» de la liste déroulante dans « Format field » comme illustré ci-dessous et cliquez ok.



le contenu de l'unité centrale de traitement,du contrôleur d'accès direct en mémoire peut aussi être visualisé,en sélectionnant « View CPU ».

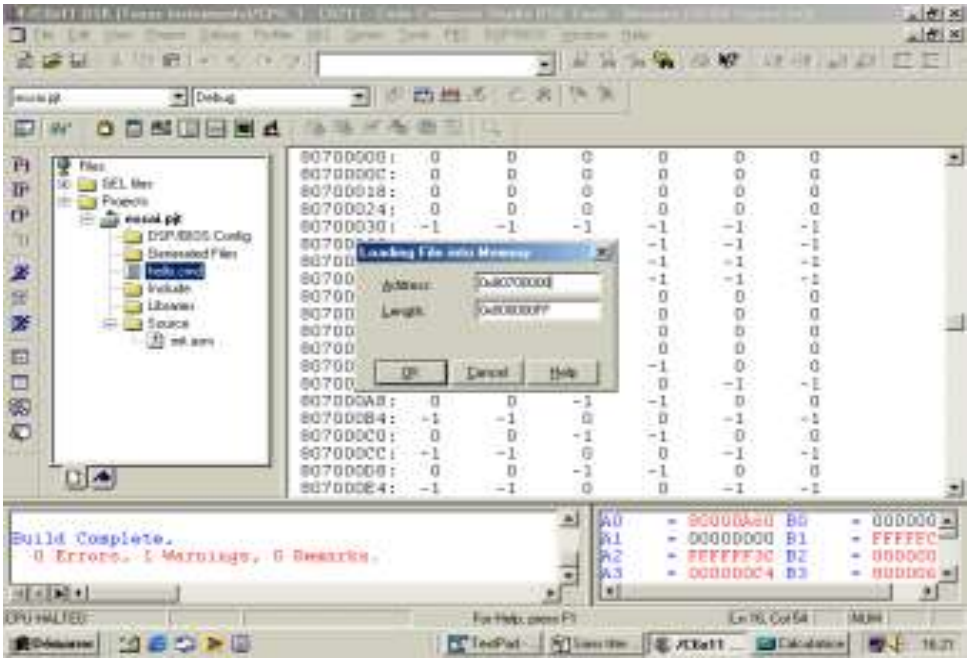
1-Chargement de données dans le DSP:

Le CCS offre la possibilité de charger des données d'un fichier sur l'ordinateur dans la mémoire du DSP, et vis-versa . Le fichier de données (données.dat) est présenté comme suit:

1656 1 0 0 0
0x0000fa05
0x00000000

0x000000ab
.....
...

choisissez « File -> Data -> Load » ,enter l'adresse mémoire,la longer des données à transférer puis OK.



3-Pointeur

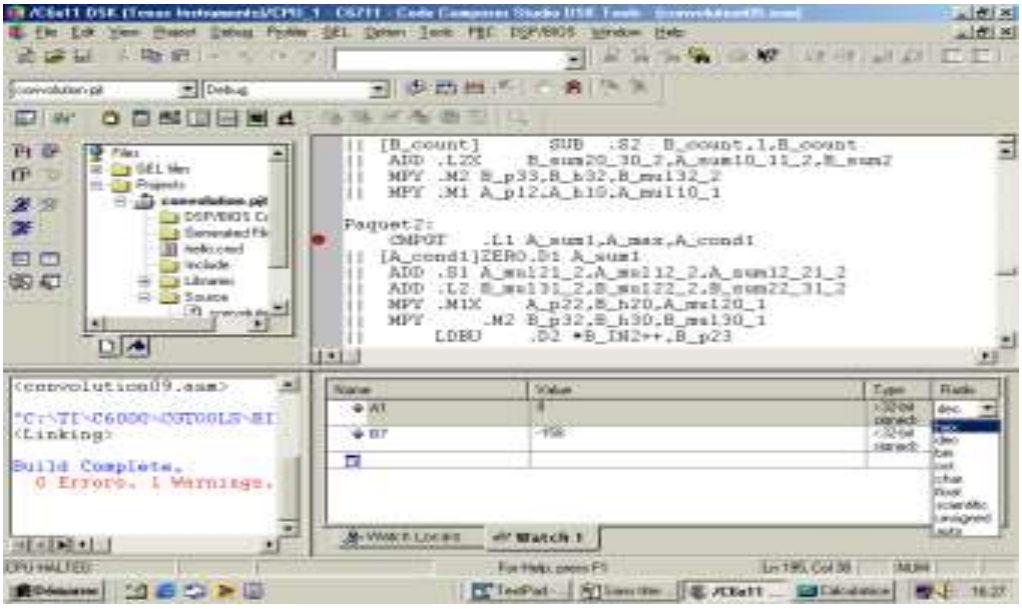
Un pointeur peut être utilisé pour mémoriser l'emplacement d'un bloc de données en mémoire. Par exemple,le code suivant peut être employé pour assigner un pointeur au début de la chaîne de valeurs utilisées dans l'exemple et parcourir celles-ci en affichant chacune dans la fenêtre Stdout:

```
#inclue <stdio.h>

void main()
{
        int i;

        short *point;

        point = (short *) 0x80800000;

        printf("BEGIN\n");
```

```
	for(i=0 ; i<10;i++)
	{
		printf("[%d] %d\n",i,point[i]);
	}

	printf("END\n");
}
```

4- Point d'arrêt:

Durant le développement ou le test des programmes, on doit souvent contrôler la valeur d'une variable pendant l'exécution du programme. Ceci peut être réalisé en utilisant des points d'arrêts (Breakpoint) et des fenêtres de surveillance (Watch Window),ainsi pour mettre un point d'arrêt double cliqué au début de la ligne ou vous voulez insérer le point d'arrêt .
Pour visualiser des variables (Watch Window), choisissez (View -> Watch Window) à partir de la barre de menu. Pour ajouter une nouvelle expression dans cette fenêtre, appuyez sur le bouton droit de la souris par-dessus cette fenêtre et sélectionnez (Insert New Expression). Dans le domaine d'expression,tapez l'expression que vous voulez examiner et cliquez sur OK . Lors du lancement du programme, celui-ci marque un point d'arrêt et les expressions dans le watch windows vont être rafraîchit.



5- Mesure de performance

Un élément important du procédé de mise au point est le chronométrage de l'exécution du programme. Rechargez le programme et choisissez (Profiler -> Enable Clock),choisissez (Profiler) pour faire apparaître une fenêtre ou sera afficher à la fin d'exécution du programme les durées de cycle de chaque fonction.

VI.Conversion de format

Dans notre projet on a choisi d'utiliser des images au format TIFF (Tag Image File Format) en niveaux de gris codé sur 8bit (0->255)(ASCII).

Parmi les Tags que l'on retrouve trois sont toujours présents : ***ImageWidth*** (largeur) , ***ImageLength*** (longueur) et ***BitsPerSample*** (nombre de bits par pixel) et indispensable .

L'environnement de développement offre la possibilité de transférer des données d'un fichier sur l'ordinateur vers la mémoire du DSP. Le format de ce fichier est le format DATA (.dat) ou les données sont codée en (HEX) et qui se présente comme suit:

8565 1 0 0 0
0x00FF8800
0x556687FF
.....

nous avons écrit deux fonction en c:

-tif2dat : conversion ASCII -> HEX => image.tif -> image.dat
-dat2tif : conversion HEX -> ASCII => image.dat -> imageout.tif

vous trouverez en annexe le code de ces deux fonctions.

Quelques résultats obtenues:

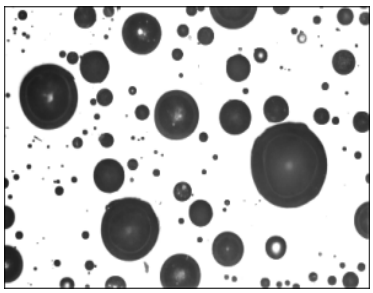
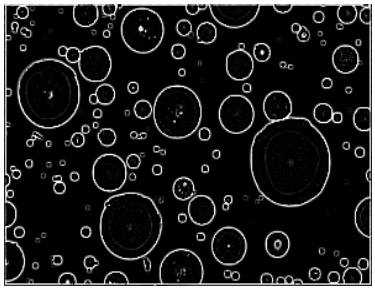
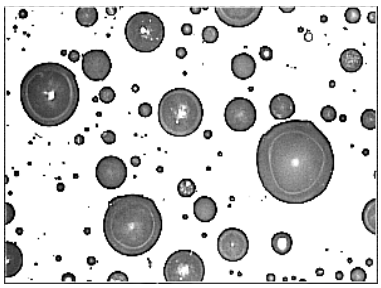


image d'origine



détection de contour



ajustement de contraste

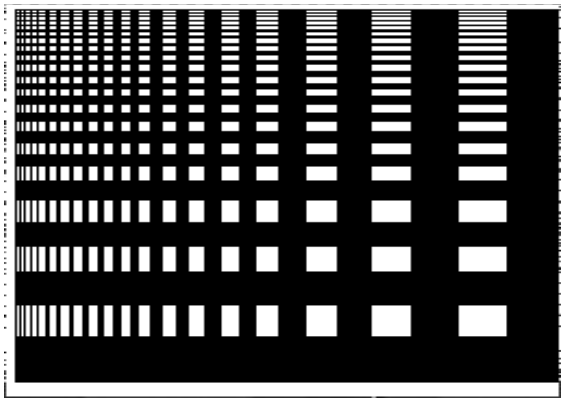
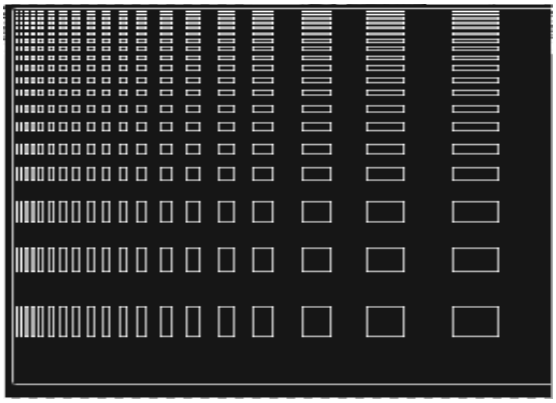


image d'origine



détection de contour

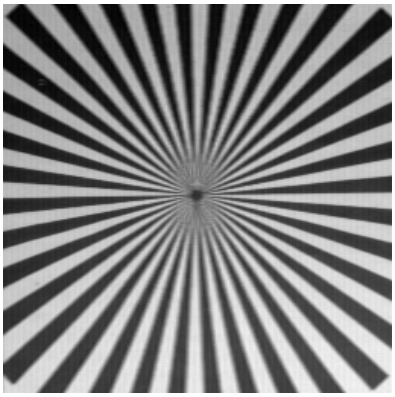
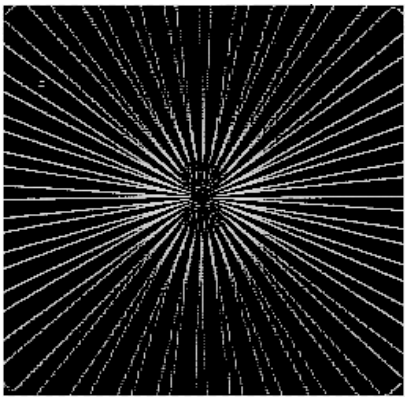


image d'origine



détéction de contour + ajustement de contraste

IX.CONCLUSION:

Le traitement d'image sur DSP s'est avéré très intéressant , dans notre cas, la convolution réalisée permet de traiter un pixel tout les 4.5 cycles et cela grâce à quelques optimisations: un simple déroulement de la boucle interne avec rotation de registre, suivi d'un repliement du graphe(pipeline logiciel).

Il est nécessaire de programmer le coeur des boucles en assembleur pour pouvoir exploiter le maximum de puissance du DSP, car les outils de compilations et d'assemblage actuels ne permettent pas de tirer parti de la puissance du DSP.

Les optimisations effectuer permettent de traiter des images 512x512 en temps réel , le temps d'exécution du coeur de boucle, en omettant la durée des transferts mémoire , est de 6ms, donc le TMS320C6711 est deux à trois fois plus rapide que les plus puissant processeurs RISC. un DSP VLIW peut être une alternative aux circuits FPGA et ASIC dans de nombreux cas.