Université BADJI Mokhtar Annaba

Faculté des sciences de l’ingénieur

Département d’électronique

Master 1 Système embarqué

Matière : DSP

**TP3 : Programmation du produit scalaire de 2 vecteurs**

Le but de ce 3eme TP est :

1. Utilisation des points d’arrêt (breakpoints)
2. Exécution pas à pas (single stepping).
3. Utilisation du “Profile Clock” pour l’estimation du temps d’exécution.

a- **Les opérations MAC**

Multiply/accumulate est une opération très importante dans le traitement numérique du signal.

C’est une partie fondamentale des algorithmes du filtrage numérique, de la corrélation, et de la transformée de Fourier rapide.

Il est donc important que l’opération MAC s’exécute en un seul cycle.

Le DSP C6713 processeur peut traiter 2 opérations MAC en un seul cycle.

**b- Le programme :**

Le fichier source dotp4.c calcule le produit scalaire de 2 vecteurs à valeurs entières.

Le premier vecteur est initialisé avec les valeurs 1, 2, 3, et 4, et le 2ème vecteur avec les valeurs 0, 2, 4, et 6.

Le produit scalaire est alors:

(1 × 0) + (2 × 2) + (3 × 4) + (4 × 6) = 40.

//**dotp4.c**

**// produit scalaire de 2 vecteurs**

#include <stdio.h>

#define count 4

int dotp(short \*a, short \*b, int ncount);

short x[count] = {1,2,3,4};

short y[count] = {0,2,4,6};

main()

{

int result = 0;

result = dotp(x, y, count);

printf("result = %d (decimal) \n", result);

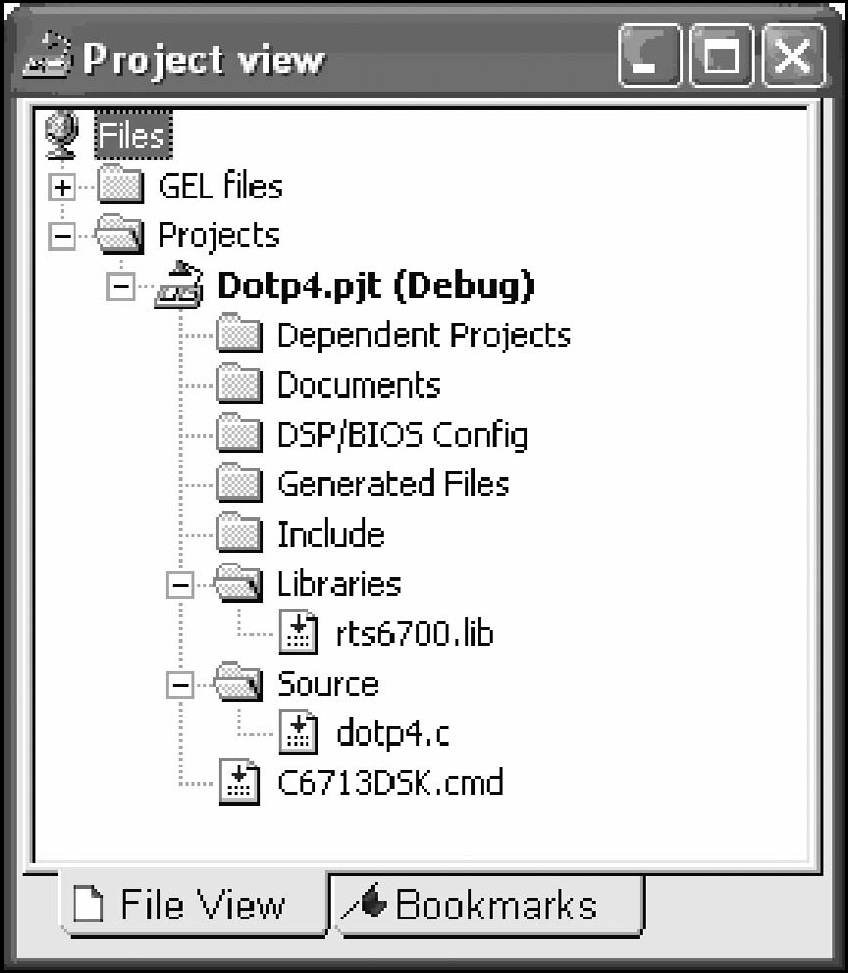
}

**c- Création du projet dotp4 qui doit inclure les fichiers suivants:**

**1. dotp4.c :** fichier source C.

**2. 6713dsk.cmd :** fichiercommand.

**3. rts6700.lib :** fichierlibrary.

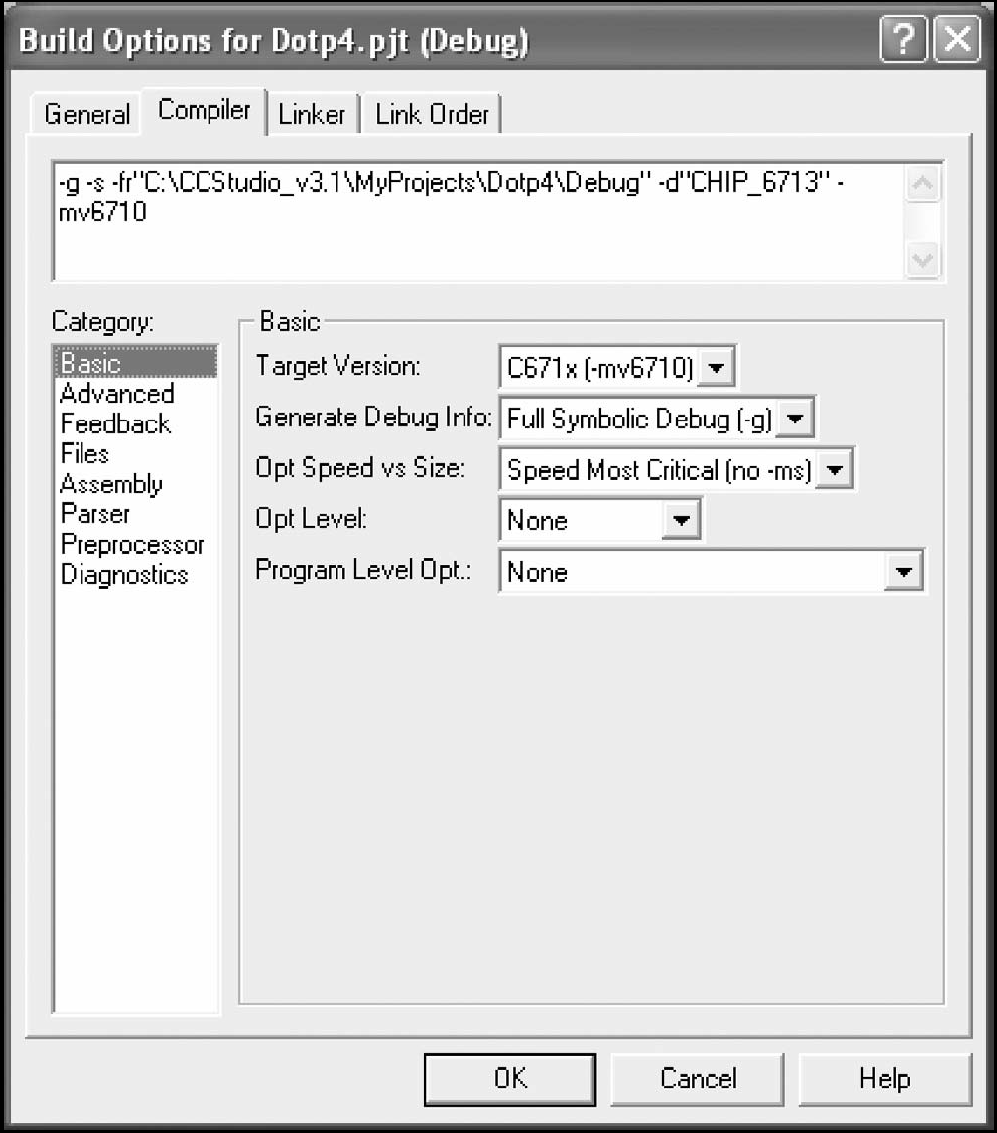


**d- Implémentation d’une variable de contrôle (watch variable)**

1. Sélectionner:

Project → Build Options

Et vérifier que la configuration basique du compilateur est telle qu’indiquée dans la figure suivante:



**2**. Build le projet en cliquant sur l’icone avec 3 flèches. (ou sélectionner Project → Build ). Charger le fichier exécutable **dotp4.out** .

**3.** Sélectionner **View → Quick Watch .** Taper **sum** (la variable sum à contrôler) **,** et cliquer sur **Add to Watch** . Le message **“ identifier not found: sum ”** doit apparaitre dans la fenêtre de contrôle (Watch window.) La variable sum est déclarée dans la fonction dotp(), c’est donc une variable locale et elle n’existe donc pas avant l’appel de la fonction.

**4.** Placer un breakpoint à la ligne

**sum += a[i] \* b[i];**

en cliquant sur la ligne du fichier dotp4.c et

soit clic droit et sélectionner **Toggle Software Breakpoint** , ou

en cliquant sur le bouton **Toggle Breakpoint** de la barre de menu.

Un **point rouge** doit alors apparaitre à gauche de la ligne en question.

**5.** Sélectionner **Debug→ Run.**

Le programme s’exécute et s’arrête juste avant la ligne ou figure le breakpoint.

Une flèche jaune apparaitra à gauche de cette ligne. A ce moment, la valeur 0 de la variable sum doit apparaitre dans la fenêtre "Watch window ".

Maintenant, la fonction sera exécutée la variable existe et sa valeur peut être affichée.

**6.** Continuer l’exécution du programme en sélectionnant  ***Debug→ Step Into ,*** *ou* **F11**. Continuer pas à pas et vérifier que la variable sum (dans la fenêtre "*Watch* window") change de valeur 0, 4, 16, et 40.

**7.** Lorsque la valeur de la variable **sum** a atteint **40,** sélectionner ***Debug→ Run*** *pour terminer l’*execution du programme, et vérifier que la valeur renvoyée par la fonction dotp() est affichée sous la forme: (l’affichage se fait dans stdout.

**result = 40 (decimal)**

A nouveau, le message “ identifier not found: sum ” apparaitra pour signaler que l’exécution de la fonction dotp() est terminée et que la variable sum n’existe plus.

**d- Animation**

1. Sélectionner  **File→ Reload Program**

pour recharger le fichier **dotp4.out (ou** sélectionner **Debug→ Restart** ).

Une fois chargée, le compteur de programmer est positionné à l’adresse identifiée par **c\_int00** . Ceci est vérifiable dans **Disassembly window**.

**2. Le précédent breakpoint est toujours à la même ligne de code.**

Sélectionner **Debug→ Animate** et observer les changements de la valeur de la variable sum dans "Watch window".

La vitesse de l’animation peut être contrôlée en sélectionnant **Option→ Customize → Animate** Speed (par default, la vitesse maximum est 0 seconds).

Estimation du temps d’exécution de la Fonction dotp() en utilisant "Profile Clock "

**1.** Ouvrir projet **dotp4.pjt .**

**2.** Sélectionner ***Project→ Build Options .***

Dans l’onglet ***Compiler , Basic category ,*** choisir ***none*** *pour* ***Opt Level****.*

**3.** Sélectionner ***Project→ Build*** *et puis* ***File→ Load Program* dotp4.out** .

**4.** Ouvrir **dotp4.c** et effacer le breakpoint précédent, puis , placer des breakpoints aux lignes:

**result = dotp(x, y, count);**

et

**printf(“result = %d (decimal) \n”, result);**

**5.** Sélectionner ***Profile → Clock → Enable .***

**6.** Sélectionner ***Profile → Clock View .***

*L’icone d’une petite horloge et le nombre de cycles d’instruction comptabilisé par Profile Clock apparait dans le coin bas droit de la fenêtre de* Code Composer.

**7.** Exécuter le programme qui doit s’arrêter au **1er breakpoint**.

**8. Reset le *Profile Clock*** *(double clic sur son icone)*

**9.** Exécuter le programme qui doit s’arrêter au **2ème breakpoint.**

Le nombre de cycles d’instruction comptabilisé par le *Profile Clock entre les 2* breakpoints, c.à.d. pendant l’exécution de la fonction dotp() , doit s’afficher à coté de l’icone.

**Questions:**

**Estimer le nombre de cycles et le temps nécessaires pour l’exécution de la fonction dotp().**

**Trouver le nombre de cycles et le temps requis par la fonction printf().**