
COMPTE RENDU DU T1

INITIATION AU CODE COMPOSER STUDIO V4

SOMMAIRE

I.	OBJECTIFS	2
II.	PRE-REQUIS	2
III.	CREATION D'UN PROJET	2
	1. Les étapes de création d'un projet	2
	2. Les étapes d'ajout des fichiers dans un projet	5
	3. Ajout du DSP/BIOS v5.x Configuration File	6
IV.	EXECUTER LE PROJET	12
	a. Ajouter les chemins des bibliothèques	12
	b. compiler le projet	13
V.	OPTIMISATION	16
	i. Ajouter les bibliothèques	16
	ii. Compiler le projet	17
VI.	IMAGE PROCESSING	18
	iii. Ajouter les chemins des bibliothèques	18
	iv. Modification du code source	18
	v. Compiler le projet et exécution	19
VII.	IMAGE PROCESSING AVEC OPTIMISATION	19
	vi. Changer en mode release	20
	vii. information assembleur	20
	viii. Exécution du code	20
	ix. Optimisation de structures itératives	21

I. OBJECTIFS

Dans ce Tp on vise trois objectifs :

- Maîtriser l'environnement l'outil de développement DSP de Code Composer Studio.
- Apprendre à programmer les DSPs de Texas Instruments en utilisant le langage C et langage assembleur.
- Etudier les techniques d'optimisation sur DSP.

II. PRE-REQUIS

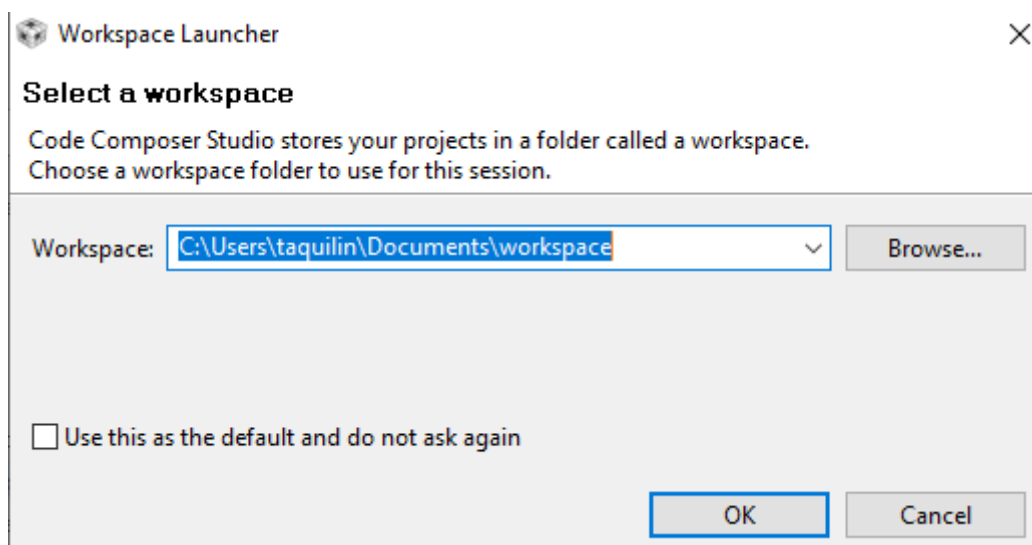
Pour ce Tp il faut avoir:

- Connaissance de l'architecture c64x.
- Maîtrise du langage C.

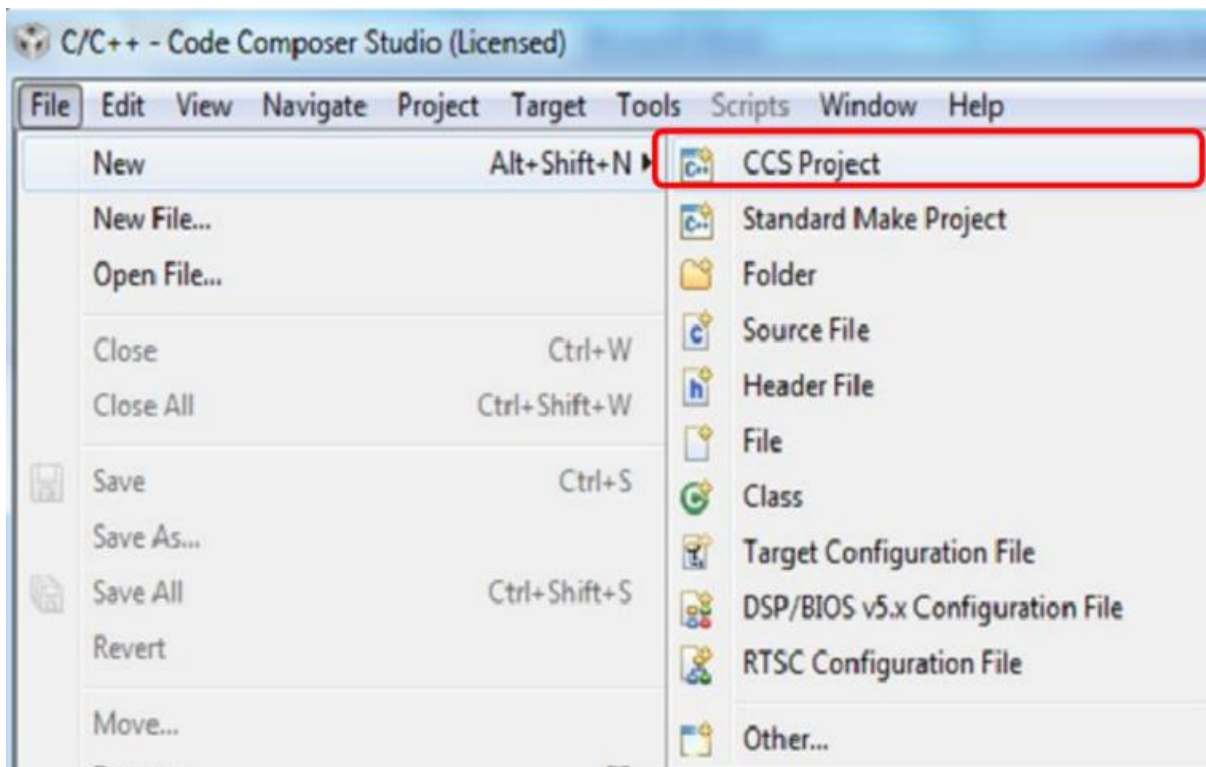
III. CREATION D'UN PROJET

1. Les étapes de création d'un projet

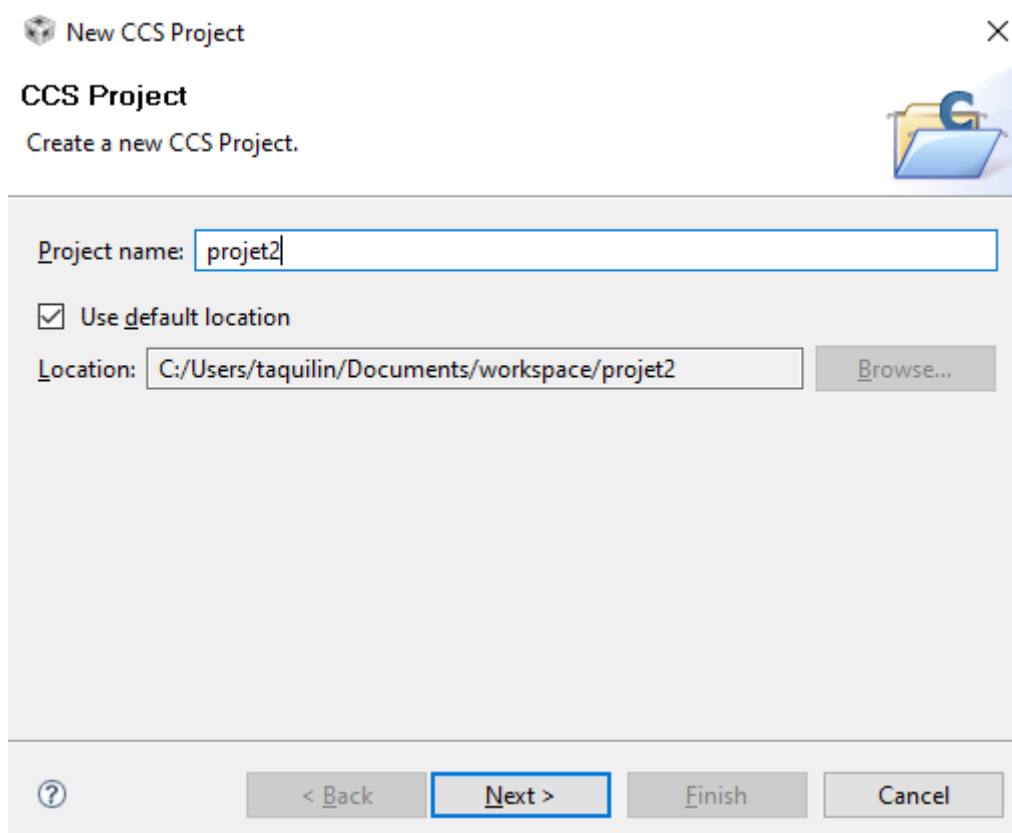
- Démarrer le logiciel Code Composer Studio v4
- Créer votre répertoire de travail (Workspace)



- Créer un nouveau projet par l'activation de File → New → CCSProject



- Attribuer un nom au projet par exemple "project2", vérifier son emplacement et par la suite cliquer sur Next.



- Choisir le type la famille de DSP à utiliser : (C6000).
- Activer les deux modes de compilation : Debug et Release.

New CCS Project

Select a type of project

Select the platform and configurations you wish to deploy on

Project Type: C6000

Configurations:

- ☒ Debug
- ☒ Release

Select All

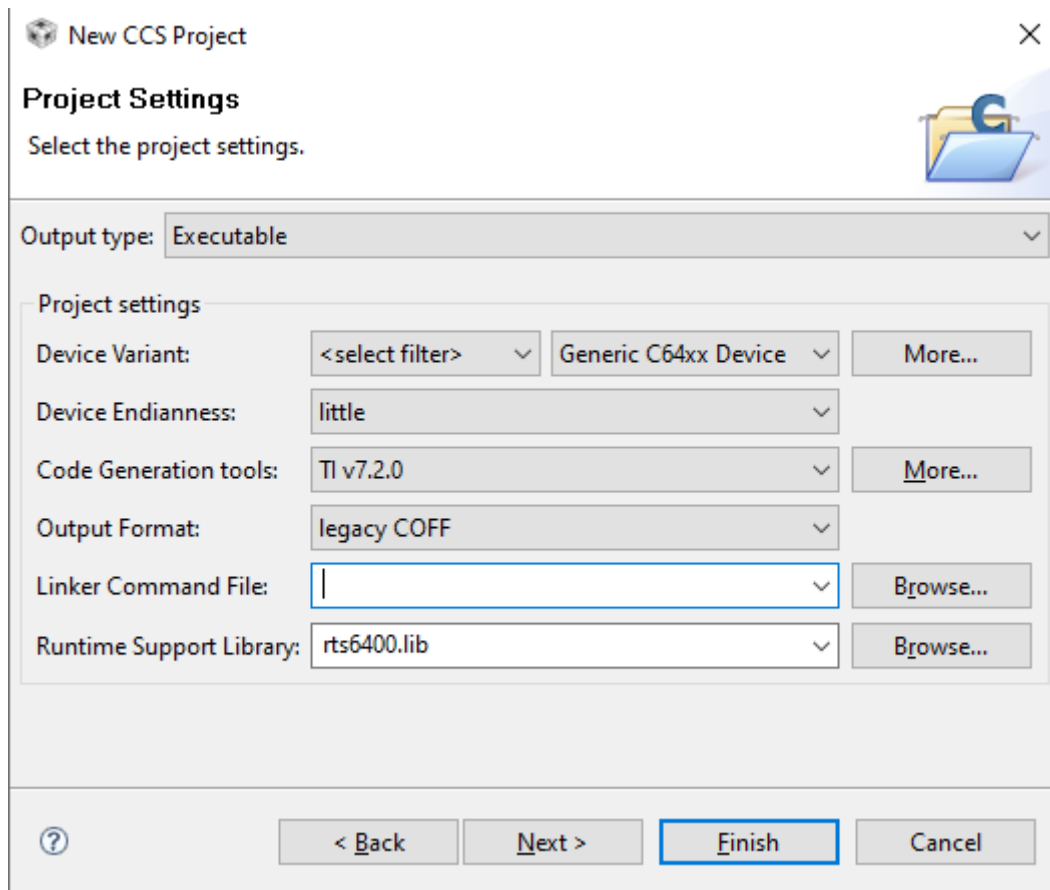
Deselect All

☐ Show All Project Types

☐ Show All Configurations

? < Back Next > Finish Cancel

- Préciser les différents paramètres du projet, à savoir :
 - Output type : Executable
 - Device Variant : Generic C64xx Device
 - Device Endianness : little
 - Run Time Support Library : rts6400.lib



Cliquer sur Next.

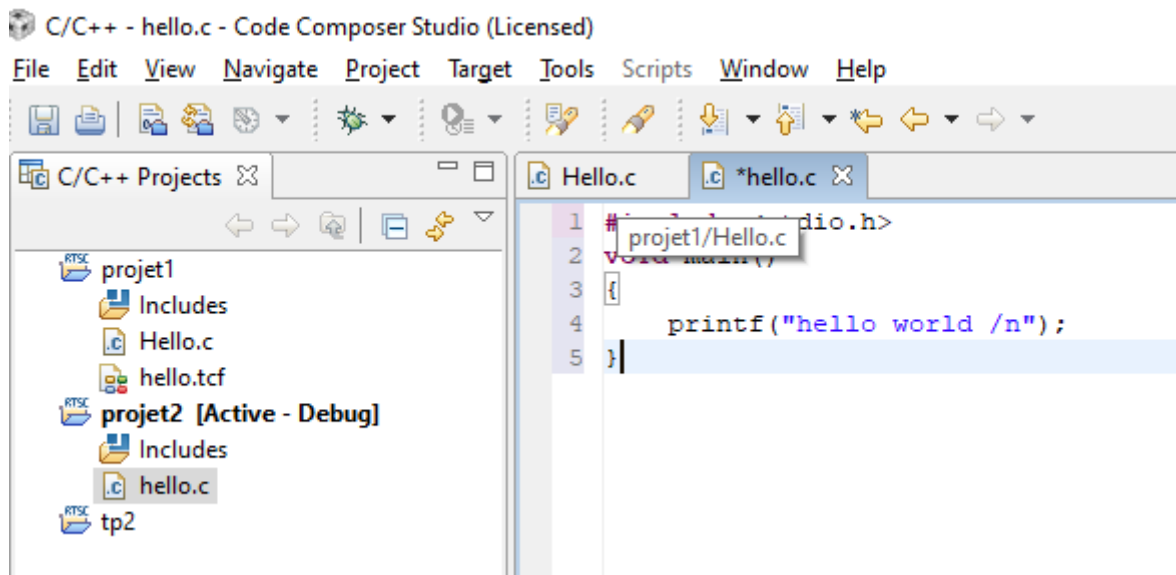
- Pour le projet Template choisir dans le menu « DSP/BIOS v5.xx Examples » le champ « Empty Exemple ».

2. Les étapes d'ajout des fichiers dans un projet

Puisque le projet crée est un projet vide, il faut ajouter les fichiers sources contenant le programme à exécuter. On ajoute un nouveau fichier source par l'activation du menu File → New → Source File ou bien par l'activation du bouton droit sur le nom du projet NewSource → File.

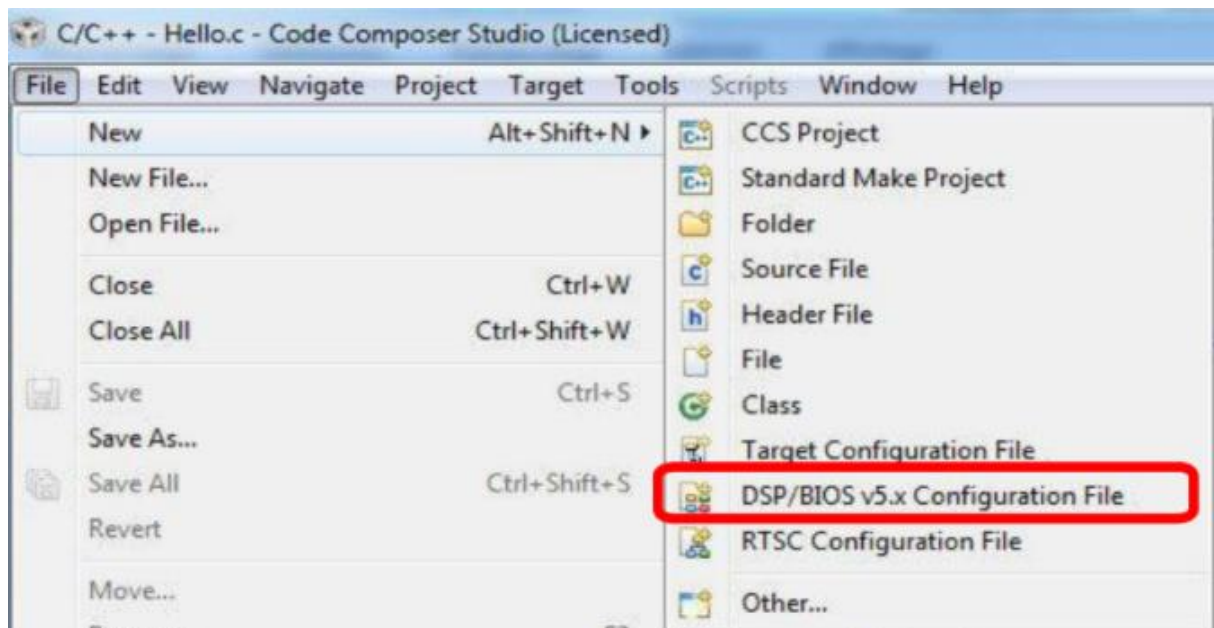
On Attribue un nom au fichier C crée tout en indiquant l'extension .c, par exemple "hello.c" et cliquer sur Finish.

on va juste afficher le message "hello world".

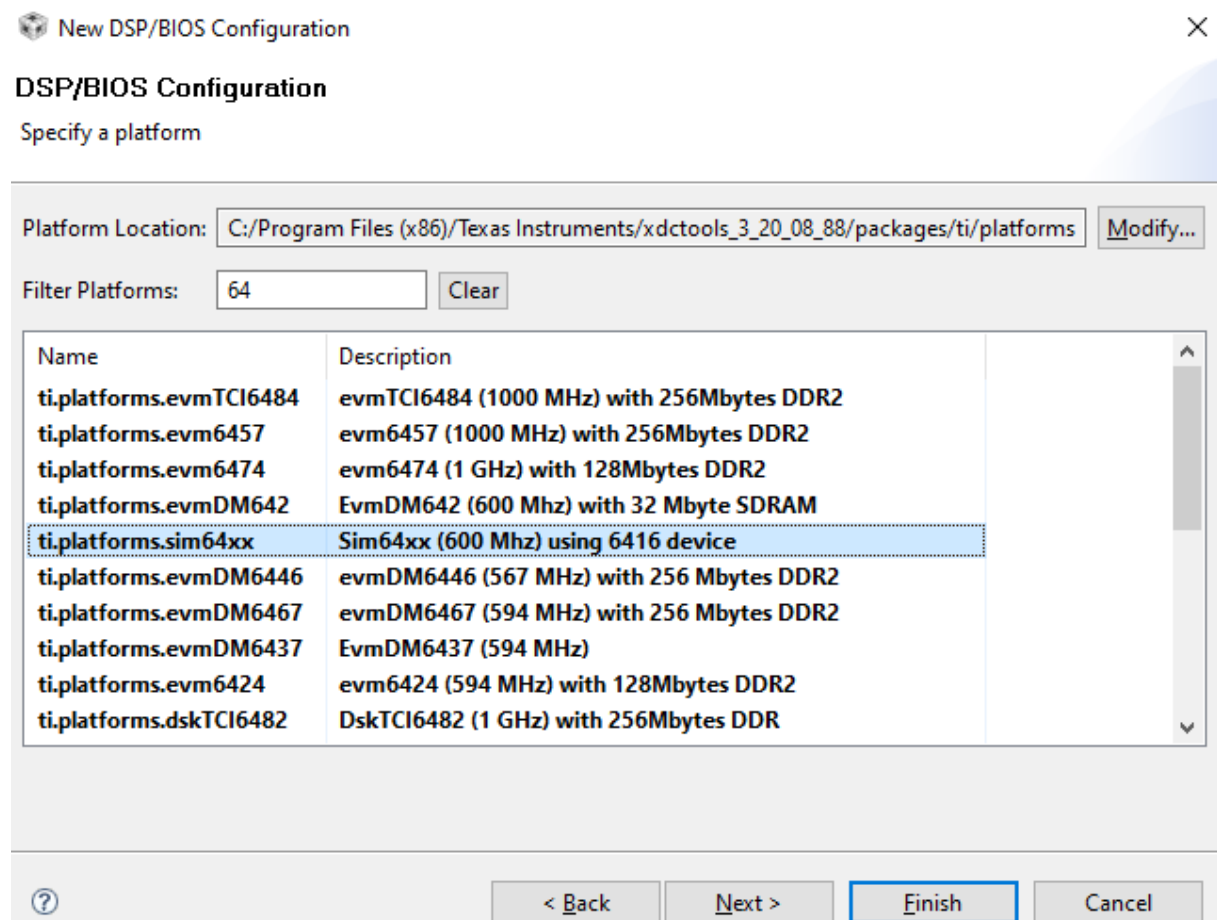


3. Ajout du DSP/BIOS v5.x Configuration File

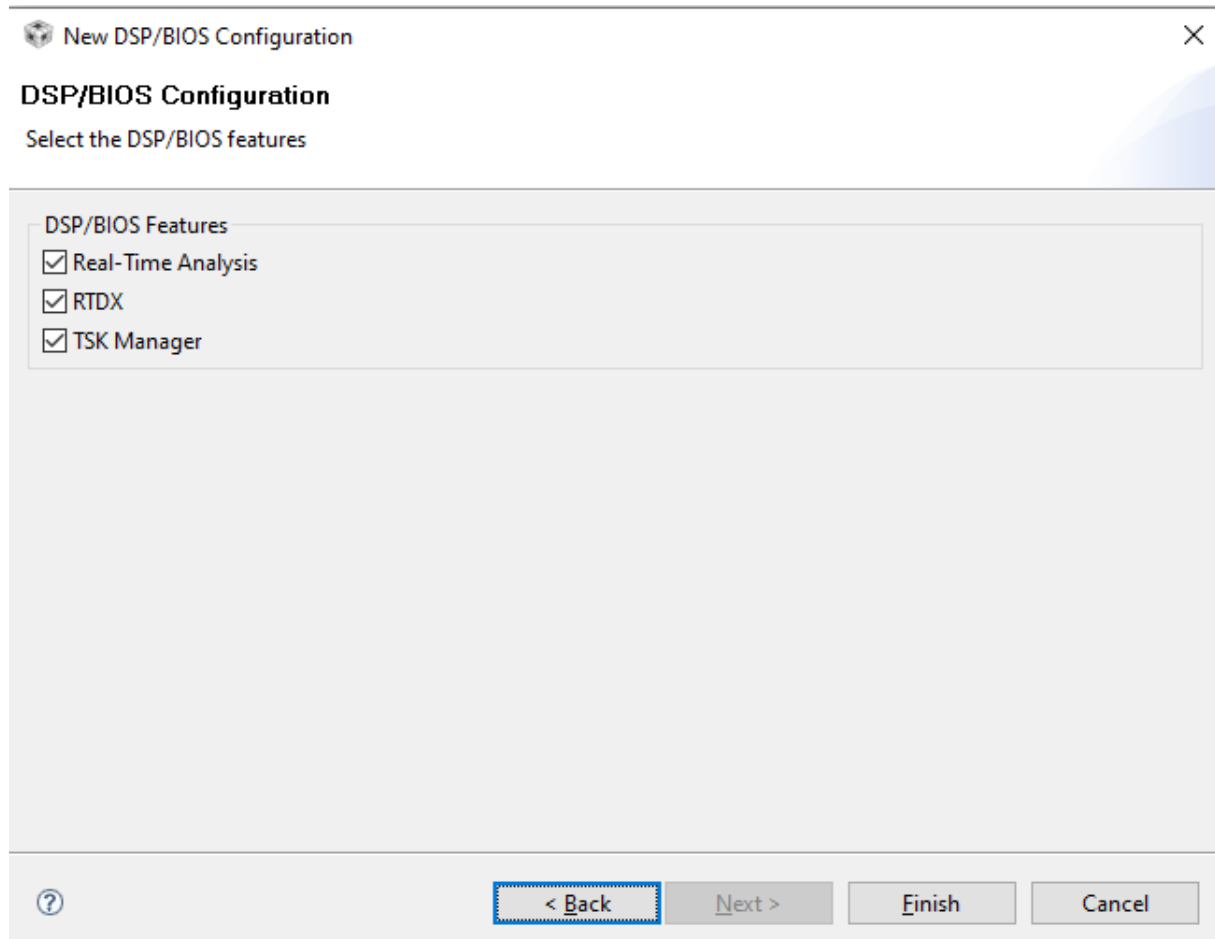
- ❖ Pour ajouter le fichier de configuration de DSP/BIOS, sélectionner File → New → DSP/BIOS v5.x Configuration File.



- ❖ Donner un nom au fichier de configuration de DSP/BIOS, par exemple "hello.tcf" et cliquer sur Next.
- ❖ Spécifier la plateforme DSP à utiliser. Pour notre cas, choisir la plateforme simulateur 6416. Pour faciliter la recherche, utiliser le filtre de plateforme (Filter Platforms) comme indiqué sur la figure et cliquer sur Next



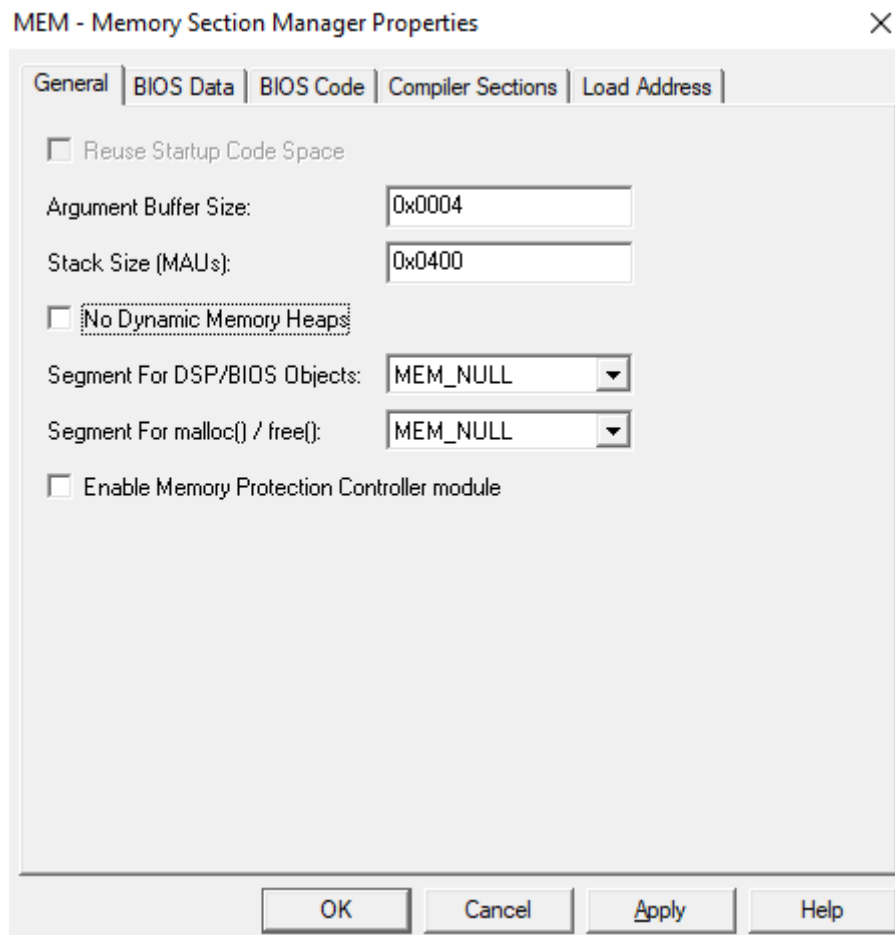
- ❖ Vérifier que les paramètres de configuration sont tous cochés et cliquer sur Finish.



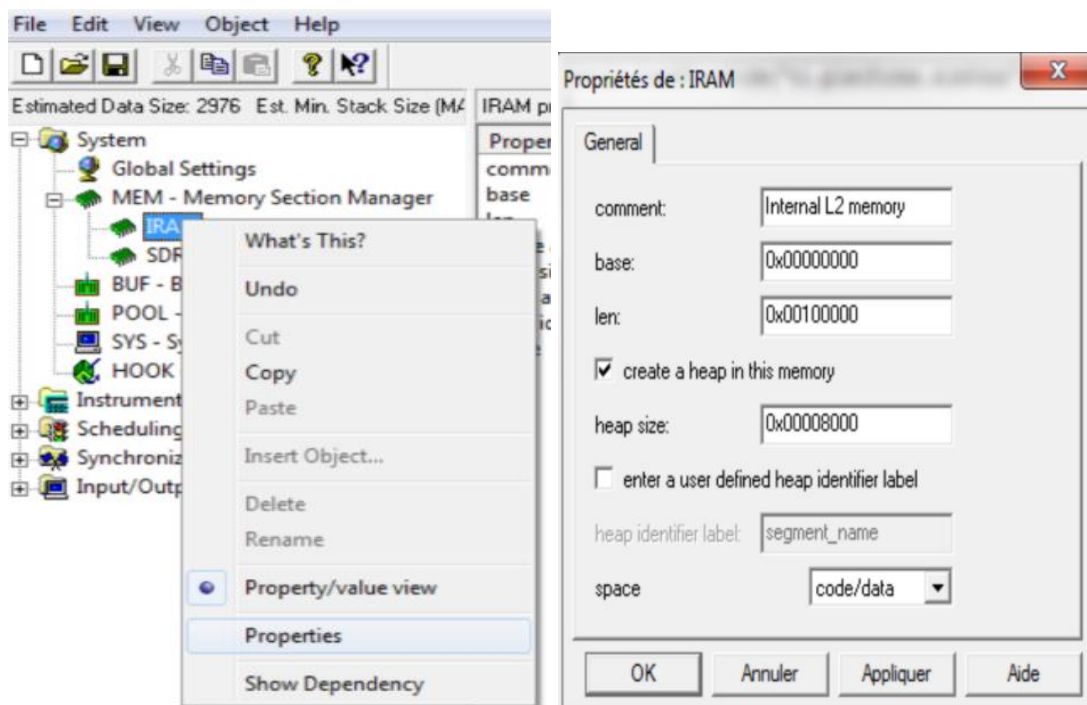
En cliquant sur finish, le fichier de configuration s'ouvre.

Ce fichier permet de faire une gestion des mémoires (création des heaps mémoires, définir la taille de la mémoire cache, etc ...), création des tasks, faire la synchronisation entre les cores en ajoutant des sémaphores, modifier les paramètres de RTDX (Real Time Data EXchange) tel que le RTDX mode : JTAG ou bien simulator, etc ...

- ❖ Dans cette étape, on va configurer les différentes sections mémoires. Dans le fichier de configuration DSP/BIOS "hello.tcf" cliquer avec le bouton droit sur MEM-Memory → Section Manager → Properties :
Décocher la case "No Dynamic Memory Heaps" et cliquer sur OK. Cette étape permet de créer des heaps (réserver des zones mémoires) dans les mémoires du DSP. Cliquer sur OK.

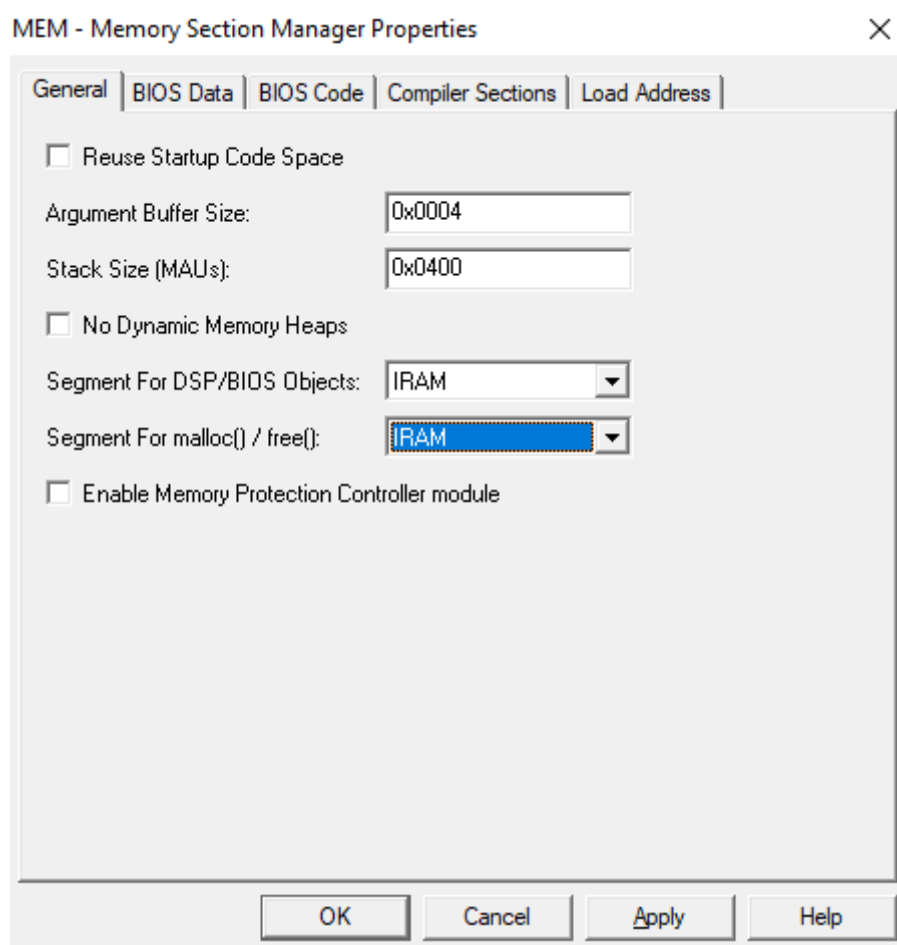


- ❖ Pour choisir maintenant le type de mémoire où on va créer un heap mémoire (par exemple dans la mémoire interne de chaque DSP IRAM), il faut suivre les deux étapes suivantes :
 - Cliquer avec le bouton droit de la souris sur IRAM Properties
 - Cocher la case “Create a heap in this memory” et cliquer sur OK.



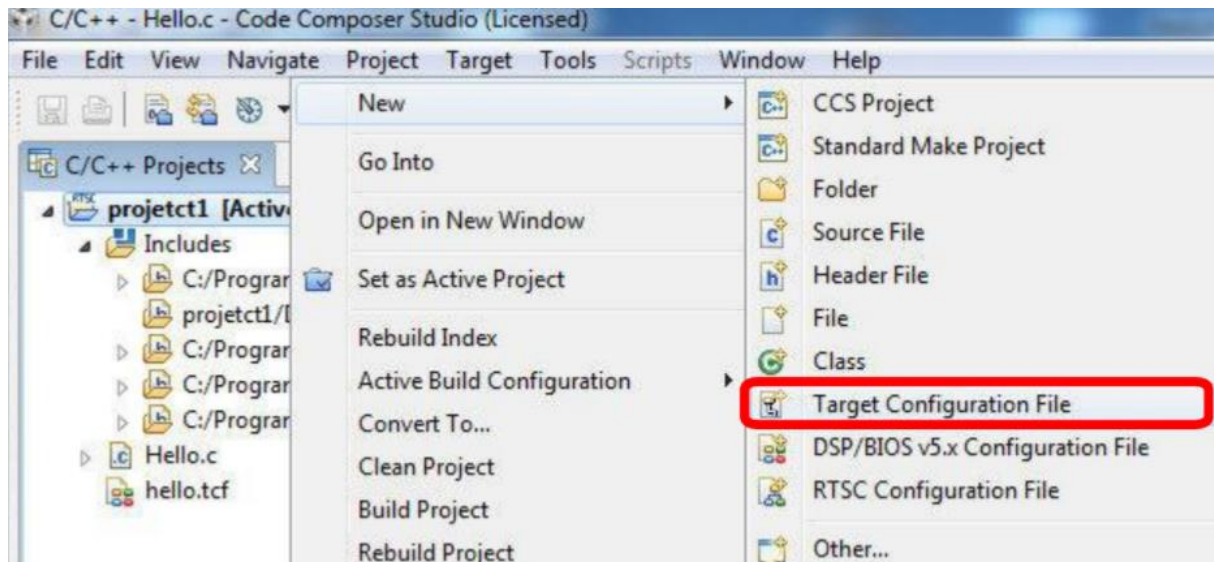
Pour activer la création de zone mémoire “heap” de mémoire SDRAM qui est la mémoire externe, il suffit de refaire les mêmes étapes.

- ❖ Pour configurer les différentes sections à créer, retourner au MEM-Memory Section → Manager → Properties.



Choisir un type de mémoire pour “Segment For DSP/Bios Objects” et “Segment For malloc()/free()”. Les mémoires qu’on peut choisir sont les mémoires où on a crée des heaps mémoires dans l’étape 6.

- ❖ Vérifier dans la même fenêtre de Properties que le type de la mémoire réservée pour tous les champs de “BIOS Data”, “BIOS Cache” et “Compiler Section” est la mémoire interne IRAM. Puis cliquer OK. Cette étape permet la bonne gestion des mémoires et par la suite le bon fonctionnement du DSP.
- ❖ Ajouter un “Target Configuration File” en cliquant avec le bouton droit sur le nom du Projet → New → Target Configuration File ou bien File → New → Target Configuration File :



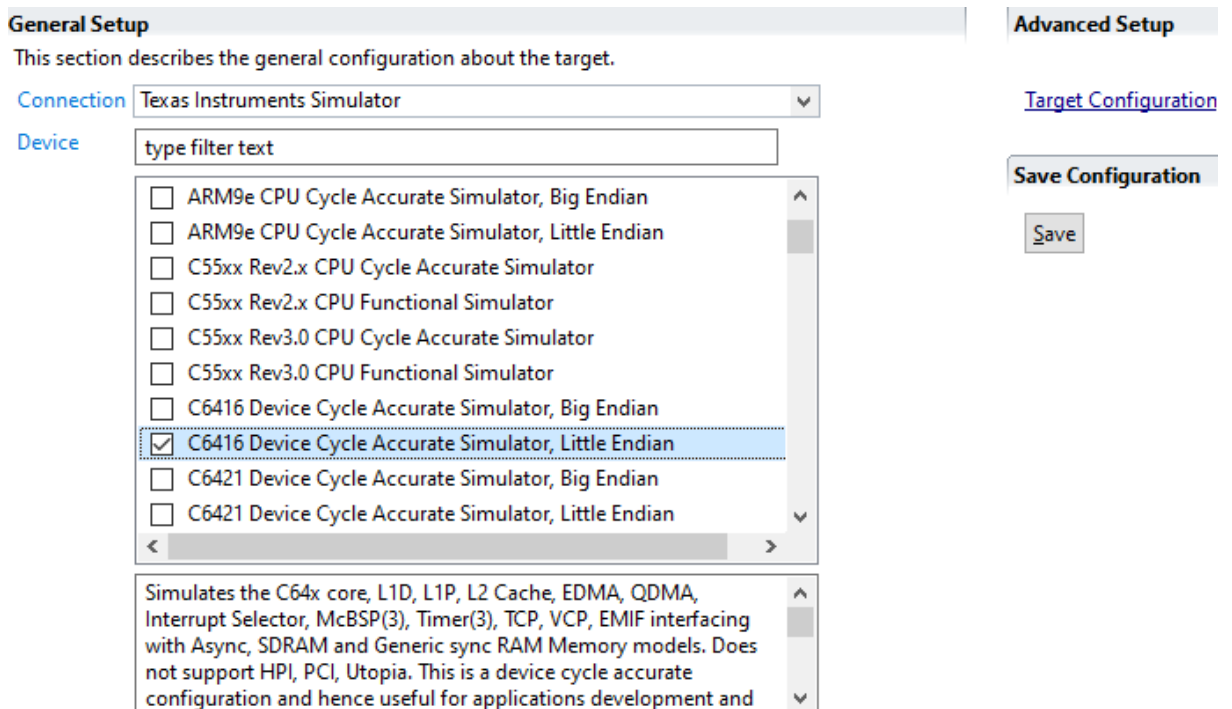
On attribue un nom "hello.ccxml" et clique sur Finish.

- ❖ Il faut définir la cible de l'implantation. On peut choisir le nom du DSP utilisé et sélectionner un simulateur ou bien un émulateur.

Dans le cas d'un simulateur, il suffit de choisir la configuration suivante :

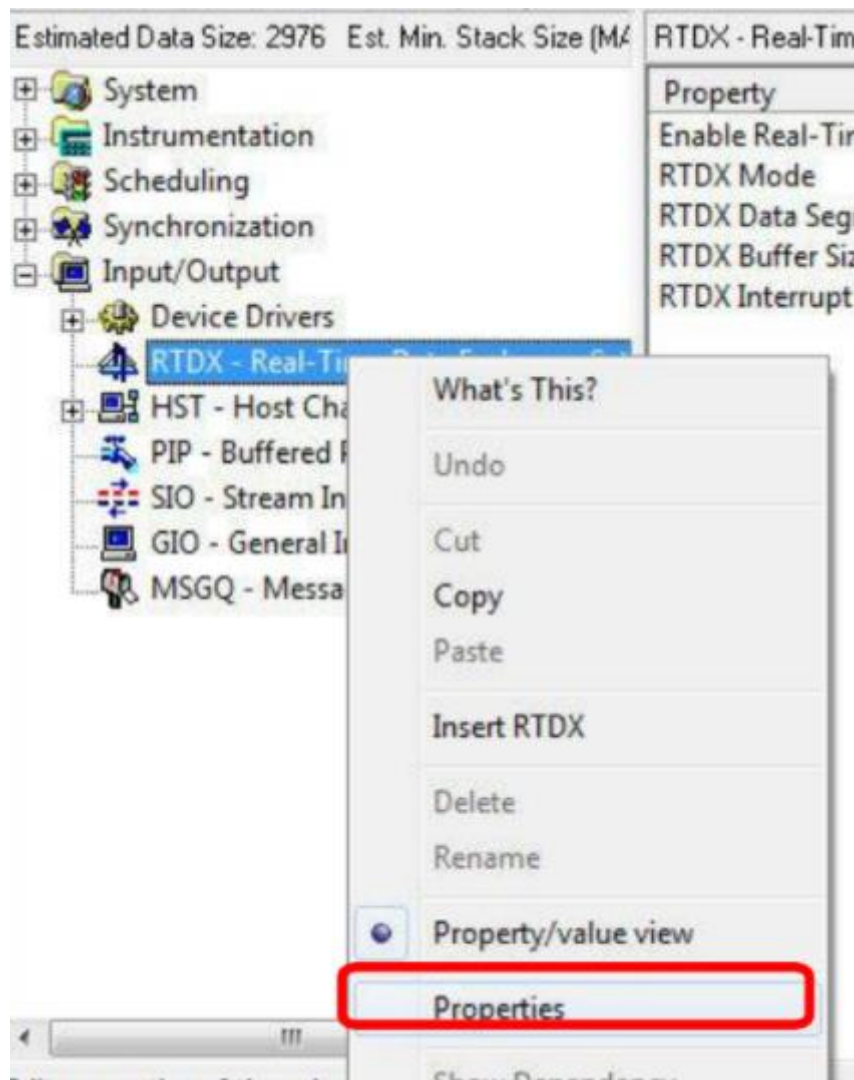
- Connection : Texas Instruments Simulator
- Device : C6416 Device Cycle Accurate Simulator, Little Endian

Cliquer sur "Save" pour enregistrer la configuration choisie :



Note: Support for more devices may be available from the update manager.

- ❖ Pour choisir de travailler avec le simulateur ou sur la plateforme, il faut définir le mode RTDX (Real Time Data EXchange) correspondant. Sur le fichier de configuration de DSP/BIOS "hello.tcf", ouvrir le menu Input/Output, avec le bouton droit, sélectionner RTDX-Real-Time Data Exchange Settings → Properties :



Dans le cas d'un émulateur, sélectionner "JTAG" pour le RTDX mode.

Dans le cas d'un simulateur, sélectionner "Simulator" pour le RTDX mode.

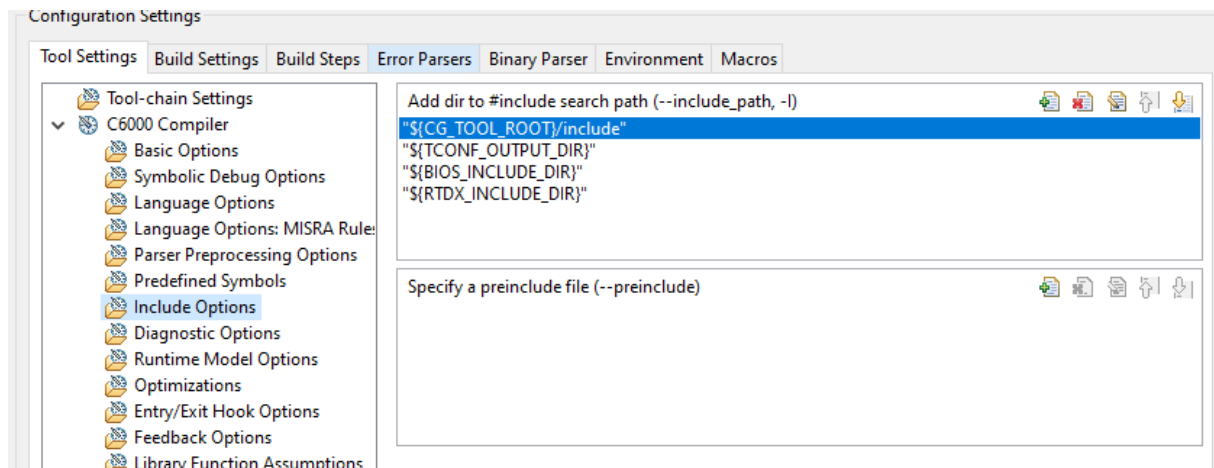
→ En terminant toutes les étapes de création du projet et d'ajout de fichier nécessaire, ainsi que la configuration nécessaire on peut passer à la compilation du projet.

IV. EXECUTER LE PROJET

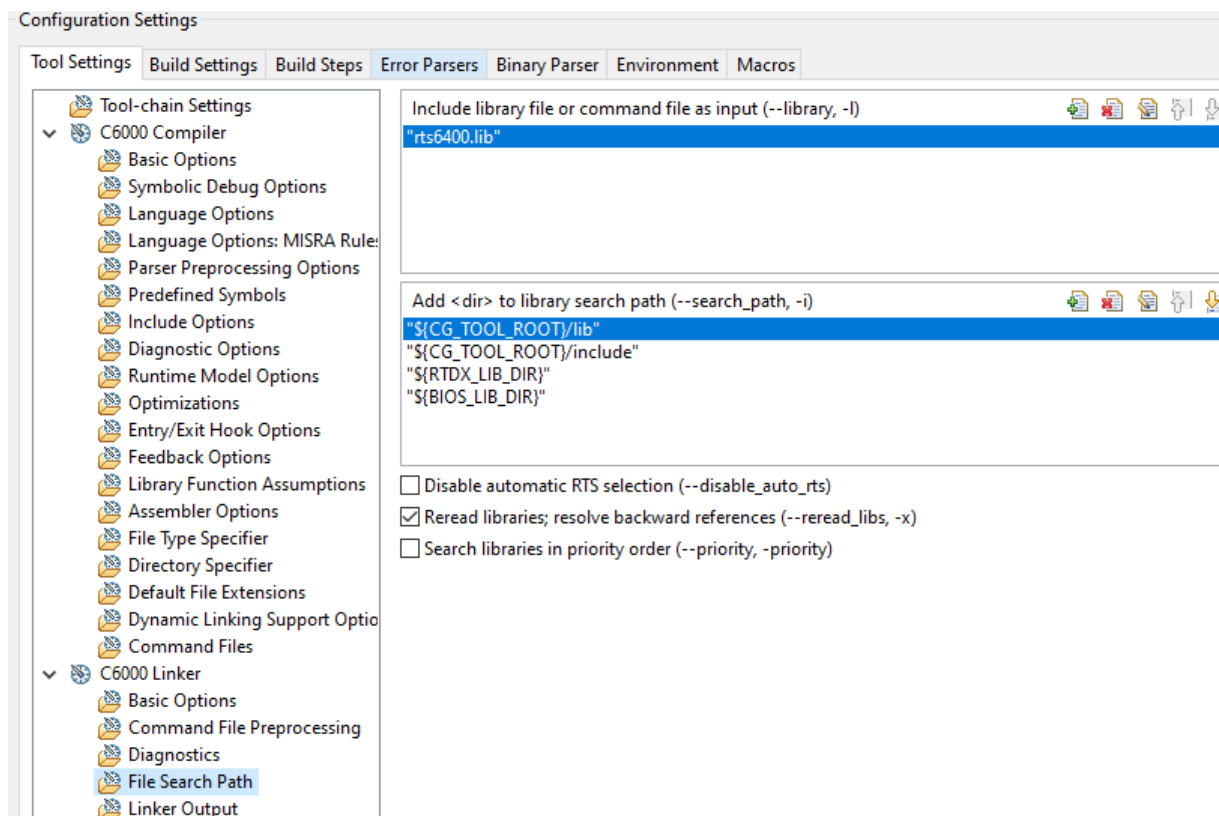
a. Ajouter les chemins des bibliothèques

Avant de commencer la compilation du projet, Il faut ajouter les chemins des bibliothèques nécessaires s'ils existent. Par exemple, si on utilise dans l'application " #include hello.h ", il faut ajouter le chemin du répertoire contenant le fichier " hello.h ". Il suffit donc de suivre ces étapes :

- Cliquer avec le bouton droit sur le nom du projet.
- Cliquer sur Build Properties.
- Sélectionner "Include Options".
- Cliquer sur l'icône Add.
- Suivre le chemin du fichier " hello.h ".

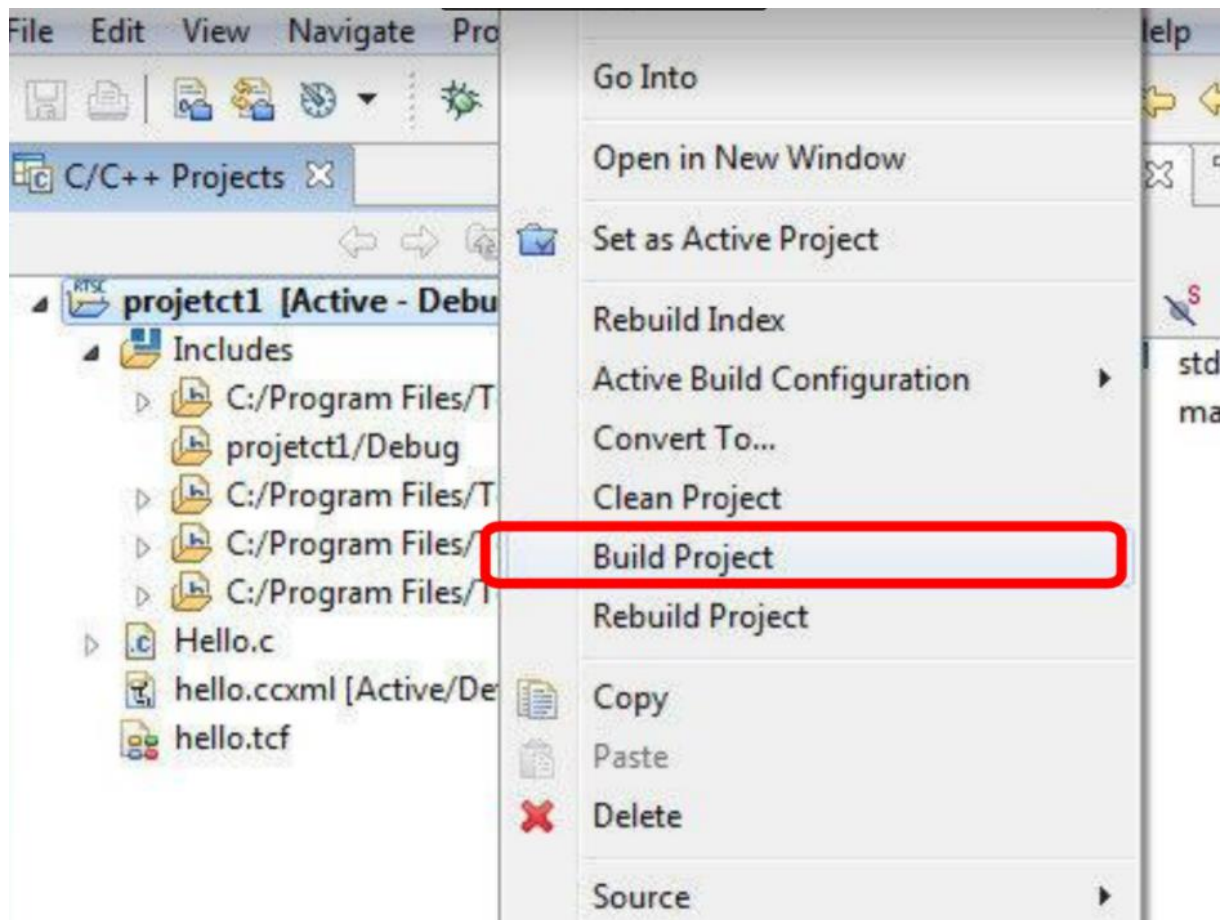


Même chose pour les bibliothèques “.lib ” utilisées. Sélectionner FileSearch Path dans la fenêtre de Build Properties et ajouter le nom de la bibliothèque et donner le chemin du répertoire contenant le fichier .lib utilisé.



b. compiler le projet

Pour compiler le projet il suffit de Cliquer par le bouton droit sur le nom du projet et sélectionner “ Build Project ” ou bien aller vers Project → Build Active Project :



Si la compilation se termine avec succès (pas d'erreurs), un fichier exécutable " Project1.out" sera généré.

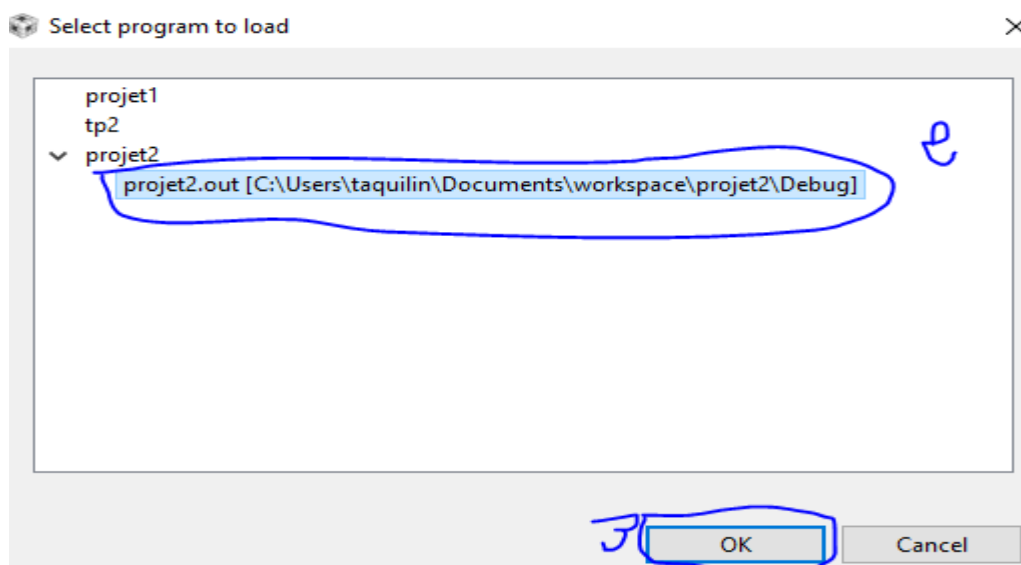
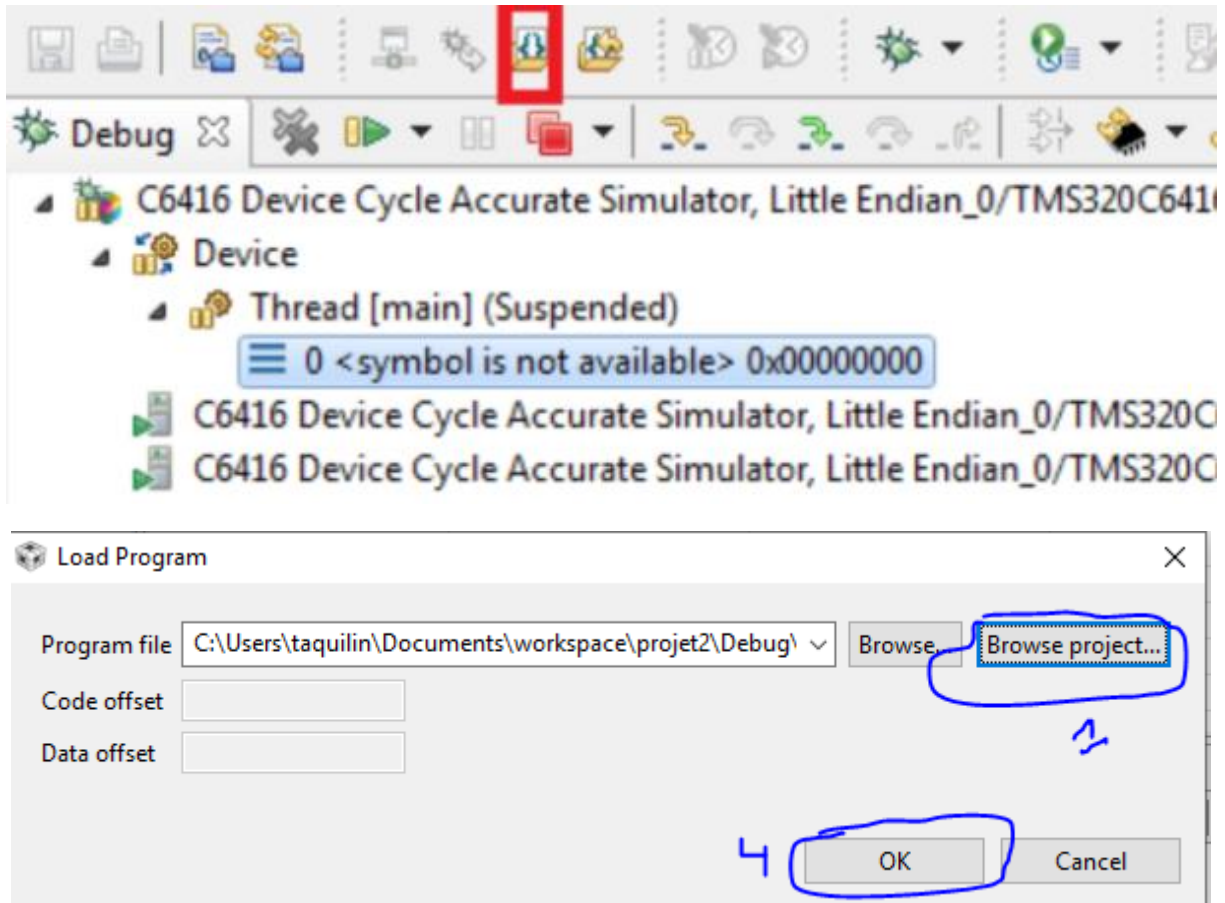
```
Instruments/ccsv4/tools/compiler/c6000/lib" -i"C:/Program Files (x86)/Texas
Instruments/ccsv4/tools/compiler/c6000/include" -i"C:/Program Files (x86)/Texas
Instruments/bios_5_41_10_36/packages/ti/rtdx/lib/c6000" -i"C:/Program Files (x86)/Texas
Instruments/bios_5_41_10_36/packages/ti/bios/lib" --reread_libs --rom_model -o "projet2.out"
-l"./hellocfg.cmd" -l"./hellocfg_c.obj" -l"./hellocfg.obj" -l"./hello.obj" -l"rts6400.lib"
<Linking>
'Finished building target: projet2.out'
'
Build complete for project projet2
```

Dans cette étape on doit établir une connexion entre le Code Composer et la plateforme cible "Target", que ce soit simulateur ou emulateur. Sélectionner le menu View Target configurations. Le fichier de configuration "hello.ccxml" apparaît dans la fenêtre "Target configuration view ". Ouvrir le fichier de configuration avec le bouton droit de la souris sur hello.ccxml et sélectionner " Launch Selected Configuration ".

En se connectant entre le Code Composer et le target, on peut maintenant charger le programme " Project1.out " dans le CPU choisi.

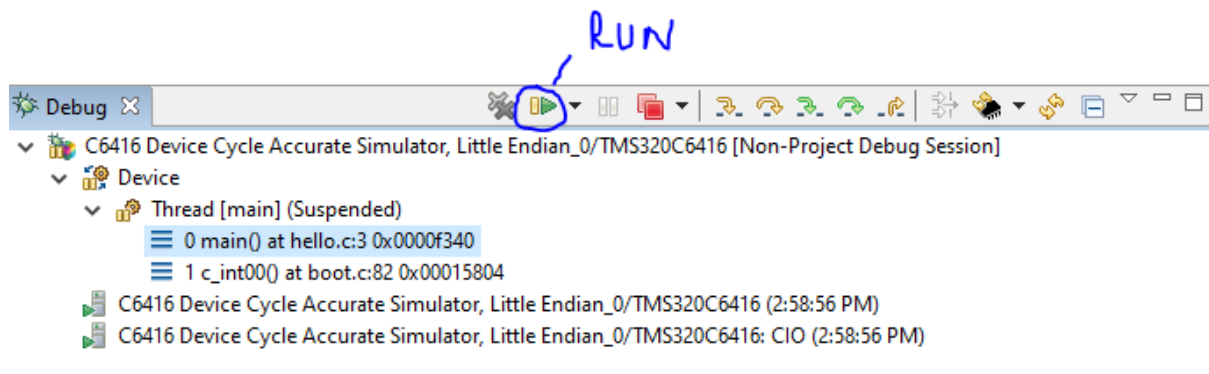
Pour faire le chargement du programme dans le core0 par exemple, il faut sélectionner le core 0 par la souris : Load program définir le répertoire de fichier exécutable généré (Project1.out).

En général le fichier exécutable est généré dans un dossier nommé Debug :



Dès que le chargement est terminé, on peut exécuter le programme.

Run pour lancer l'exécution :

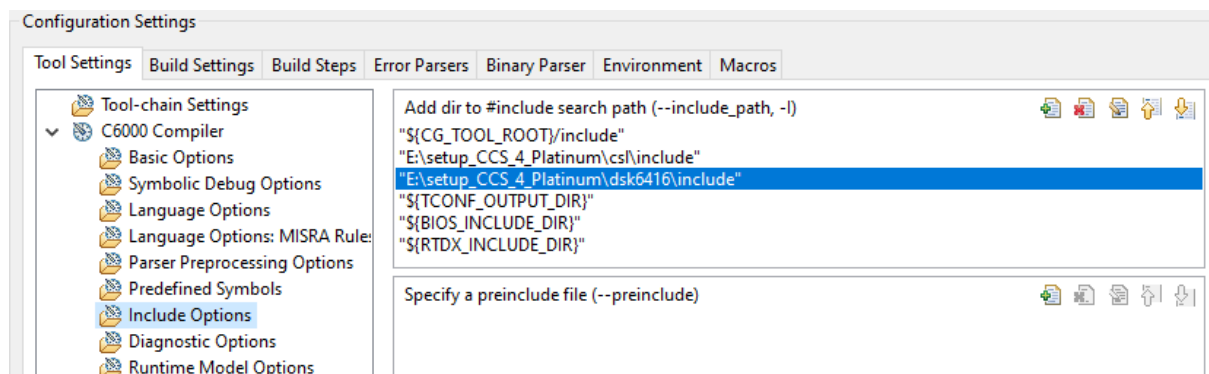


V. OPTIMISATION

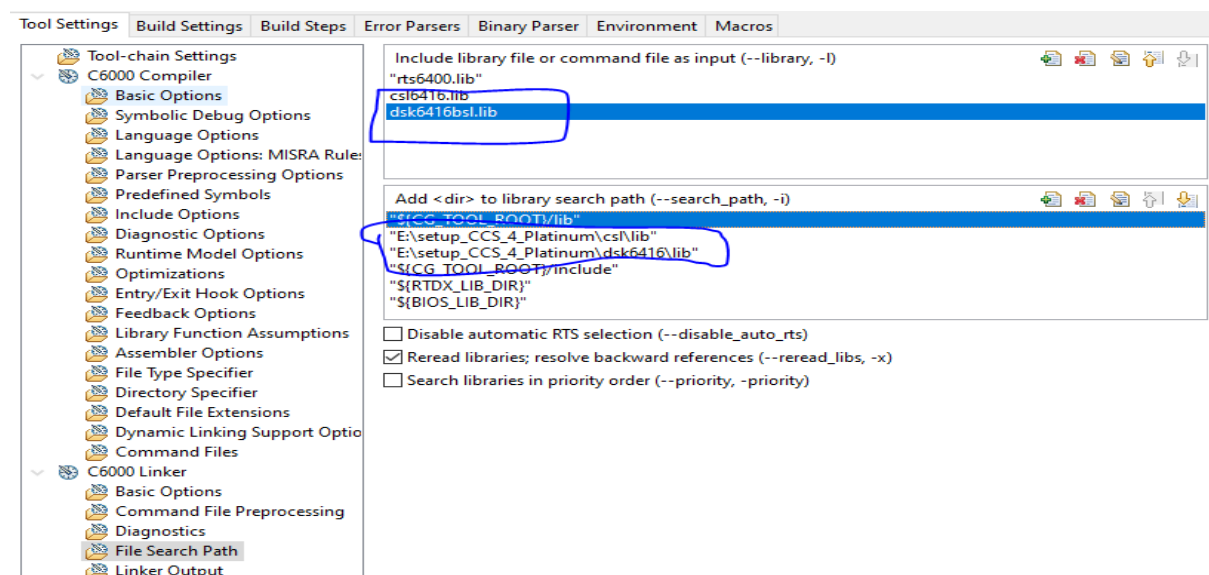
i. Ajouter les bibliothèques

Avant de commencer la compilation du projet, Il faut ajouter les chemins des bibliothèques nécessaires.

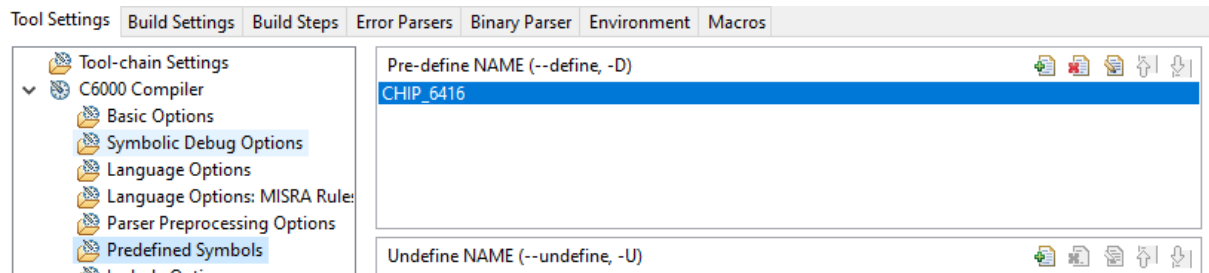
- Pour « include options »



- Pour « File Search Path »

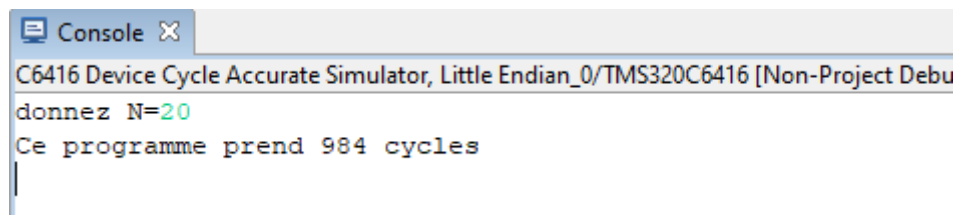


- Définir le « Predefined Symbols » utilisé



ii. Compiler le projet

On exécute le code pour avoir le résultat :



Sachant que le code est le suivant :

```

1 #include "BIB.h"
2 void main()
3 {
4     unsigned char src[100];
5     unsigned char dst[100];
6     int i,N;
7     unsigned int start, stop; // timer count store
8     hTimer1 = TIMER_open(TIMER_DEV1, NULL);
9     DSK6416_init_timer(timer1Ctl,hTimer1);
10    // start timer
11    TIMER_start( hTimer1);
12    start = TIMER_getCount(hTimer1);
13    /*****IMPORTANT*****/
14    N peut prendre l'une de ces valeurs: 10,20,30,40,50,60,70,80,90,100****/
15
16    printf("donnez N=");
17    scanf("%d",&N);
18    for(i=0;i<100;i++)
19    {
20        src[i]=i+20;
21    }
22    /* for(i=0;i<100;i++)
23    {
24        src[i]=7*100+i*20;
25        printf("%d\n",src[i]);
26    }*/
27    for(i=0;i<N;i++)
28    {
29        dst[i]=i;
30    }
31    stop = TIMER_getCount(hTimer1);
32    printf("Ce programme prend %u cycles\n", (stop-start));
33
34 }
```

On déduit alors que pour exécuter deux boucles itératives : l'une avec 100 itérations et l'autre choisie et dans notre cas c'est 20.

On a pris 984 cycles pour le réaliser.

VI. IMAGE PROCESSING

iii. Ajouter les chemins des bibliothèques

Avant de commencer la compilation du projet, Il faut ajouter les chemins des bibliothèques nécessaires pour calculer le nombre de cycle.

Vue qu'on utilise la même bibliothèque que le projet Optimisation on refait les mêmes étapes

iv. Modification du code source

Pour ajouter la fonctionnalité qui compte le nombre de cycle comme dans le projet optimisation

On va rectifier notre code source initial comme suit :

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "BIB.h"
#define TAILLE 9485 //Taille de l'image bmp
void main()
{
    int i;
    unsigned char *pixel,*pixel_out, diff=0;
    unsigned int start, stop;// timer count store

    FILE *image_in; //pointeur sur l'image d'entrée
    FILE *image_out;//pointeur sur l'image de sortie
    pixel=malloc(TAILLE* sizeof(unsigned char)); //Allocation dynamique
de la taille de l'image
    pixel_out=malloc(TAILLE* sizeof(unsigned char)); //Allocation
dynamique de la taille de l'image
    memset(pixel,0, TAILLE);//initialisation à zéro de l'espace alloué

    image_in = fopen("input1.jpg","rb") ; // ouverture du fichier image
d'entrée
    image_out= fopen("output.jpg","wb") ;//ouverture du fichier image de
sortie
    fread(pixel,1,TAILE,image_in);//lecture et stockage de l'image
d'entrée dans l'espace mémoire alloué
    // init timer
    hTimer1 = TIMER_open(TIMER_DEV1, NULL);
    DSK6416_init_timer(timer1Ctl,hTimer1);
    // start timer
    TIMER_start( hTimer1);
    start = TIMER_getCount(hTimer1);
    // copiage de l'entete
    for(i=0;i<359;i++)
    {
        pixel_out[i]= pixel[i];
```

```

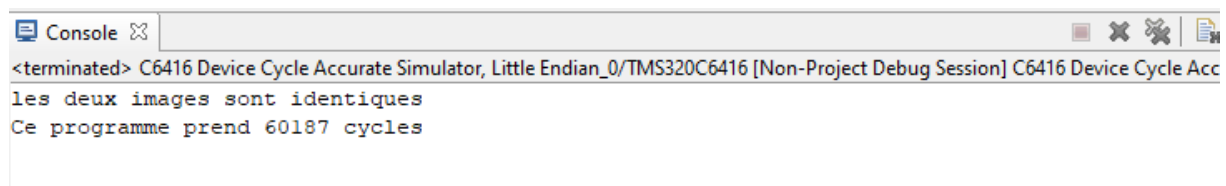
    }
    /*****fin du processus de copiage de l'entete*****/
    /*****copiage des pixels*****/
    for(i=359;i<9485;i++)
    {
        pixel_out[i]=pixel[i];
    }
    /*****fin du processus de copiage des pixels*****/
    /*****comparaison entre l'image source et l'image
reconstruite*****/
    for(i=0;i<9485;i++)
    {
        diff+=(pixel_out[i]-pixel[i]);
    }
    if(diff==0)
        printf("les deux images sont identiques\n");
    else
        printf("les deux images sont différentes\n");
    stop = TIMER_getCount(hTimer1);
    printf("Ce programme prend %u cycles\n", (stop-start));
    /*****fin du processus de comparaison*****/
    fwrite(pixel_out,1,TAILLE,image_out); //écriture de contenu du
pointeur pixel dans le fichier image de sortie

    free(pixel); //libérer l'espace mémoire alloué
    fclose(image_in);
    fclose(image_out);
}

```

v. Compiler le projet et exécution

On compile le projet et on l'exécute. On obtient le résultat suivant :



```

<terminated> C6416 Device Cycle Accurate Simulator, Little Endian_0/TMS320C6416 [Non-Project Debug Session] C6416 Device Cycle Acc
les deux images sont identiques
Ce programme prend 60187 cycles

```

Noter programme va parcourir l'image pixel par pixel et copier dans un autre fichier image puis comparer si l'image source et identique à l'image destination.

Dans notre cas l'image elle été copier et les deux images sont identiques.

Pour faire ces opérations on a effectué 60187 cycles en mode debug. Ce qui est largement supérieur au nombre de cycles effectuer par le projet optimisation.

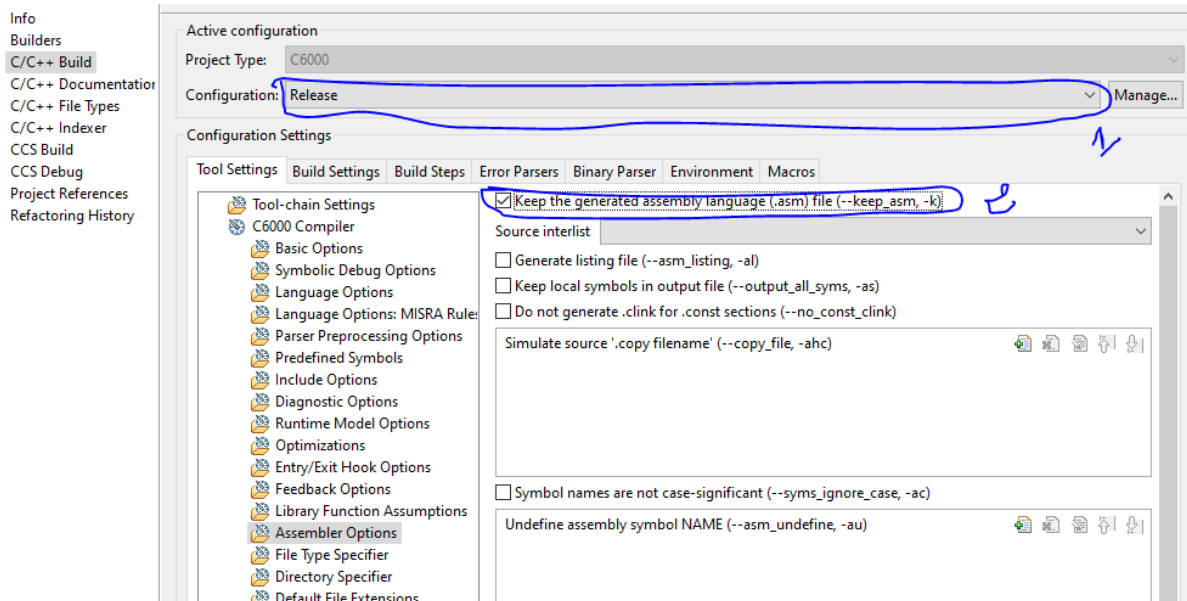
Cela s'explique par nombre de complexité du programme et la tâche plus compliqué (nombre d'opérations élevés).

VII. IMAGE PROCESSING AVEC OPTIMISATION

vi. Changer en mode release

Afin de changer le mode release il faut suivre les étapes suivantes :

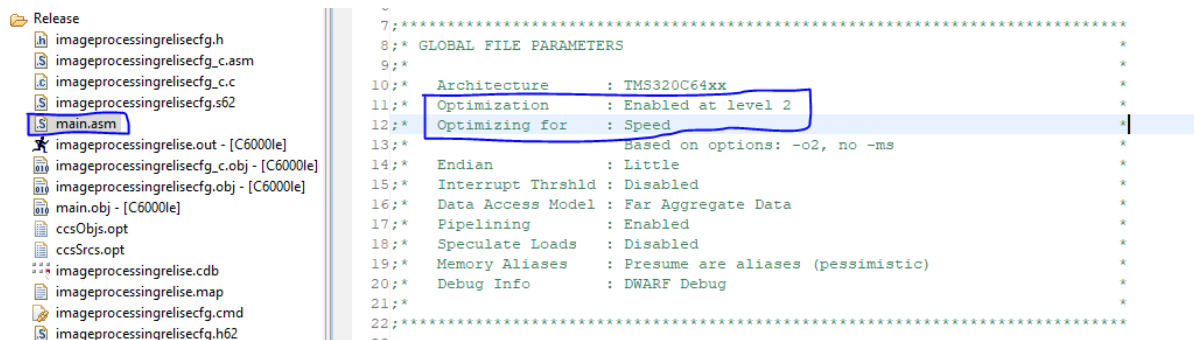
- Clic droit sur le projet → propriétés → C/C++ build → configuration : on le passe en mode Release
- Ensuite on clic sur Assembler Option et on active « keep the generated assembly language »



- Puis on clic sur OK

vii. information assembleur

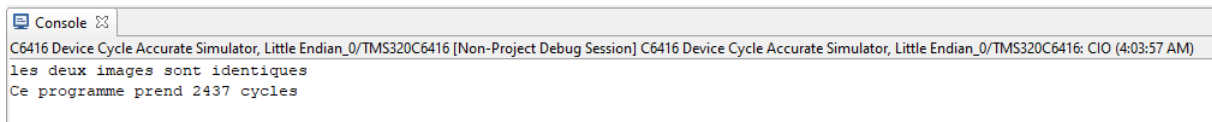
Après le build en mode release on trouve un fichier main.asm contenant les informations suivantes dont deux nous intéressent : optimization et Optimization for.



Ces informations indiquent que l'optimisation est active avec un niveau 2 et que le type d'optimisation influe sur la vitesse d'exécution des jeux d'instructions.

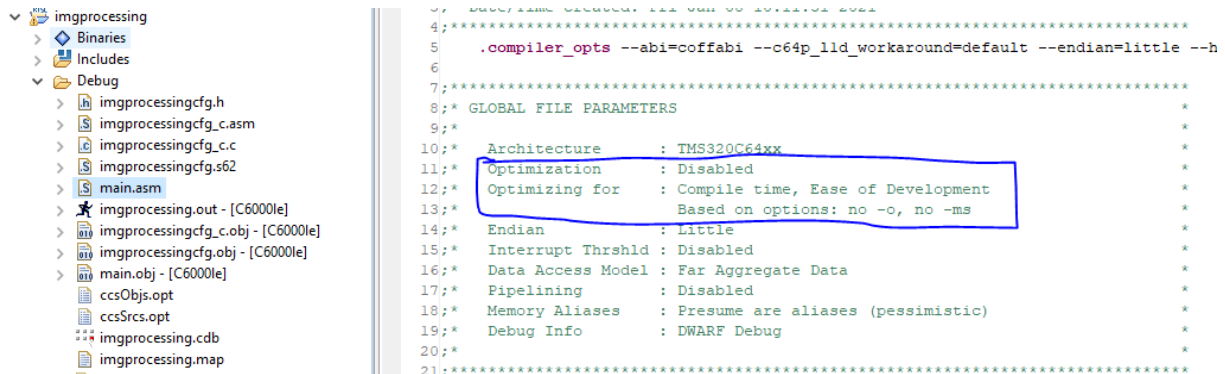
viii. Exécution du code

Après l'exécution on obtient le résultat suivant :



Pour cet exemple on a effectué la copie de l'image avec un temps de 2437 cycles par rapport au temps de l'exécution de copie d'image sans optimisation qui présente 60187 cycles.

Cela est expliqué par le fait qu'en mode « debug » le mode optimisation est désactivé.



Les instructions sont exécutées série une par une alors que dans le cas avec optimisation l'assembleur exécute les jeux d'instructions en parallèle.

D'où si on calcule **l'efficacité = (temp principal - temp optimisé) / temp principal**

→ Efficacité = $(60187 - 2437) / 60187 = 0.9595$

→ Efficacité = 95,95 %

ix. Optimisation de structures itératives

Il est possible d'optimiser le traitement de structures itératives ou boucles par déroulage. Le déroulage.

Pour effectuer ce type d'optimisation, nous allons utiliser les instructions de type pragma. Cette directive du préprocesseur offre deux fonctions de déroulage de boucles : UNROLL() et MUST_ITERATE().

Afin d'effectuer une optimisation optimale, le choix du nombre de déroulage est très important. Il représente le nombre d'itérations de cette boucle nécessaire à l'obtention du temps le plus optimisé sans pour autant en affecter les résultats.

UNROLL(N)

Pour déterminer ce nombre, nous procéderons par choix de N avec un peu de chance et de hasard

→ Pour N = 8

```
Console
C6416 Device Cycle Accurate Simulator, Little Endian_0/TMS320C6416 [N
les deux images sont identiques
Ce programme prend 2438 cycles
```

On a un nombre de cycle = 2438 > 2437

→ Pour N= 7

```
Console
C6416 Device Cycle Accurate Simulator, Little Endian_0/TMS32
les deux images sont identiques
Ce programme prend 2474 cycles
```

On a un nombre de cycle = 2478 > 2437

→ Pour N= 16

On a un nombre de cycle = 2430 < 2437

→ Efficacité = $(60187 - 2430) / 60187 = 0.9596$

→ Efficacité = 95,96 %