

Compte rendu des TP : Code Composer Studio V4

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.

IDE et langage de programmation :

Code Composer Studio comprend une suite d'outils utilisés pour développer et déboguer des applications embarquées. Il comprend un compilateur d'optimisation C / C ++, un éditeur de code source, un environnement de génération de projet, un débogueur, un profileur et de nombreuses autres fonctionnalités.

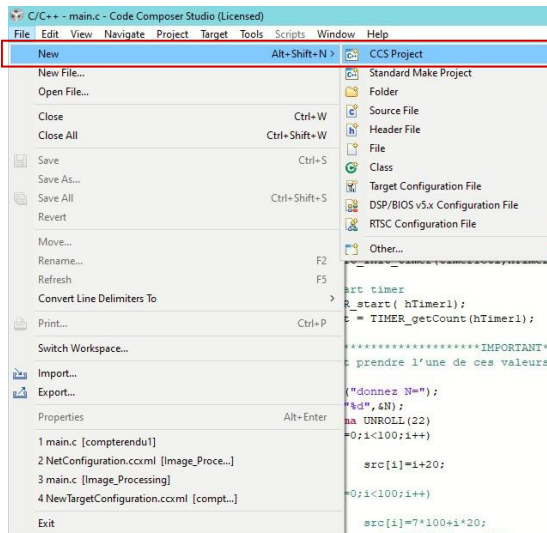


TP 1 : INITIATION AU CODE COMPOSER STUDIO V4

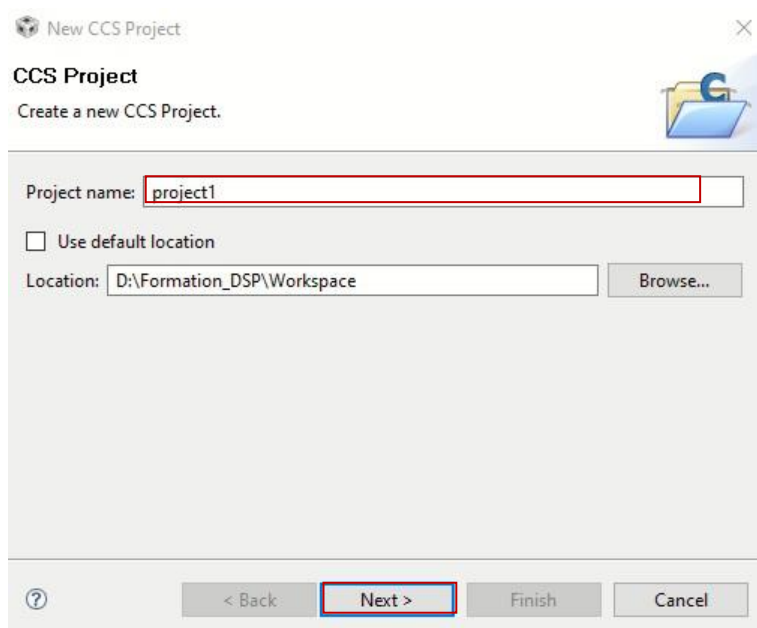
1- Création d'un projet

1.1 Les étapes de création d'un projet

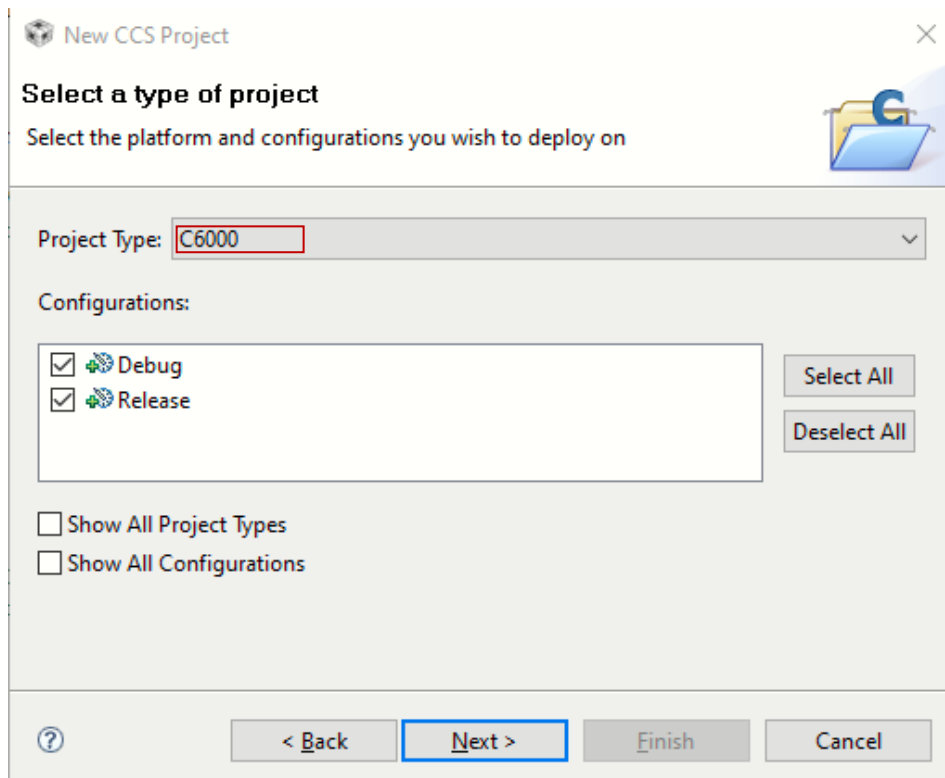
- 1) Démarrer le logiciel Code Composer Studio v4 :
- 2) Créer votre répertoire de travail (Workspace) :
- 3) Créer un nouveau projet par l'activation de File New CCSProject :



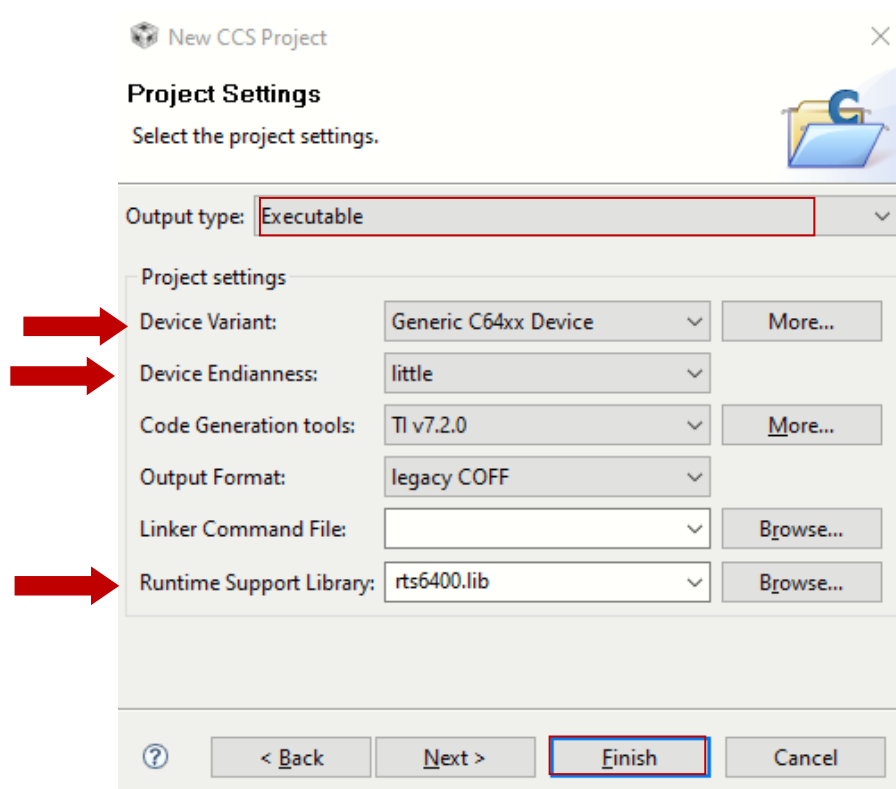
- 4) Attribuer un nom au projet par exemple "project1", vérifier son emplacement et par la suite cliquer sur Next.



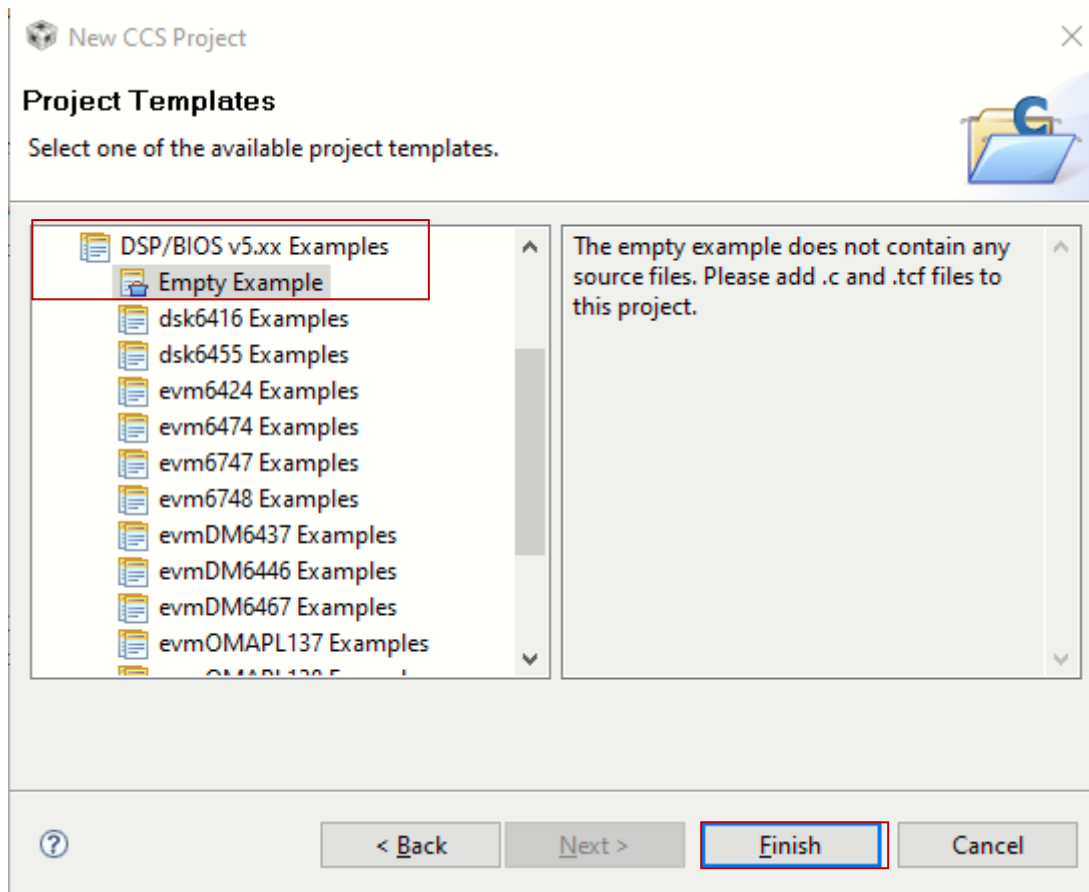
- 5) Choisir le type la famille de DSP à utiliser : (C6000).
- 6) Activer les deux modes de compilation : Debug et Release.



7) Préciser les différents paramètres du projet, à savoir

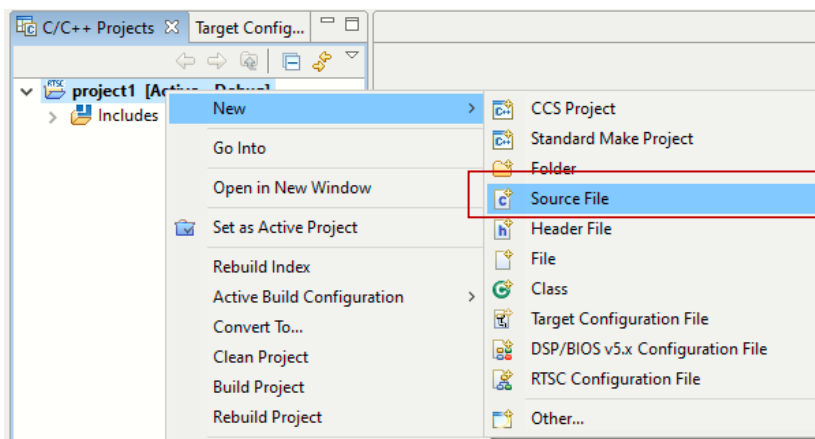


Pour le projet Template choisir dans le menu « DSP/BIOS v5.xx Examples » le champ « Empty Exemple ».



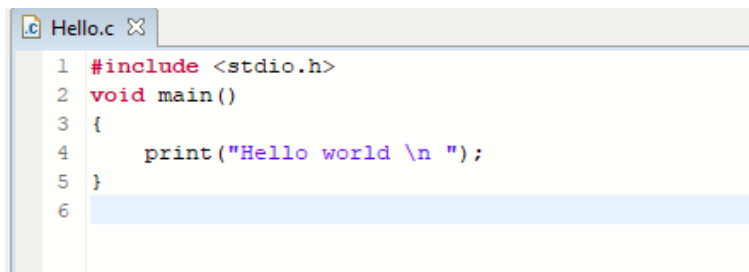
1.2 Ajout des fichiers au projet :

Puisque le projet crée est un projet vide, il faut ajouter les fichiers sources contenant le programme à exécuter.



Attribuer un nom au fichier C crée tout en indiquant l'extension .c, par exemple "**Hello.c**" et cliquer sur Finish.

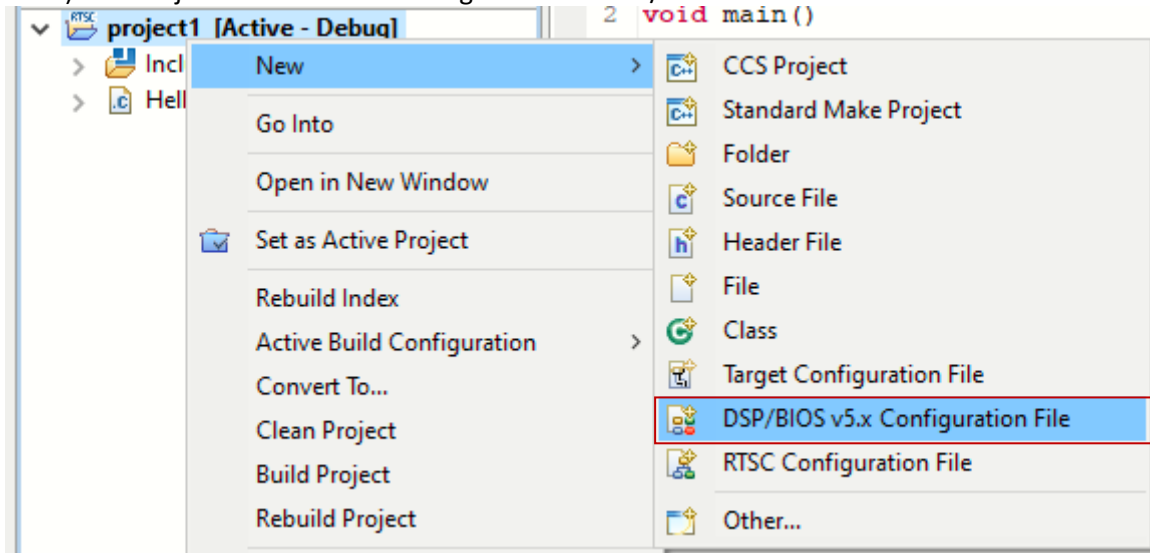
Il suffit maintenant de taper le programme souhaité. Pour ce Tp, on va juste afficher le message "**hello world**".



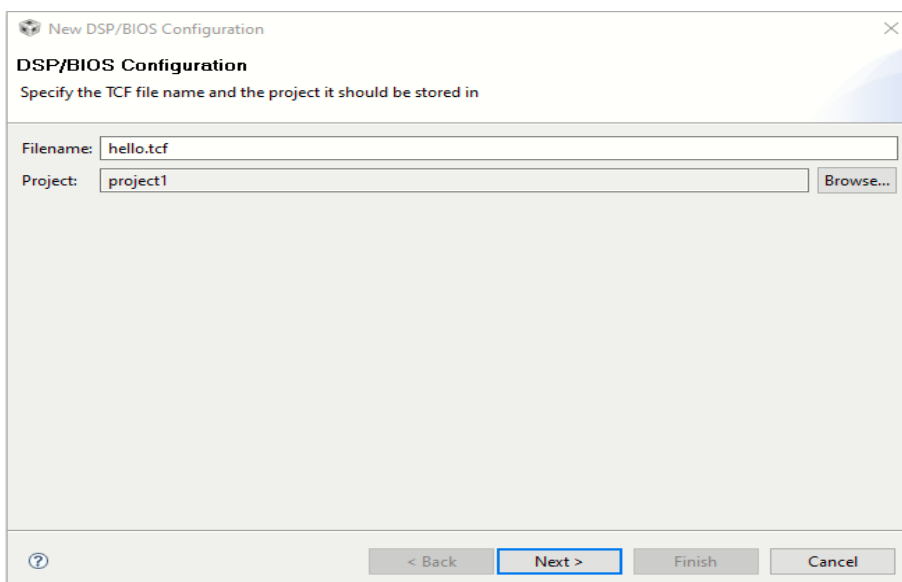
```
1 #include <stdio.h>
2 void main()
3 {
4     print("Hello world \n ");
5 }
6
```

1.3 Ajout des fichiers au projet :

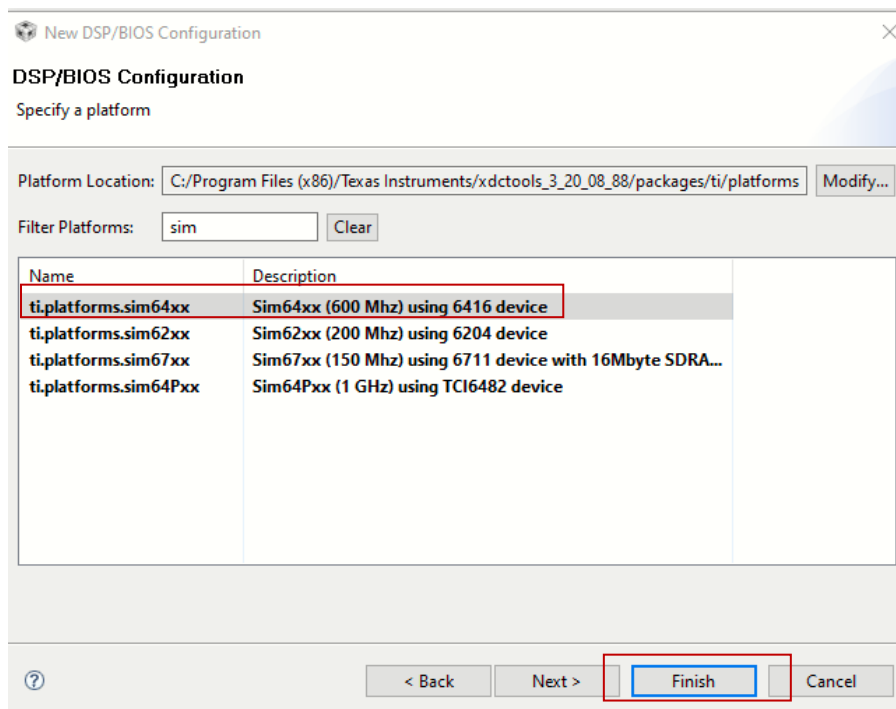
1) Pour ajouter le fichier de configuration de DSP/BIOS



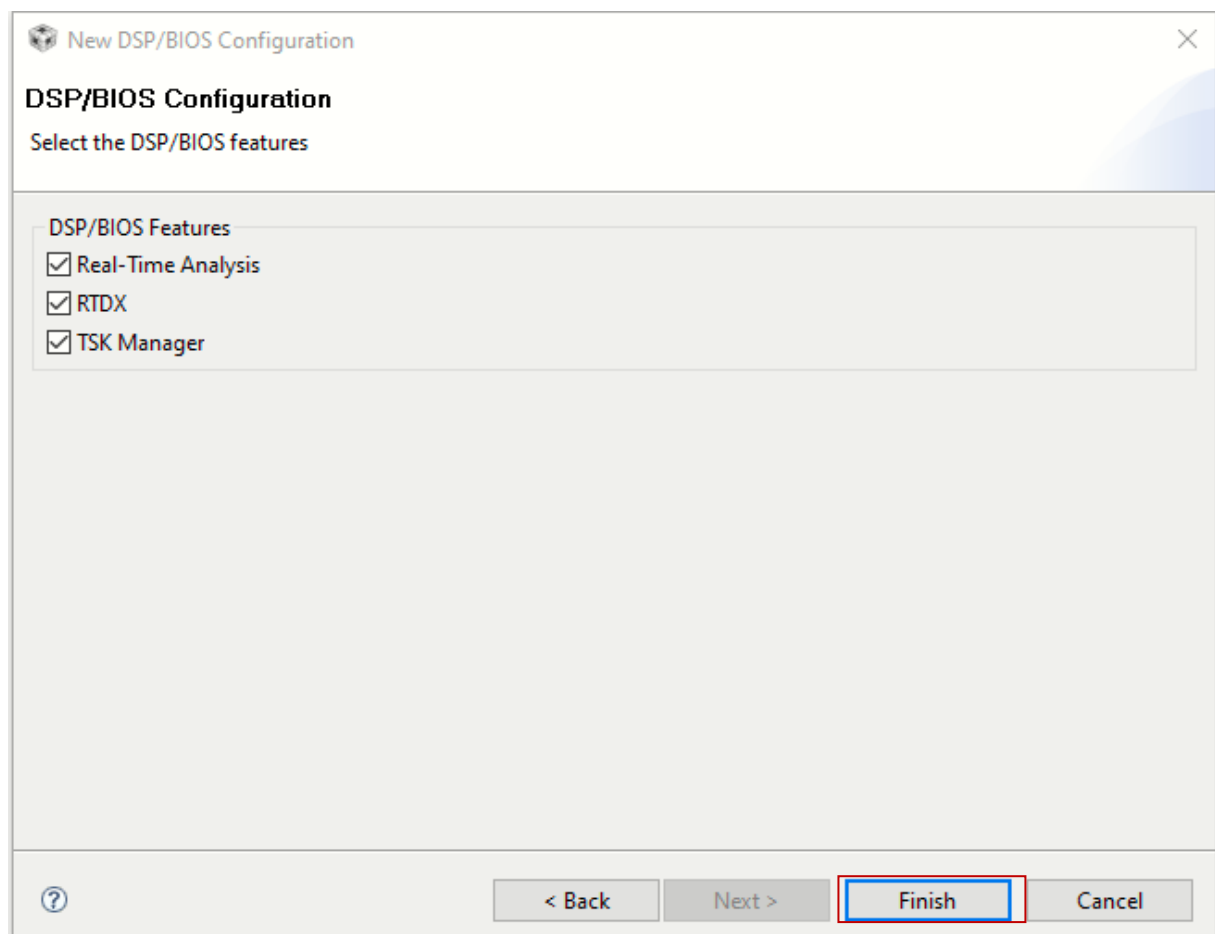
2) Donner un nom au fichier de configuration de DSP/BIOS, par exemple "hello.tcf" et cliquer sur Next



3) Spécifier la plateforme DSP à utiliser. Pour notre cas, choisir la plateforme **simulateur 6416**.



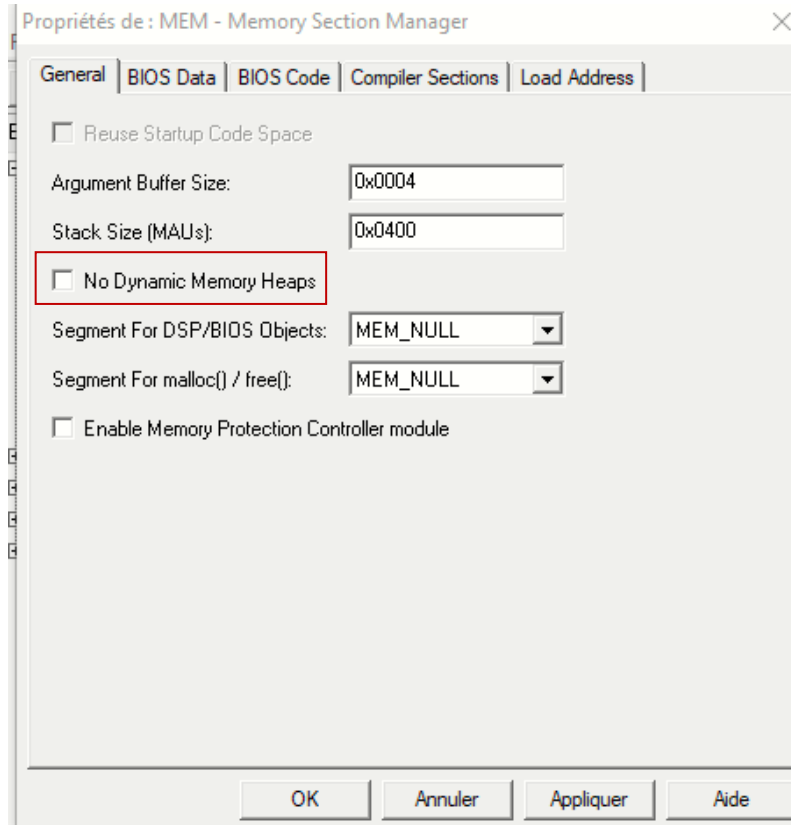
1) Vérifier que les paramètres de configuration sont tous cochés et cliquer sur **Finish**.



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

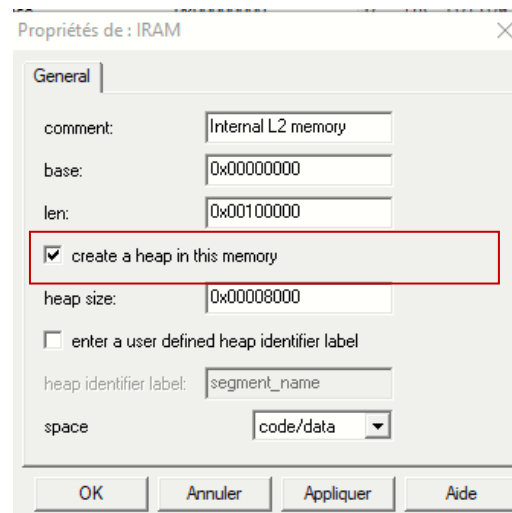
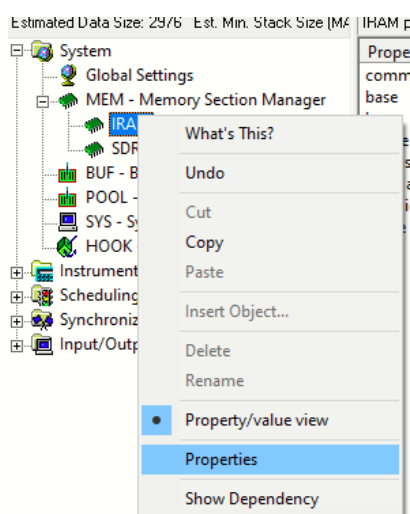
Ce fichier permet de faire une gestion des mémoires (création des **heaps** mémoires, définir la taille de la mémoire cache, etc ...), création des tasks, faire la synchronisation entre les cores en ajoutant des sémaphores, modifier les paramètres de RTDX (Real Time Data EXchange) tel que le RTDX mode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

5) Dans cette étape, on va configurer les différentes sections mémoire

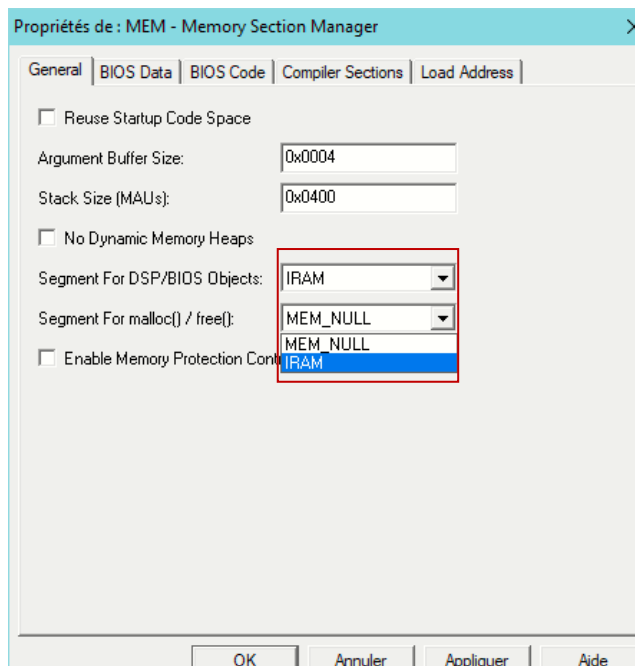


6) Pour choisir maintenant le type de mémoire où on va créer un heap mémoire (par exemple dans la mémoire interne de chaque DSP IRAM), il faut suivre les deux étapes suivantes :

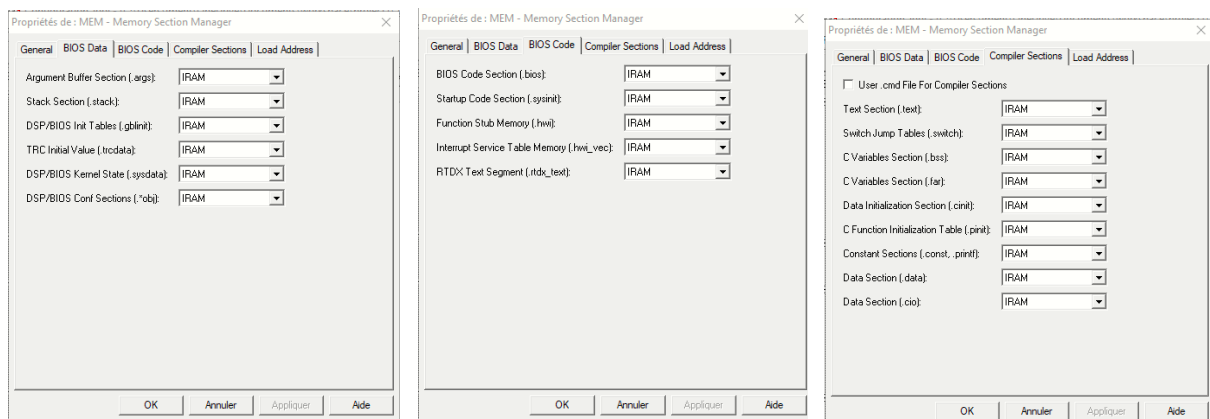
- Cliquer avec le bouton droit de la souris sur **IRAM Properties**
- Cocher la case **“Create a heap in this memory”** et cliquer sur **OK**.



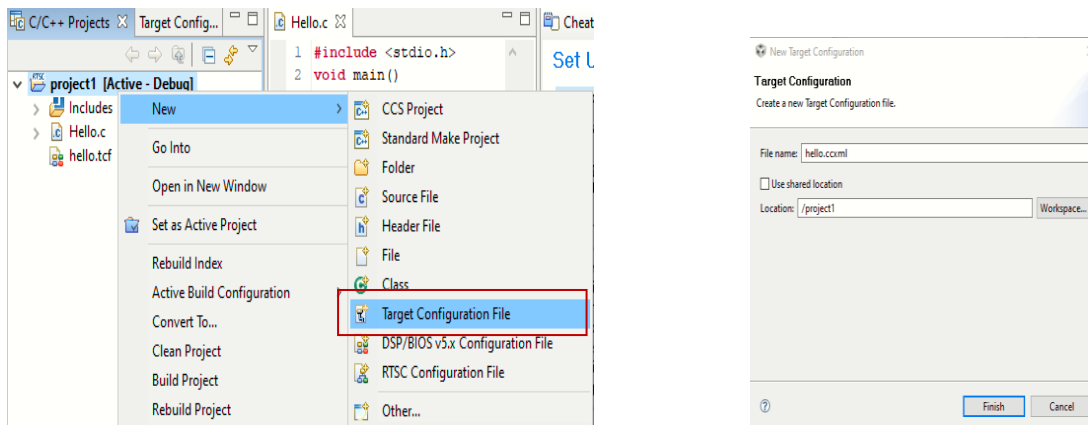
7) Pour configurer les différentes sections à créer, retourner au **MEM-Memory Section Manager Properties**.



8) 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**.



9) Ajouter un “**Target Configuration File**” :

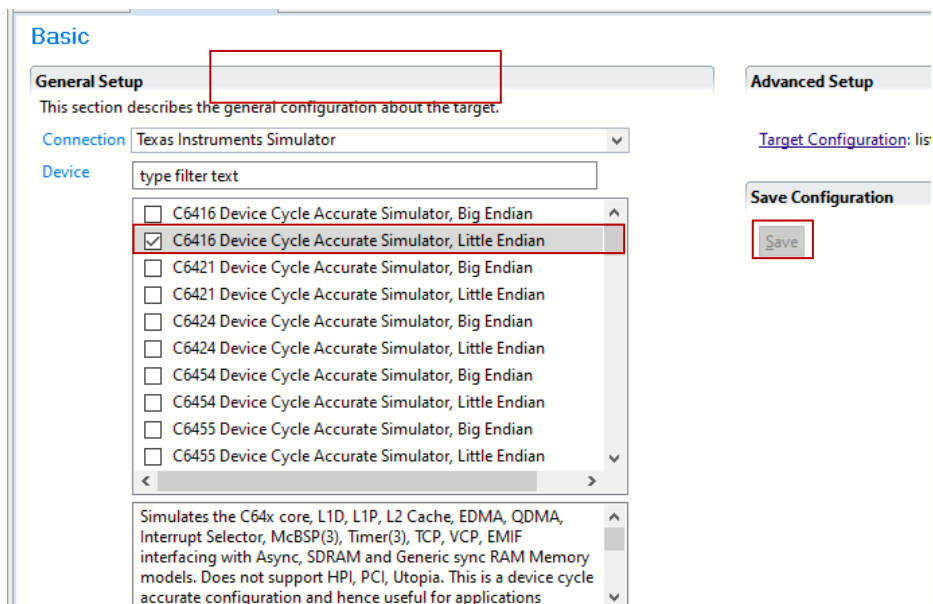


10) Il faut définir la cible de l'implantation. On peut choisir le nom du DSP utilisé et sélectionner un simulateur ou bien un émulateur.

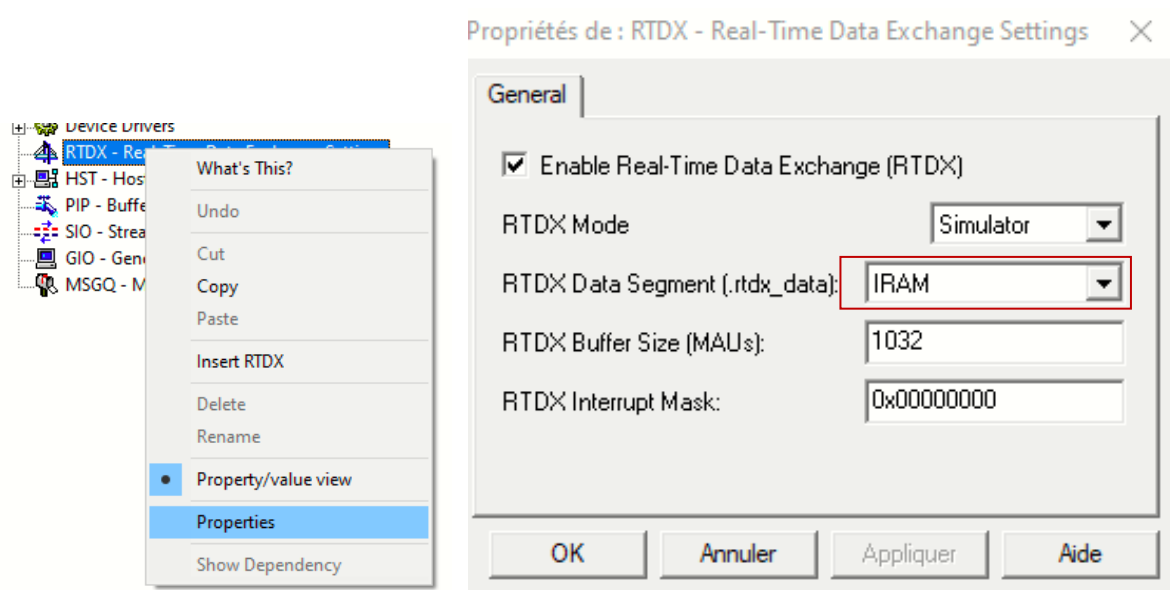
Dans le cas d'un simulateur, il suffit de choisir la configuration suivante :

Connection : **Texas Instruments Simulator**

Device : **C6416 Device Cycle Accurate Simulator, Little Endian**



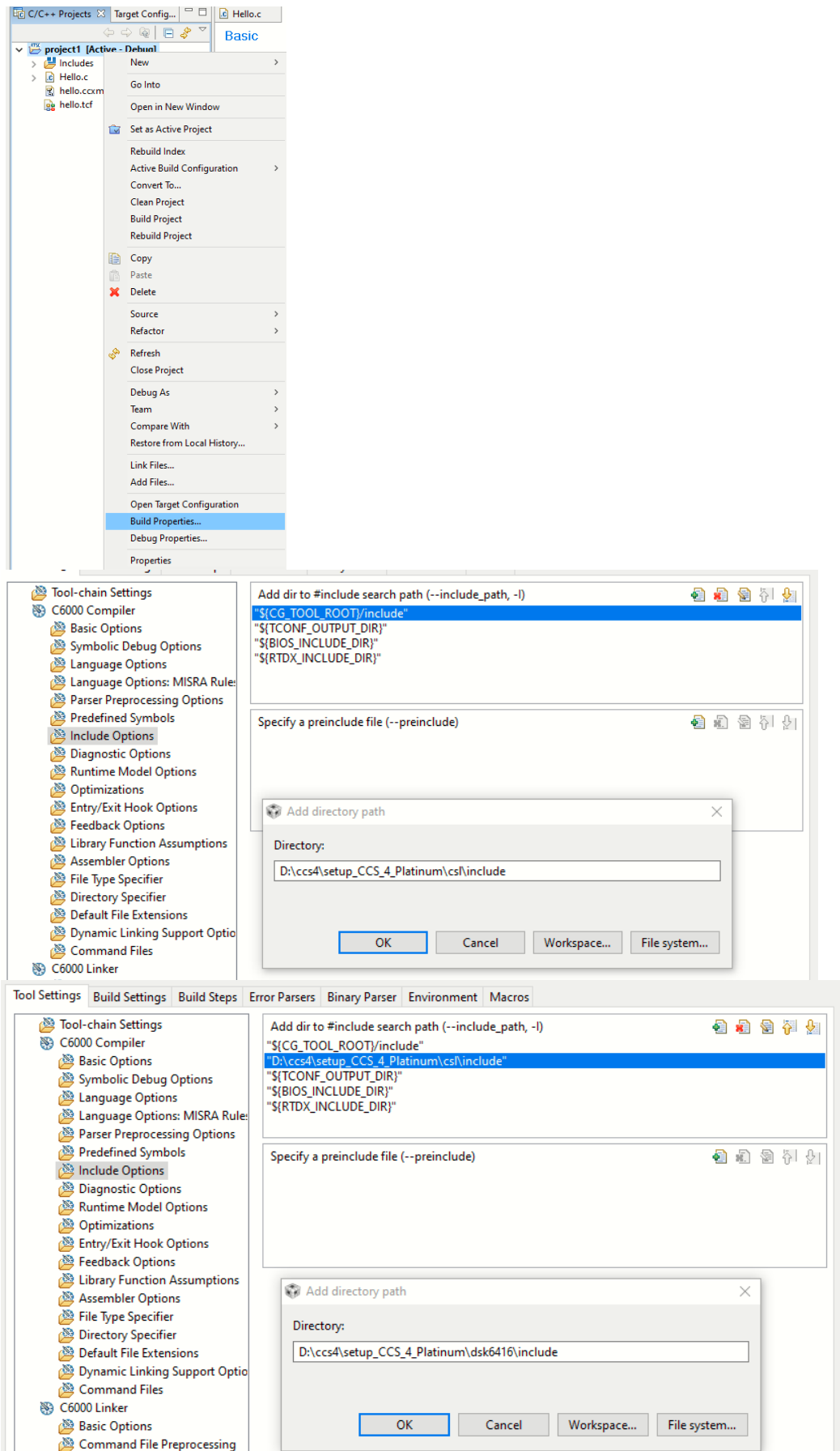
11) Pour choisir de travailler avec le simulateur ou sur la plateforme, il faut définir le mode **RTDX (Real Time Data EXchange)** correspondant.



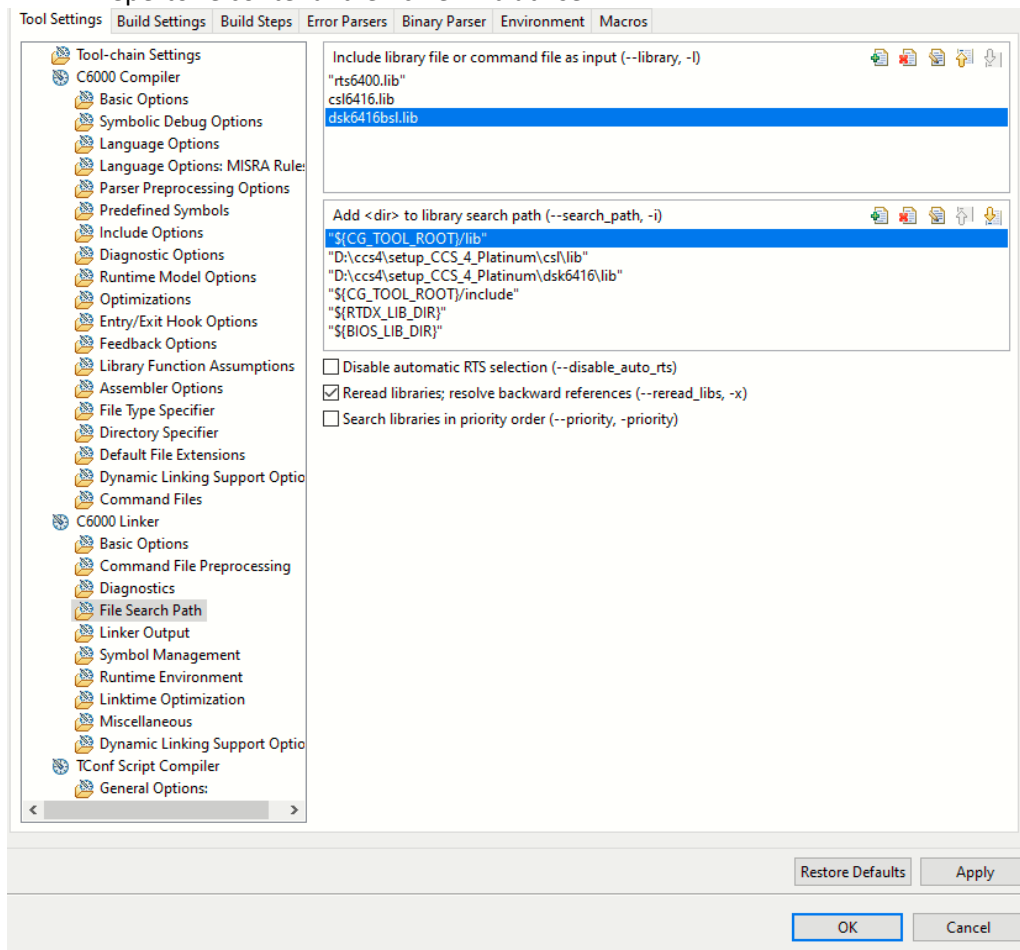
2- Compilation du projet :

- 1) 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 :

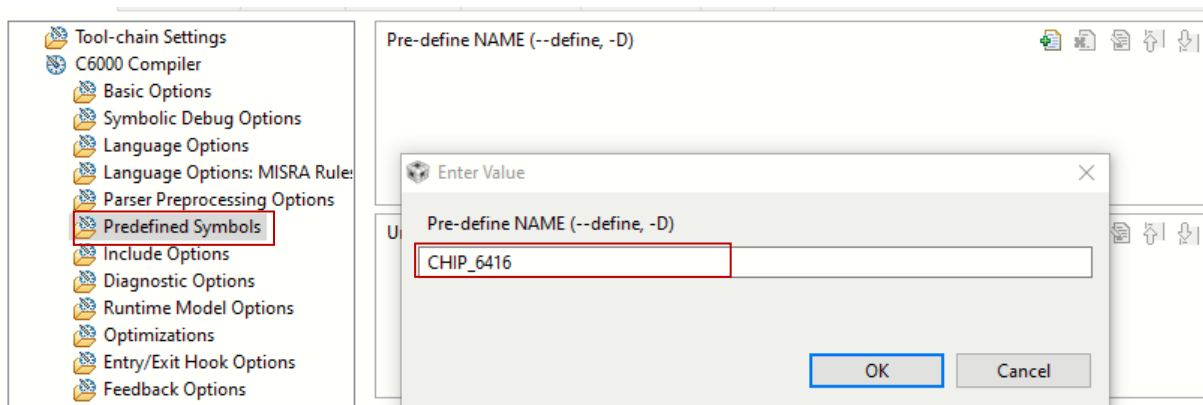




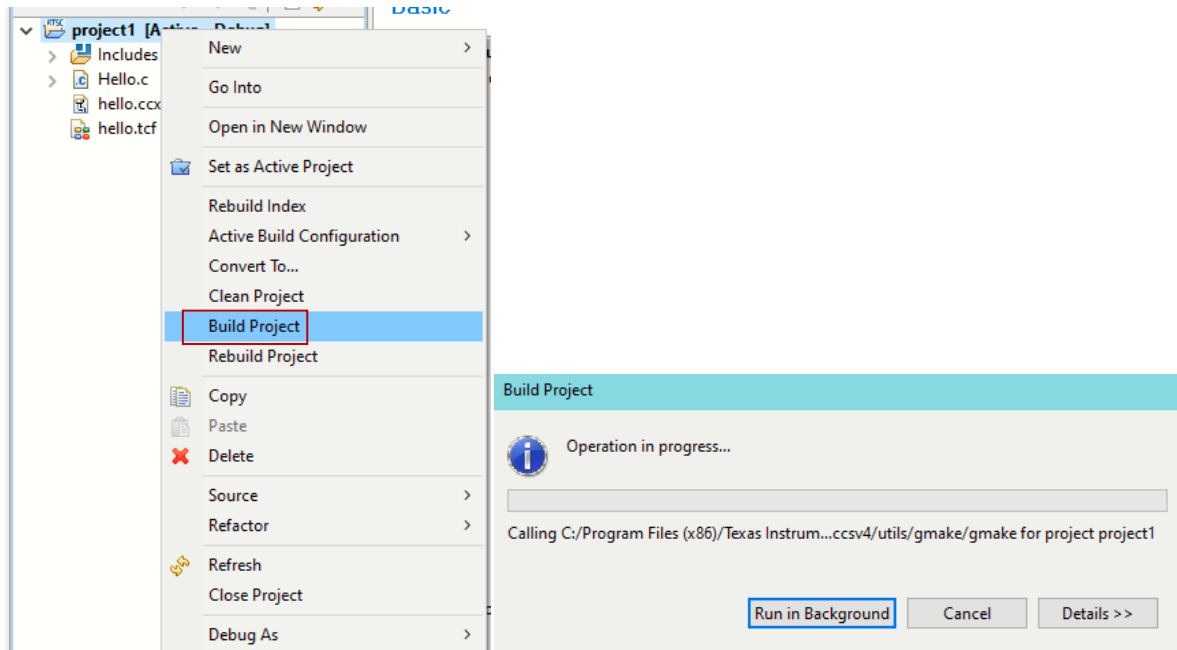
Même chose pour les bibliothèques “.lib” utilisées. Sélectionner **FileSearch Path** dans la fenêtre de **Build Properties** et ajouter le nom de la bibliothèque et donner le chemin du répertoire contenant le fichier .lib utilisé



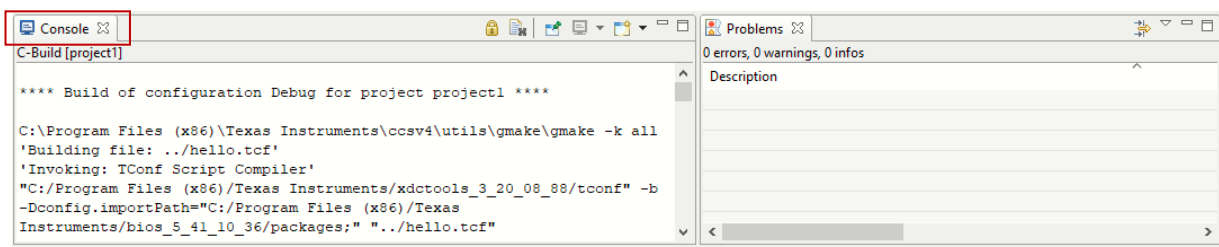
On n’oublie pas d ajouter le **predefined symbols** **CHIP_6416** :



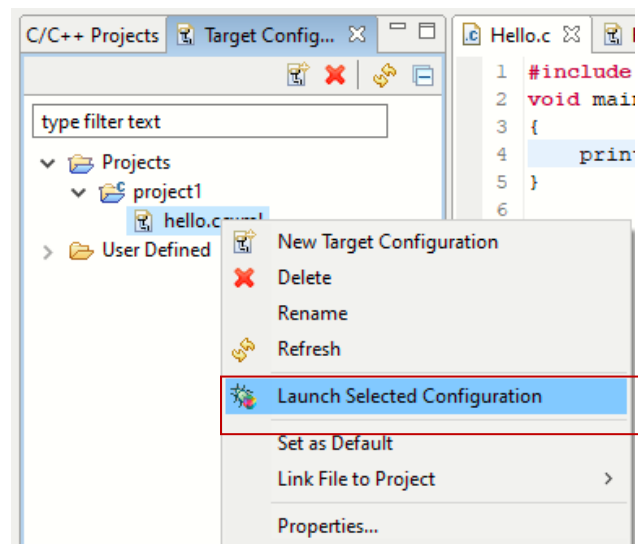
- 1) Pour compiler le projet il suffit de Cliquer par le bouton droit sur le nom du projet et sélectionner “**Build Project**” :



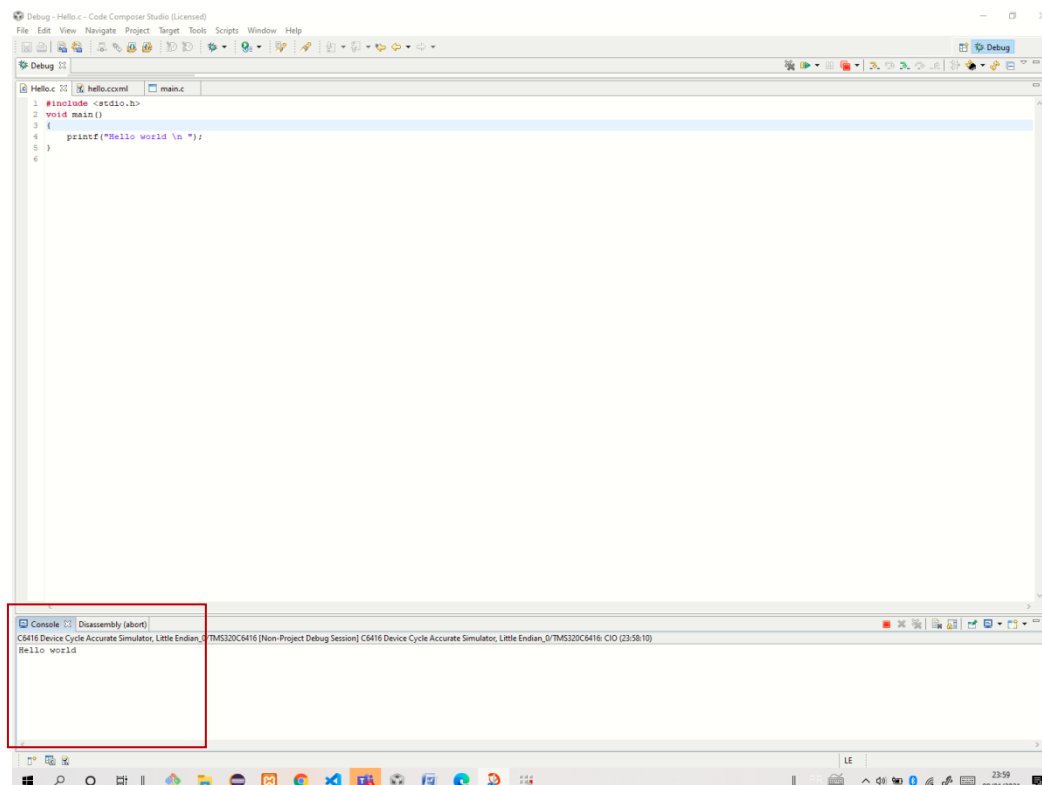
3) Si la compilation se termine avec succès (pas d'erreurs), un fichier exécutable "Project1.out" sera généré.



4) 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** :

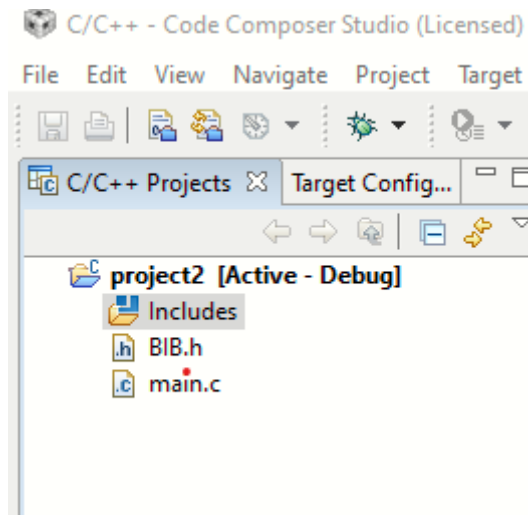


TP 2 : Optimisation :

1. Création d'un projet :

On répète les mêmes étapes que le TP pour créer un projet : project2

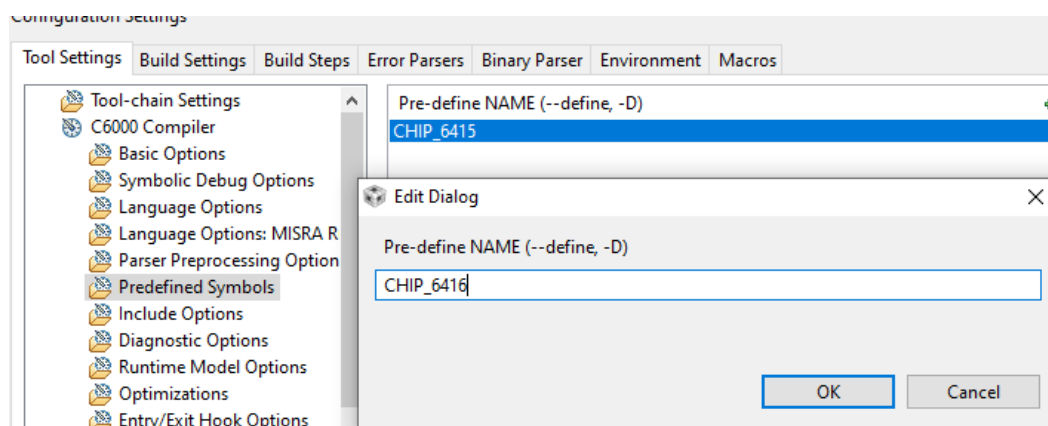
On fait inclure les deux fichiers **bib.h** et **main.c** dans notre projet :



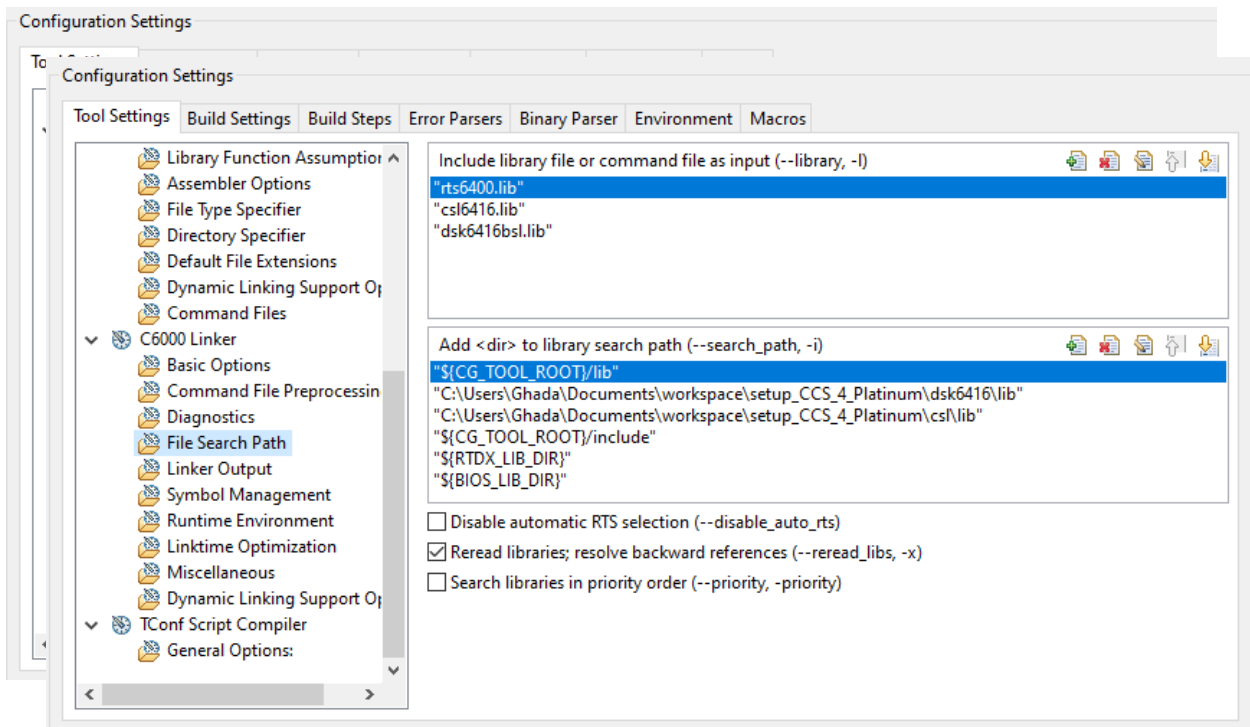
2. Compilation d'un projet :

Il faut suivre ces étapes pour ajouter les chemins de biblio correctement :

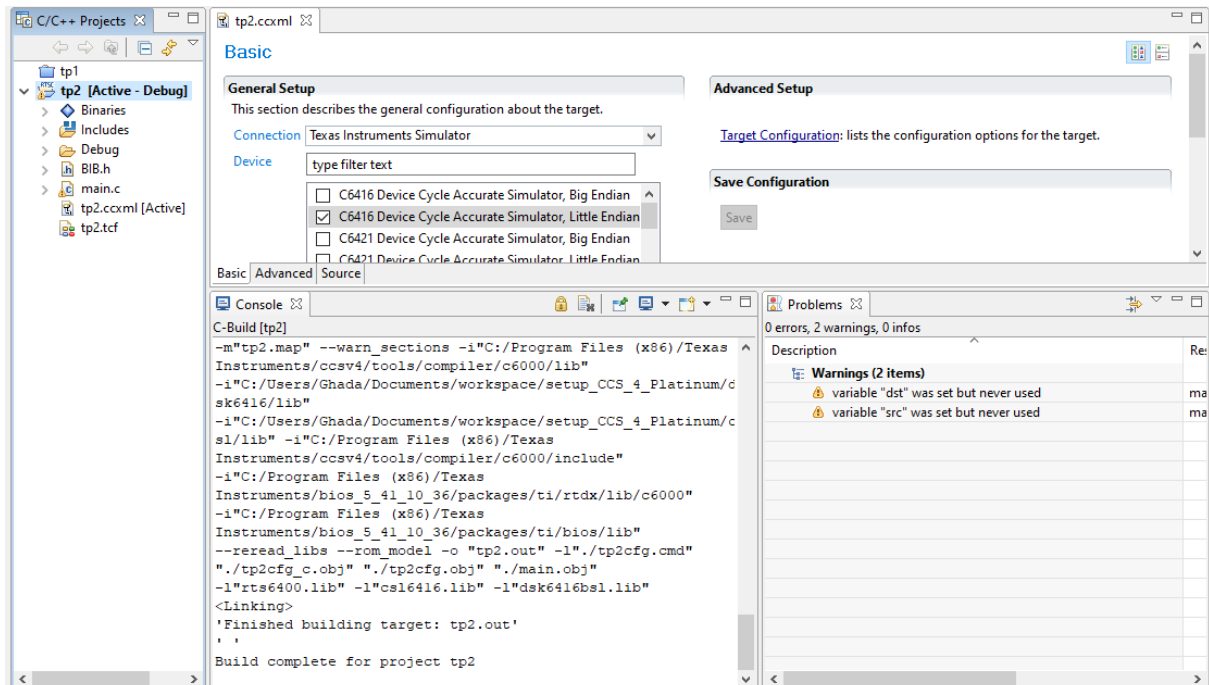
On remplit **Predefined Symbols** avec **CHIP_6416** comme suivant :



Include Options



On compile le projet comme la compilation de tp1 :



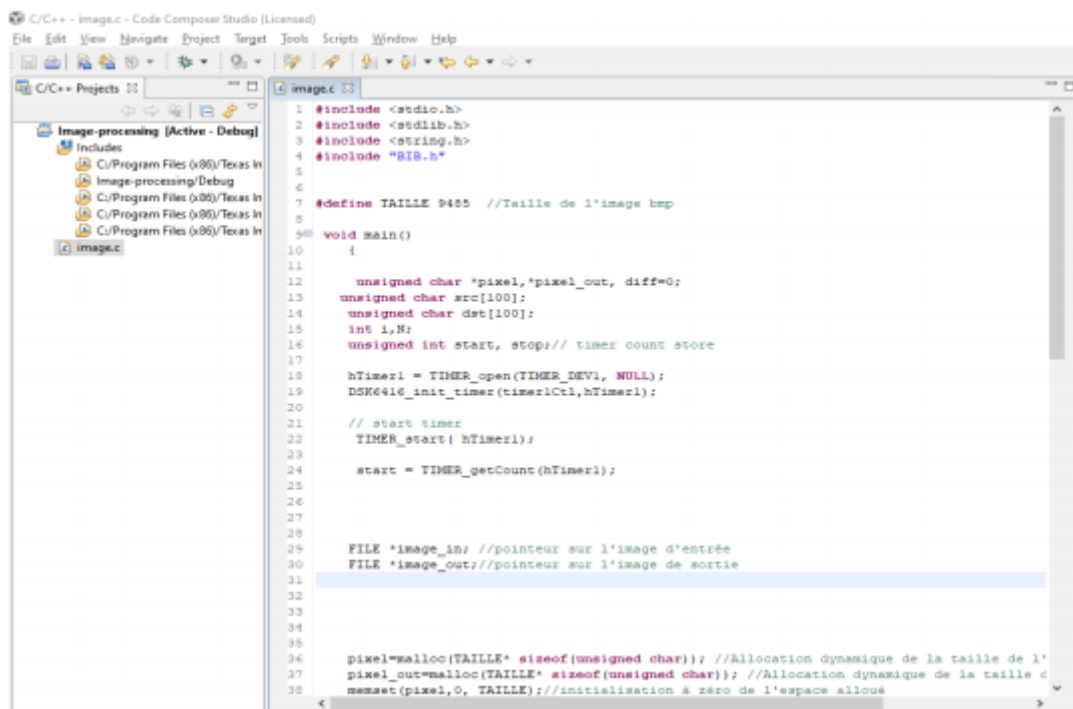
TP 3 : Image Processing :

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

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

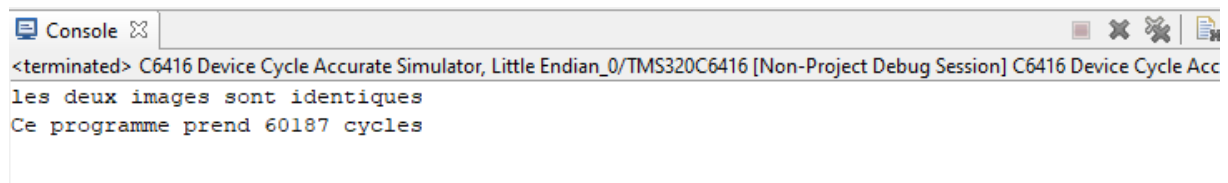
On va rectifier notre code source initial comme suit :



```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include "BIS.h"
5
6
7 #define TAILLE 9455 //Taille de l'image bmp
8
9
10 void main()
11 {
12     unsigned char *pixel,*pixel_out, diff=0;
13     unsigned char src[100];
14     unsigned char dst[100];
15     int l,W;
16     unsigned int start, stop;// timer count store
17
18     hTimer1 = TIMER_open(TIMER_DEV1, NULL);
19     DSK6416_init_timer(timer1Ctrl,hTimer1);
20
21     // start timer
22     TIMER_start(hTimer1);
23
24     start = TIMER_getCount(hTimer1);
25
26
27
28
29     FILE *image_in; //pointeur sur l'image d'entrée
30     FILE *image_out;//pointeur sur l'image de sortie
31
32
33
34
35
36     pixel=malloc(TAILLE* sizeof(unsigned char)); //Allocation dynamique de la taille de l'
37     pixel_out=malloc(TAILLE* sizeof(unsigned char)); //Allocation dynamique de la taille d
38     memset(pixel,0, TAILLE);//initialisation à zéro de l'espace alloué
  
```

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
  
```

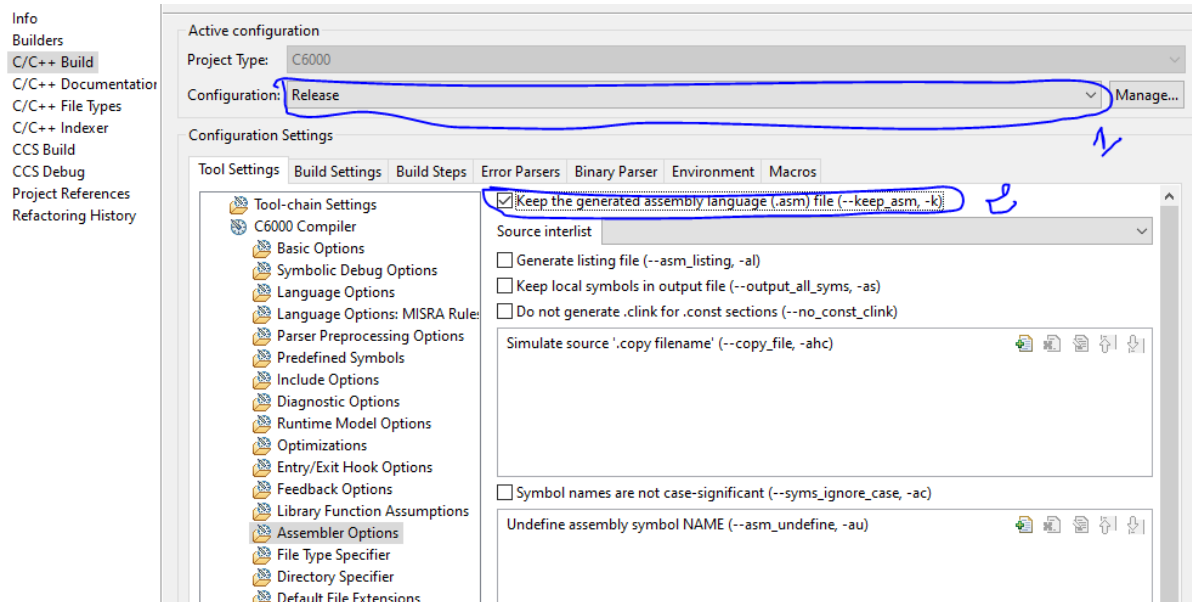
Comme l'indique au console notre programme a été exécuté avec succès et nous donnons une image identique à celle de l'entrée « input1 » .

Le programme fait 60187 cycles pour obtenir ce résultat.

On a un programme exécutable mais avec un grand nombre d'opération.

TP 4 : Image Processing + Optimisation :

Dans ce 3ème TP on doit faire un mélange entre les 2 anciens TP pour avoir un résultat plus optimisé et pour cela on doit changer la configuration vers « release » comme l'indique le schéma suivant



Après ce changement on trouve un nouveau fichier qui s'appelle « main.asm » avec quelques paramètres détaillés pour l'assembleur comme :

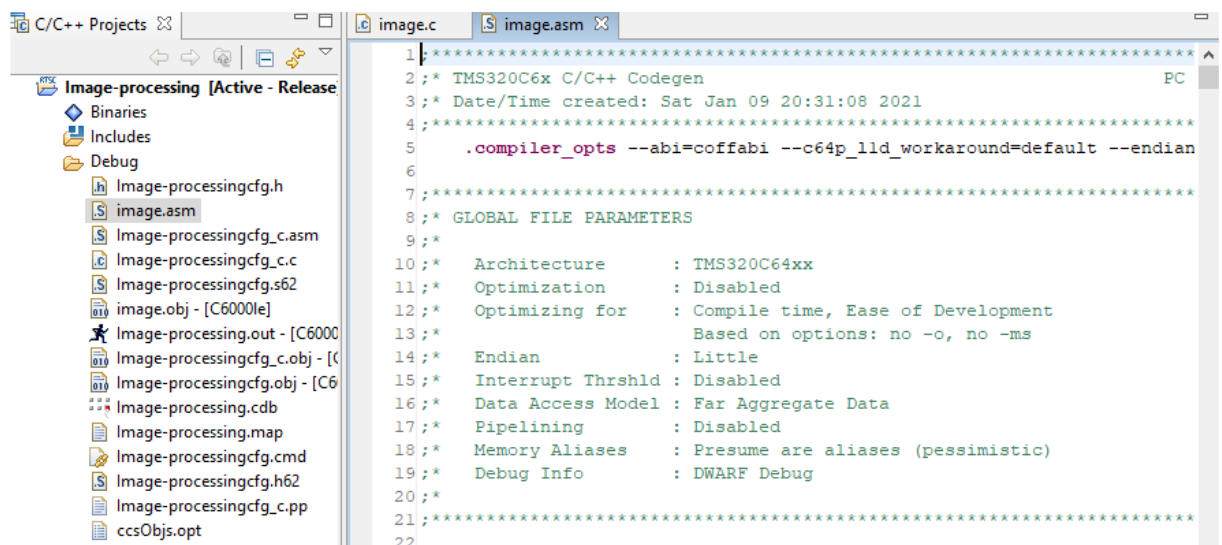
Architecture utilisée.

Optimisation.

Optimizing for.

Endian.

....



Le nombre de cycle est réduit avec Release grâce a Pipelining et Optimizing

Après l'exécution avec ces nouveaux paramètres on trouve que le nombre diminue de 60187 à 9608.

Avec un simple calcul on trouve que on a optimisé 84% en mode release. Dans ce mode les exécutions seront faire en mode parallèle.

Conclusion :

Dans le 1^{er} TP on a commencé par les paramètres et configurations nécessaires pour commencer notre travail avec CCS v4.

On a fait un programme d'optimisation pour minimiser le nombre de cycle effectué par un programme normal et on l'a comparé avec un autre programme optimisé.

Image Processing un programme qui est chargé par des opérations élevées, on avoir un grand nombre des cycles effectuées pendant l'exécution.

Finalement on a minimiser le programme image processing avec le changement de mode de configuration vers « Release » et on a comparé entre les deux nombre de cycle effectuée par le programme.