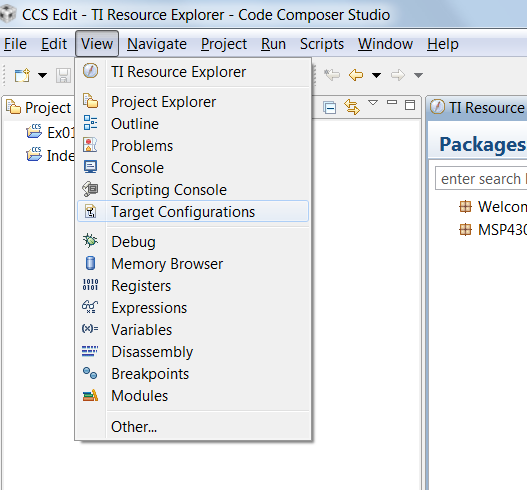
Laboratoire 2 :   
Introduction à la carte DSP et à sa programmation

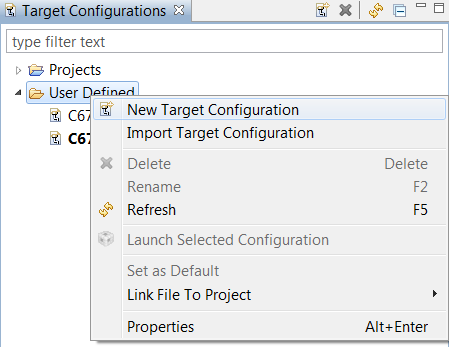
Le but de ce laboratoire est de se familiariser avec l’utilisation du logiciel Code Composer Studio et à l’utilisation de la carte TMS320C6713. Il fait suite au cours et permet de rentrer concrètement dans la programmation en lagunage C et assembleur. Le logiciel a été installé sur <http://processors.wiki.ti.com/index.php/Download_CCS> et il s’agit de la version .   
Nous commencerons par voir une brève introduction à la carte DSP su CCS, ensuite nous verrons la programmation en elle-même pour enfin étudier les performances des programmes.

1. **Introduction à la carte DSK TMS320C6713**

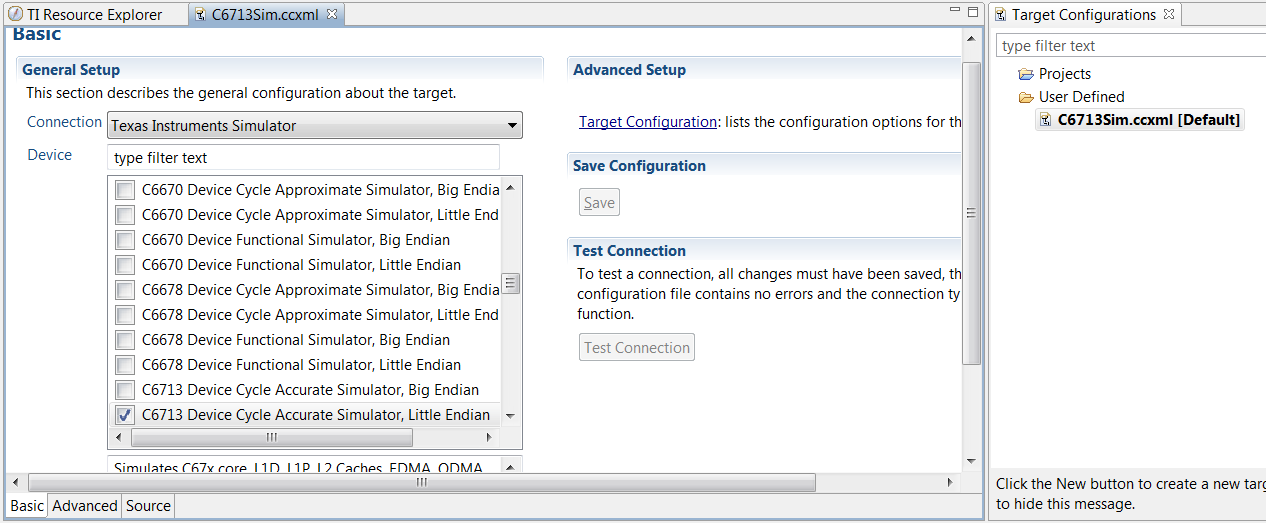
Avant de pouvoir se lancer dans la programmation, il est nécessaire de passer par toute sorte de configuration pour rendre la carte DSP compatible avec le logiciel. On commence donc par définir la carte utilisée dans  après avoir bien sûr lancer le logiciel...

1. 

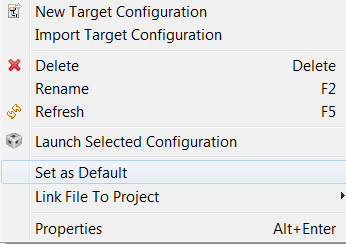
Sur la nouvelle fenêtre, on clique droit sur  et on crée avec le nom voulu.

1. 

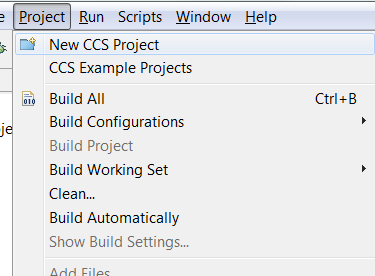
On double clique sur la nouvelle configuration que l’on vient de créer (toujours dans la fenêtre  ) et on sélectionne la bonne carte parmi la liste disponible. Sans oublier de .

1. 

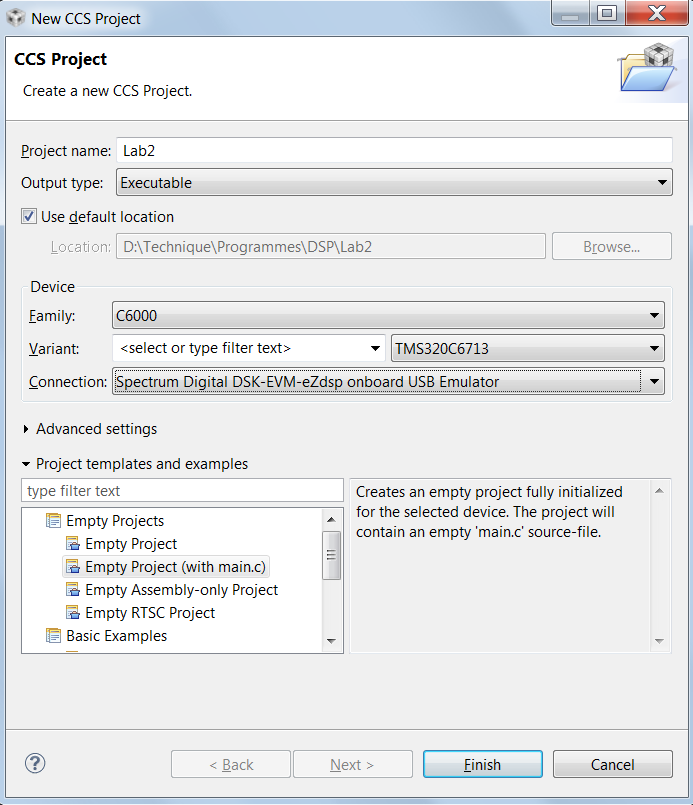
En faisant un clique gauche sur la configuration, on la met par défaut.

1. 

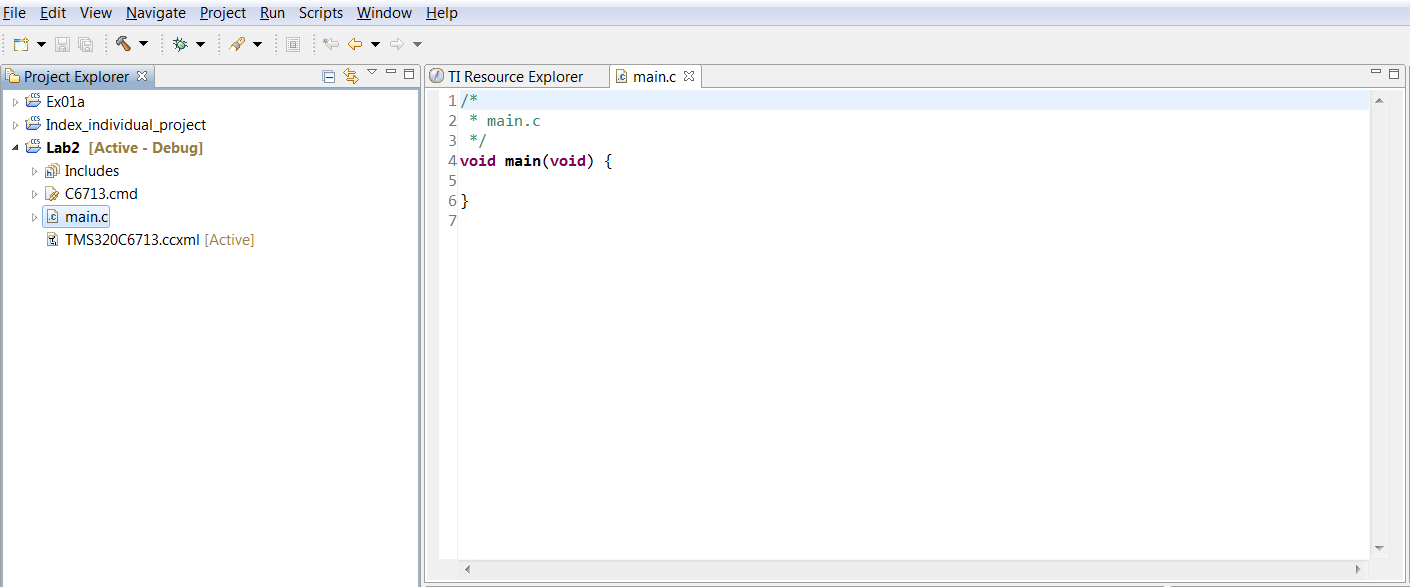
Après avoir défini la configuration, on peut créer un nouveau projet

1. 

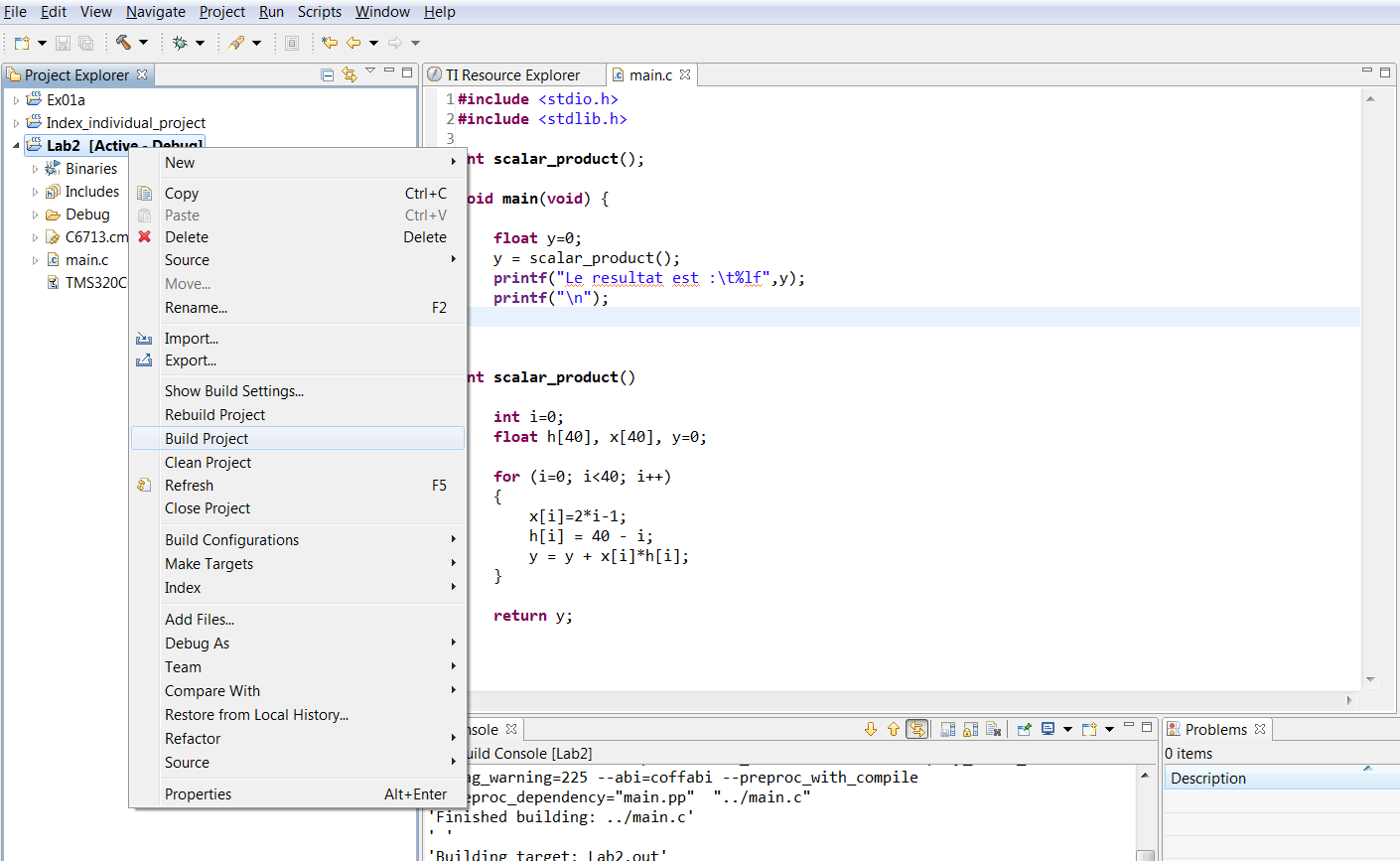
En rentrant les informations voulues, en laissant bien  et en choisissant un projet vide avec main.c

1. 

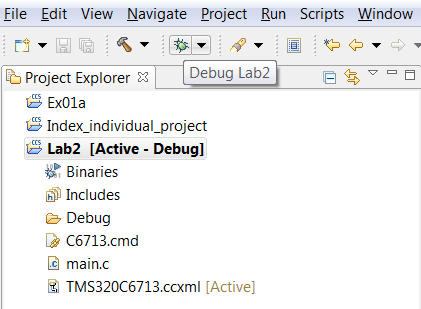
Le projet s’affiche alors dans la fenêtre 

1. 

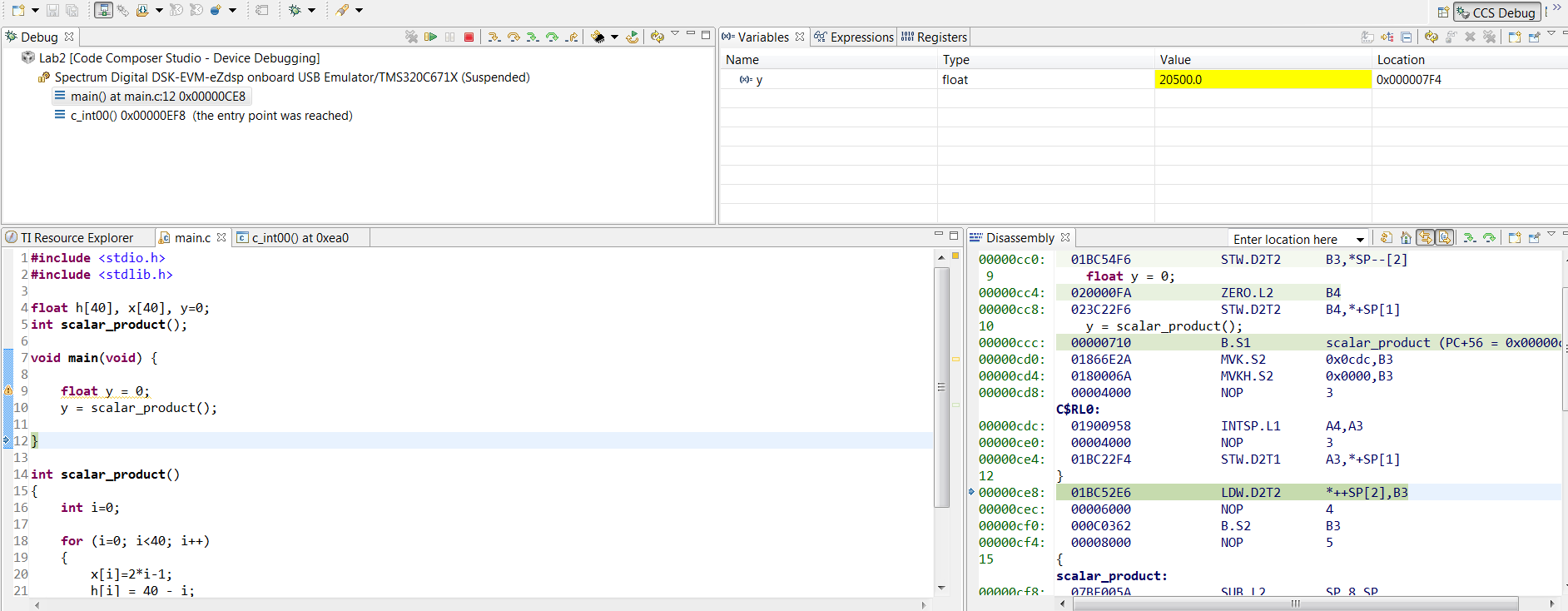
Il « suffit » ensuite de coder, et une fois fini on « build ».

1. 

Lorsqu’il n’y a plus d’erreur, on peut alors debugger notre code en cliquant sur l’icône correspondante,

1. 

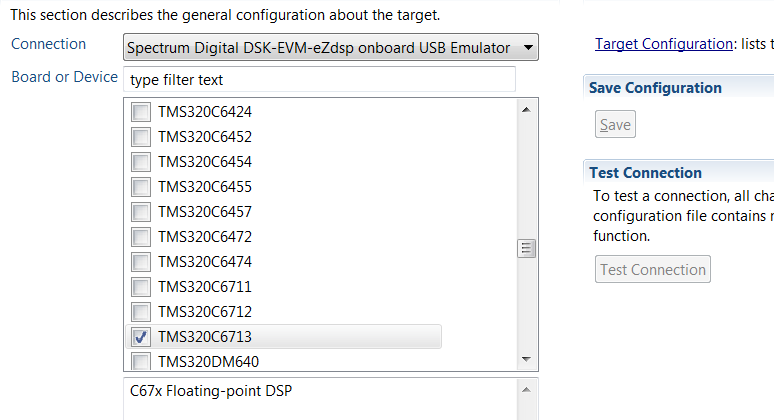
En déroulant le code on vérifier le résultat de retour dans la fenêtre 

1. 
2. **Mesure de performance**

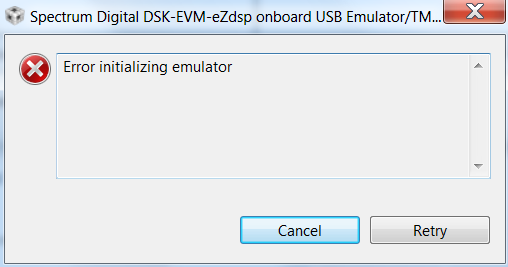
Après avoir débugger son programme, il convient de mesurer ses performances pour ensuite optimiser son code. Pour analyser les performances, CCS propose un profiler assez puissant, seulement il ne marche pas avec toutes les cartes (dont TMS320C6713). C’est pourquoi on créer un nouveau  qui va permettre de simuler une carte.

(Répéter I-2 à I-4)

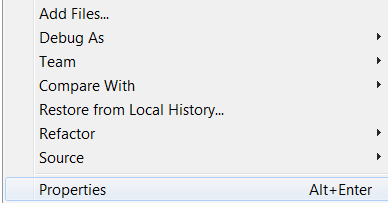
En utilisant la configuration voulue.



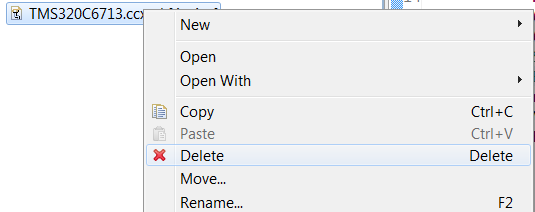
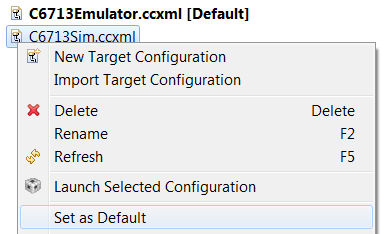
Si cela ne marche pas toujours pas lorsque l’on se met en mode debug



Alors il est nécessaire de configurer manuellement en cliquant gauche sur notre projet et en cliquant sur 



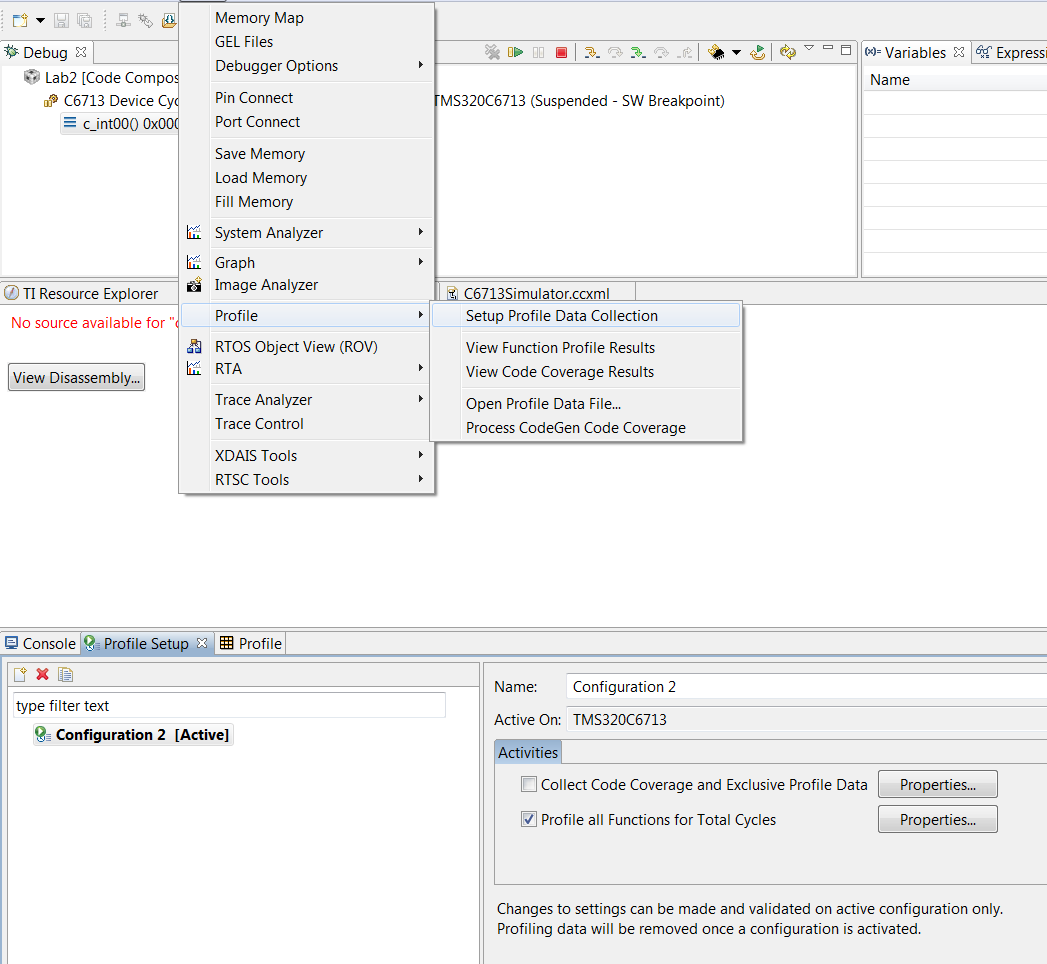
On supprime le fichier suivant et on remet l’ancienne  par défaut

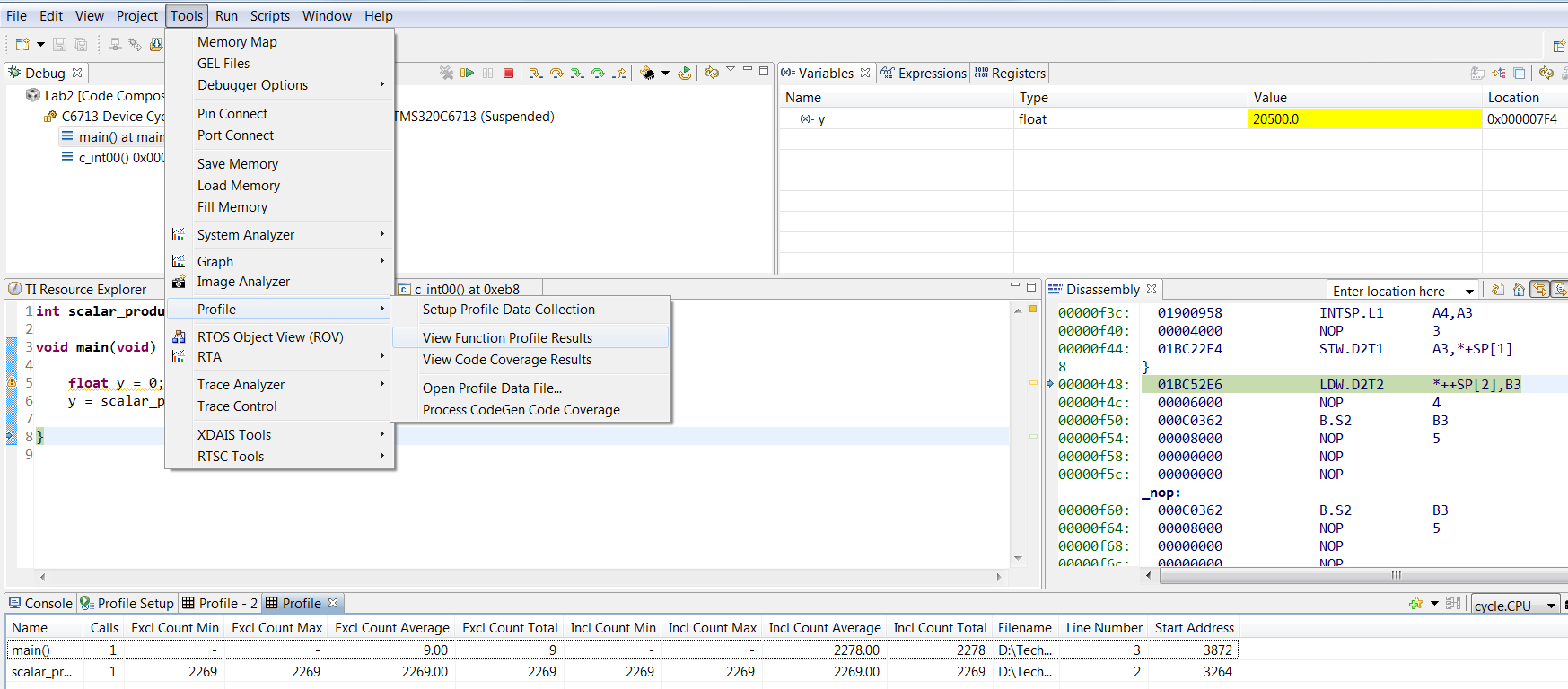
On peut alors enfin lancer le mode debug.

(Répeter I-8 à I-10)

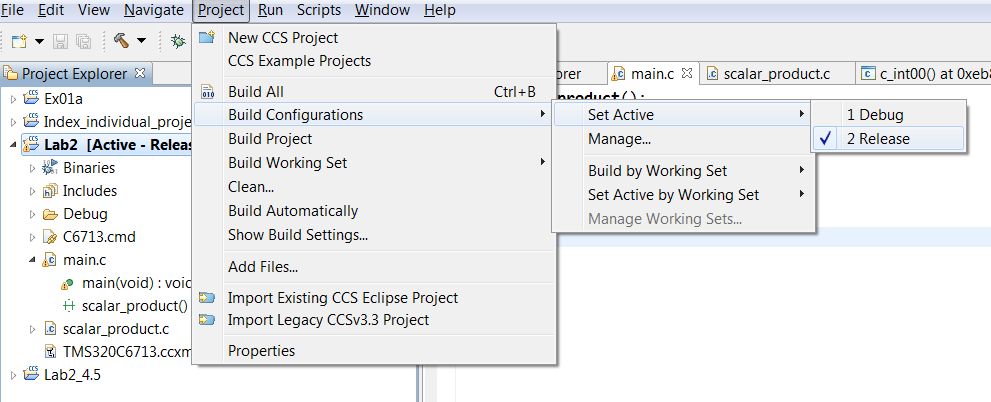
On définie les paramètres de notre profiler sur l’onglet , sans oublier de  et  la configuration.

1. 

On regarde les résultats du profiler en lançant notre programme 2 fois après avoir cliqué sur .

1. 

Lorsque l’on a étudié notre programme en mode « Debug », il convient de le tester de la même manière en mode « Release » cette fois-ci. C’est-à-dire que le compilateur va optimiser le temps de calcul. On commence donc commencer par changer la configuration du build,

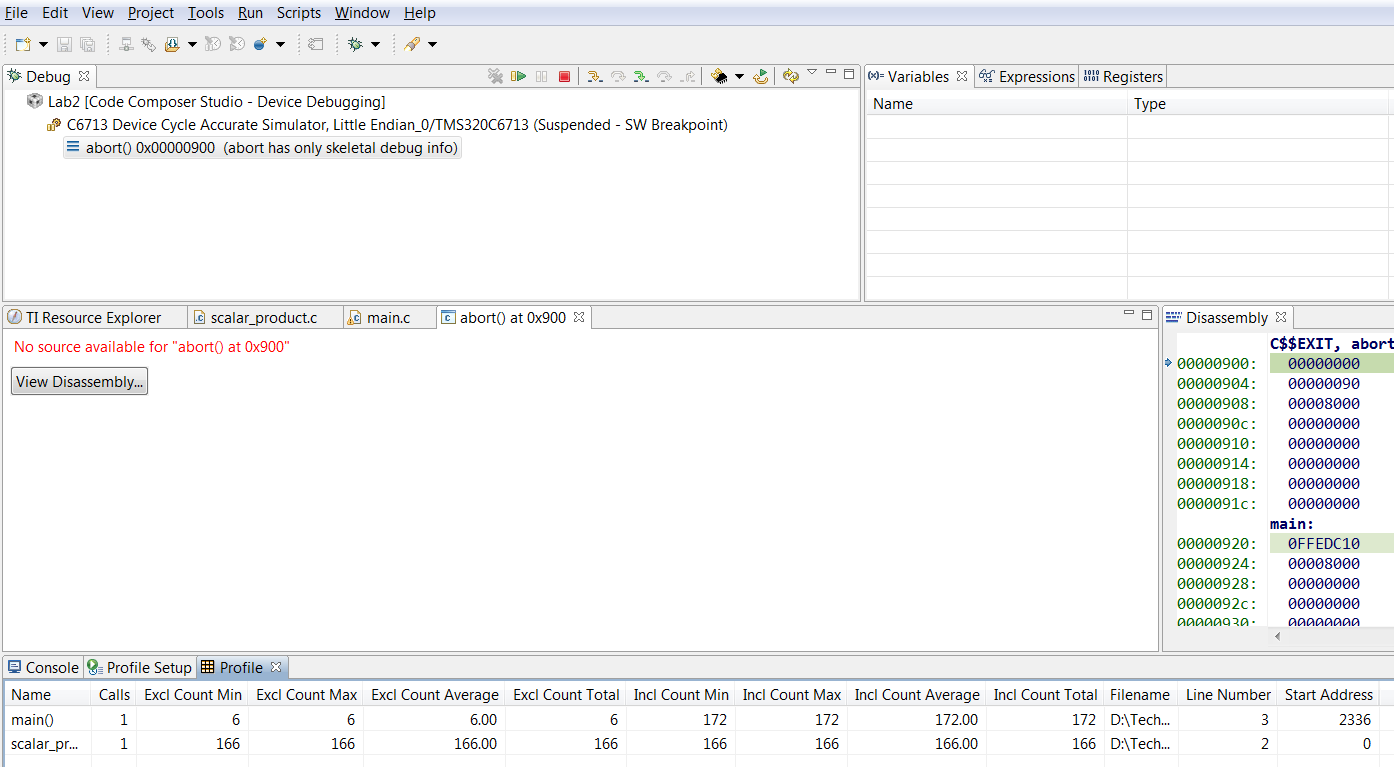
1. 

On rebuild et on lance le programme (clique sur debug mais ce n’est pas du debug)

(Répéter I-8 et I-9)

On recommence les étapes pour le profiler

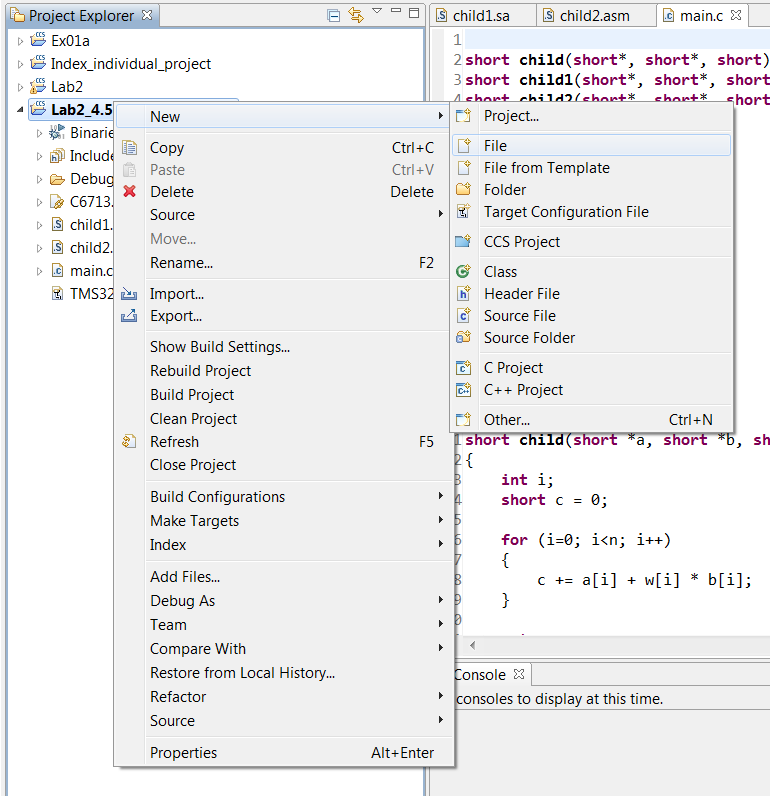
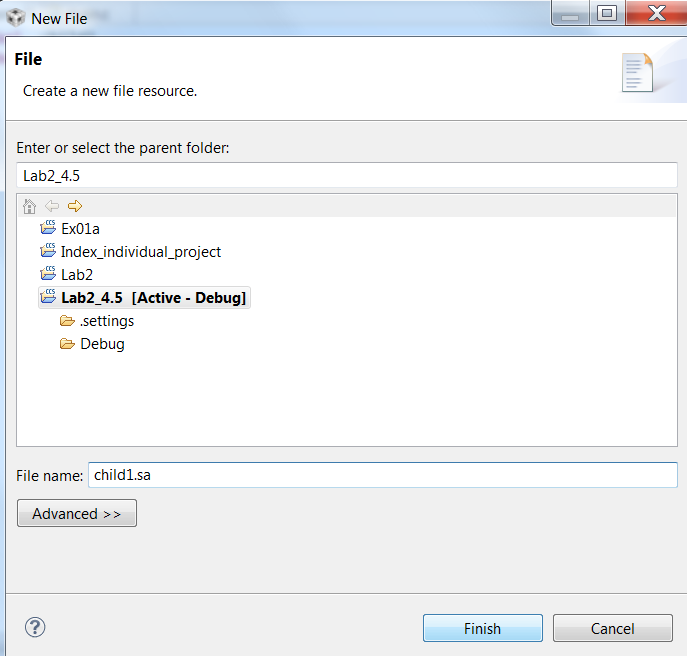
(Répéter II-2)



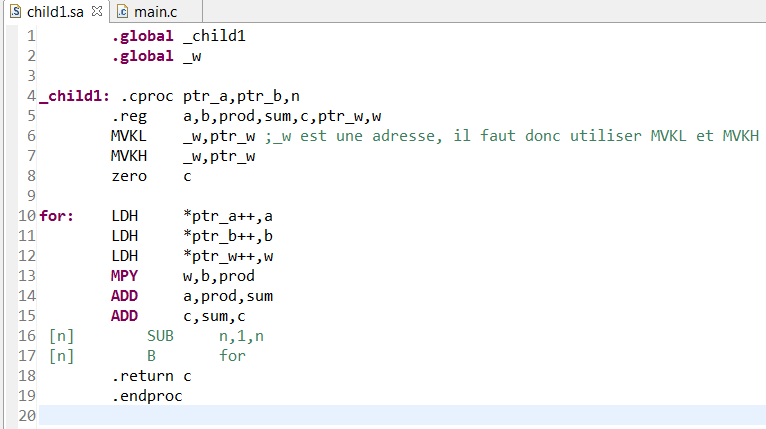
On voit que l’on a de bien meilleures performances avec le mode release. Par contre en mode release on a plus accès aux informations sur nos variables, c’est pourquoi il convient bien évidemment de débugger son programme avant de passer en mode « Release ».

1. **Programmation en assembleur**

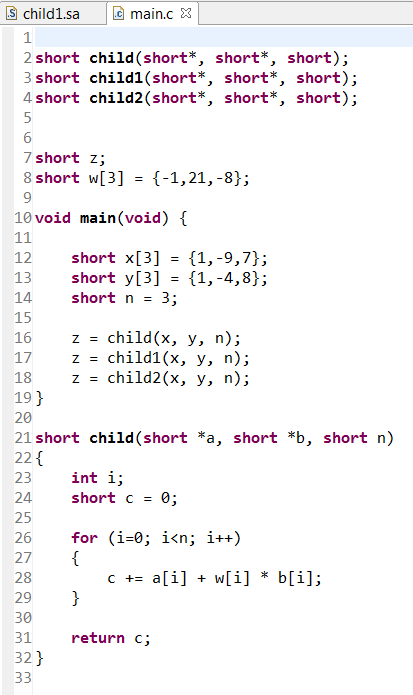
La programmation en assembleur nécessite beaucoup plus de rigueur qu’en langage C, le débogage devient encore plus important en assembleur. Après avoir écrit le programme en C, j’ai commencé par faire l’assembleur linéaire. Pour créer un fichier en assembleur linéaire il faut créer un nouveau fichier en précisant l’extension « .sa ».

1.  

Contrairement à l’assembleur optimisé, ici on ne s’occupe pas des registres, des unités de calculs, des délais des instructions et de la parrallélisation).



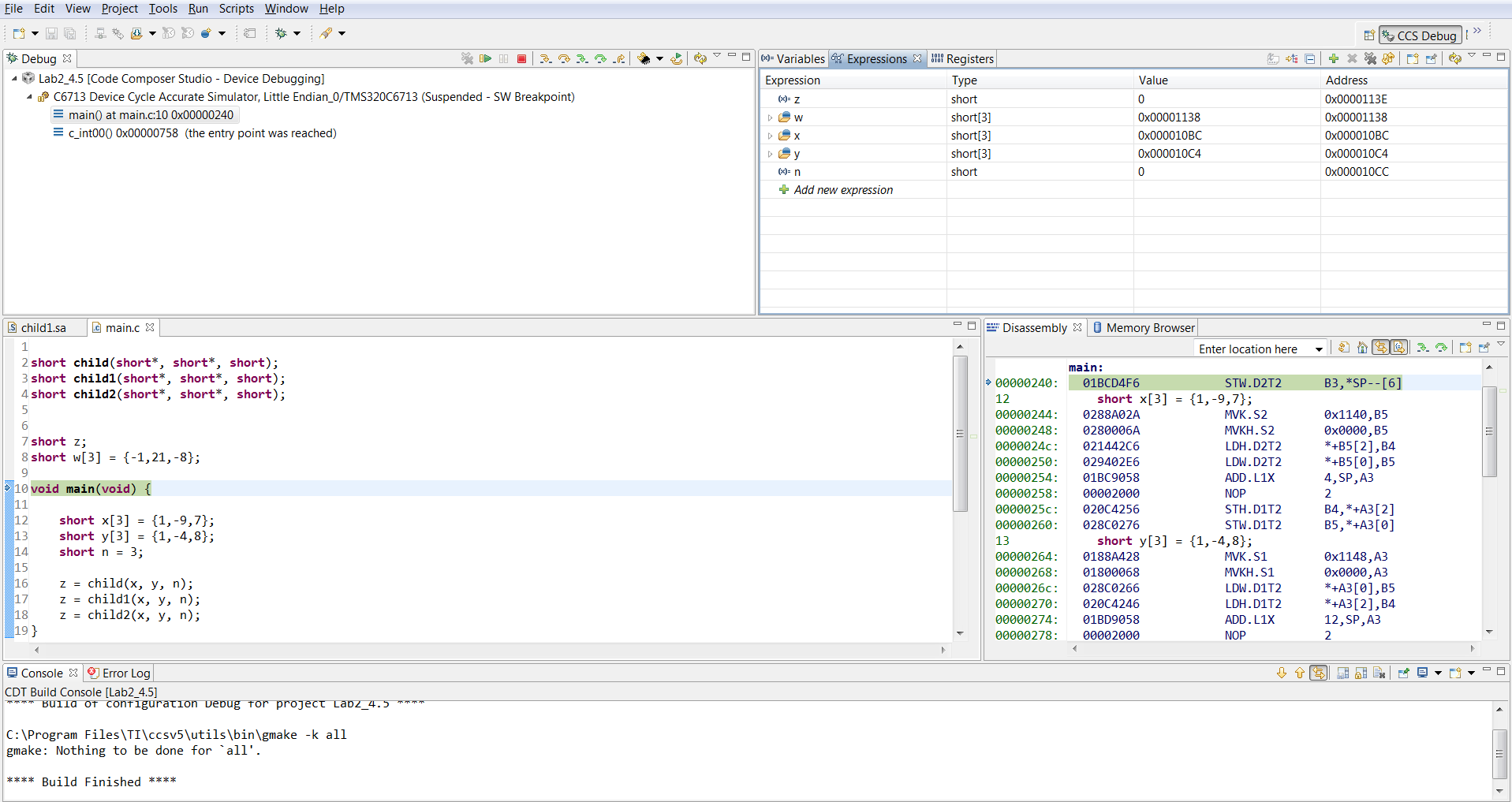
Il ne faut pas oublier de définir la fonction dans le fichier en C.



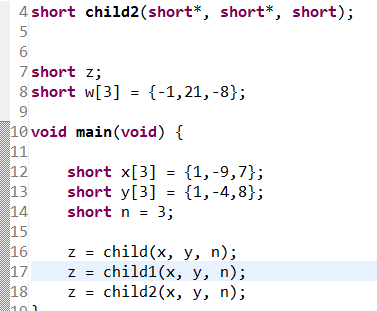
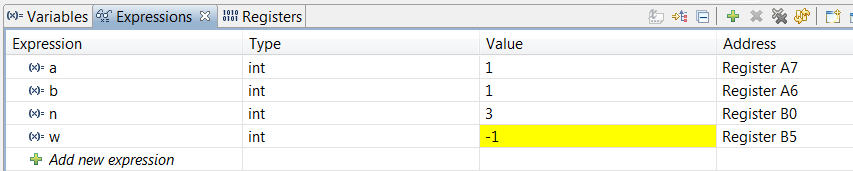
On accède au mode debug,

(Répéter I-8 et I-9)

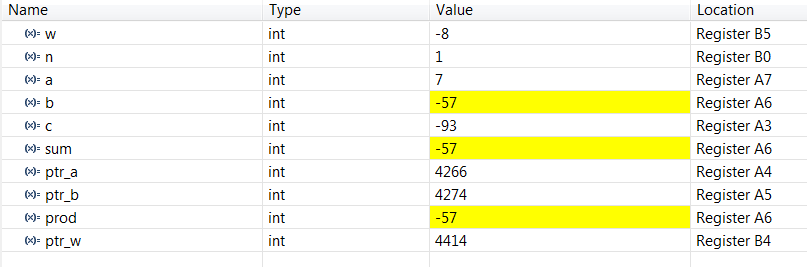
On a accès à plusieurs informations que l’on peut choisir dans l’onglet Window. Pour lancer le programme pas à pas on peut appuyer sur F5 (et dans de cas on rentre dans les fonctions), en utilisant F6 on se déplace aussi pas à pas mais seulement dans le main (sans rentrer dans les fonctions).



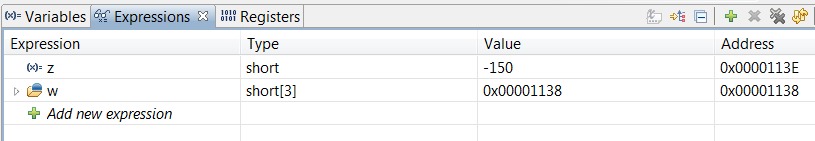
Pour débugger, il vaut mieux commencer par voir si l’on obtient bien les bonnes valeurs des variables locales et globales, définies dans le C.



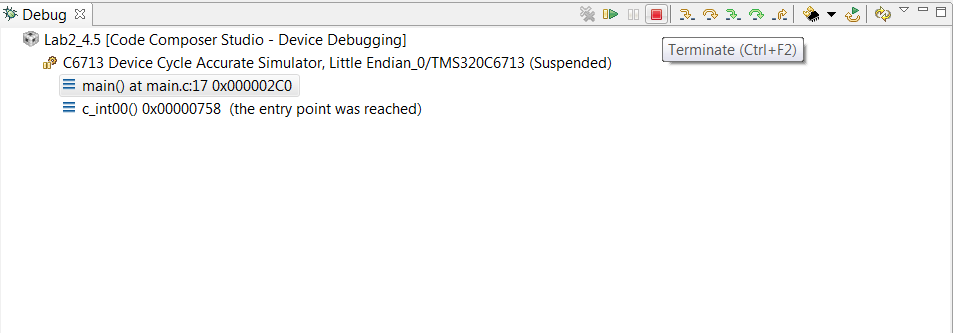
On peut remarquer que le compilateur choisit de rassembler les variables prod, sum et b par exemple car selon lui il est plus efficace de travailler comme cela



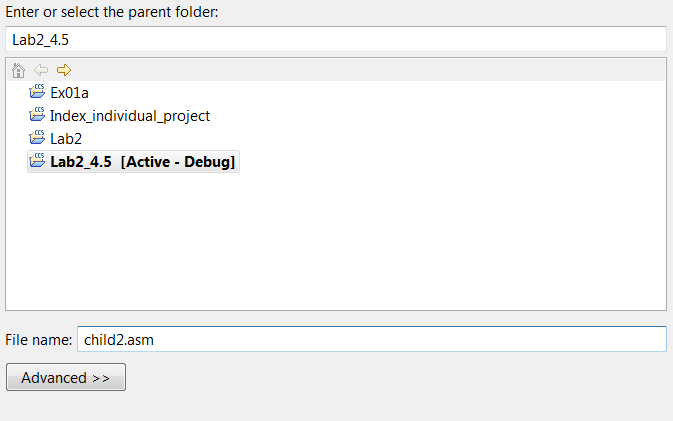
On déroule ensuite le programme pour voir ce que l’on récupère en sortie



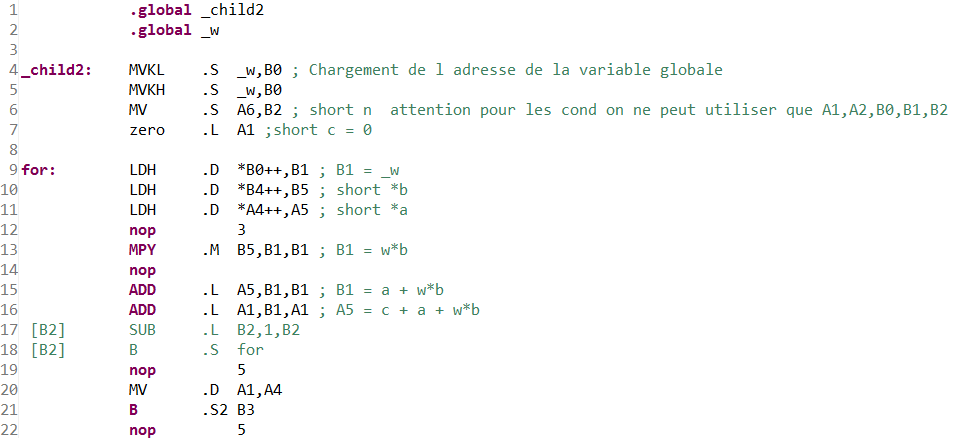
Une fois la simulation terminée on peut l’arréter



Pour l’assembleur normal cette fois-ci, il est nécessaire de créer un autre type de fichier « .asm ».

1. 

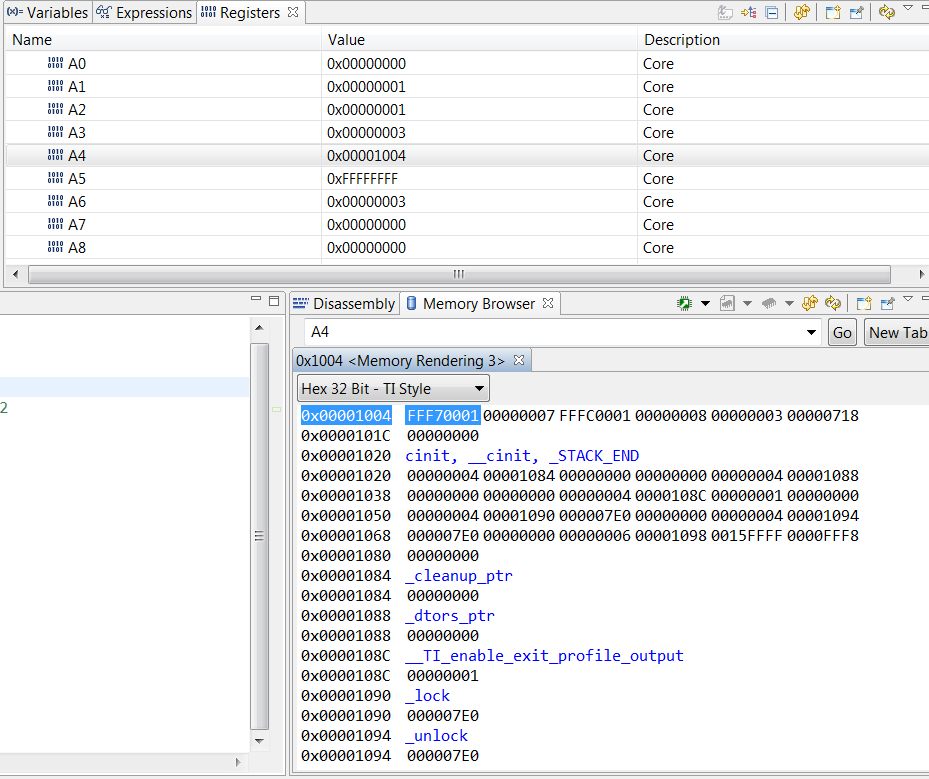
Voici le code du programme en assembleur. Il ne faut pas oublier de prendre en compte les temps de cycles des instructions en rajoutant des nop.



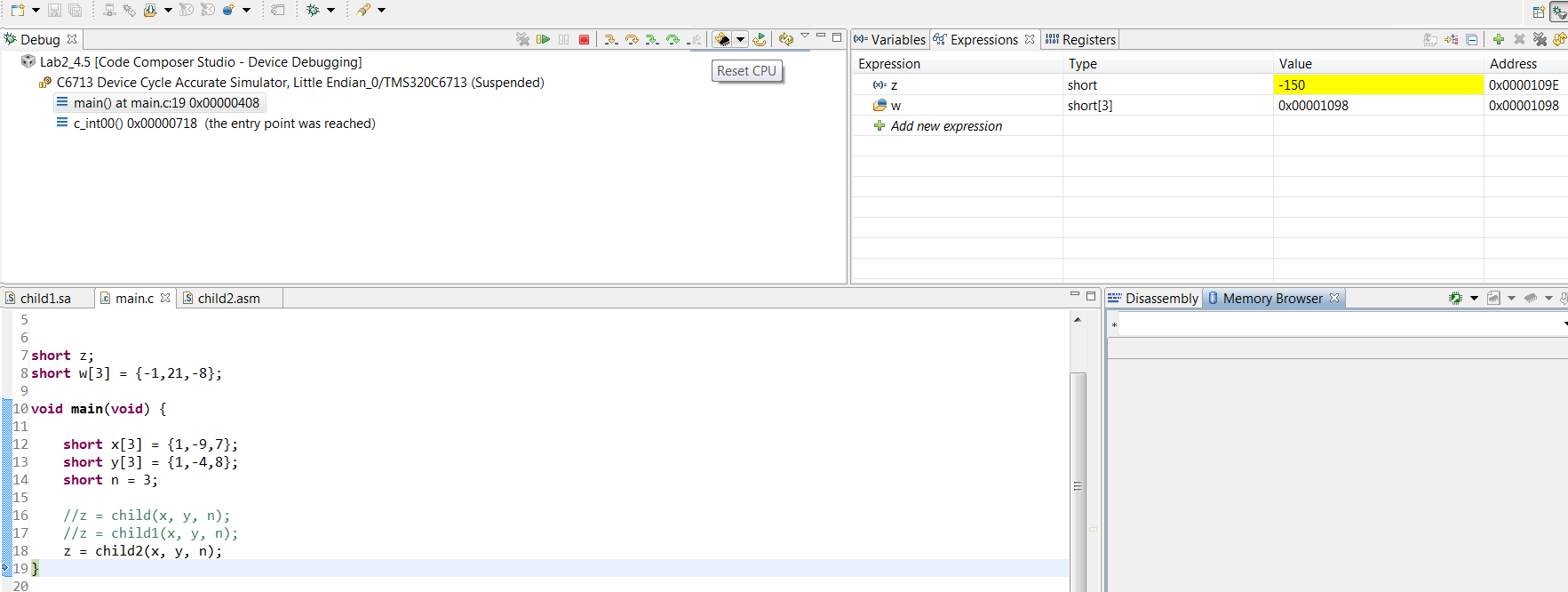
La procédure de debug est ensuite la même que précedemment.

(Répéter I-8 et I-9)

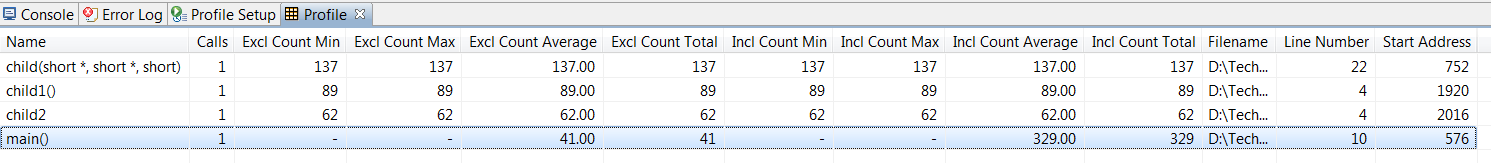
Pour vérifier les variables, il peut être plus pratique d’utiliser la fenêtre Memory Browser. Ainsi au début du programme, si on veut vérifier le premier argument de la fonction ( dans notre cas) il suffit de taper A4 (registre qui contient l’adresse du 1er paramètre dans le cas d’un pointeur). On trouve ainsi dans l’ordre  ;  et ce qui correspond bien à ce qu’on a définit dans le C pour . En effet en complément à 2 : 🡺 1111111111110111 🡺 -9. On peut remarquer que le registre A6 (3ème paramètre) contient directement la valeur de . En effet, on rentre bien directement la valeur comme paramètre et pas par pointeur (adresse).



Pour être sûr que l’assembleur marche bien indépendamment du reste, il peut être intéressant de désactiver les autres fonctions (comme elles ont modifiés les registres, ça a pu faciliter la tâche à l’assembleur comme pour la variable globale par exemple). On clique sur  pour remettre tout à zero et ensuite cliquer à côté sur .



Voici les résultats du profiler de CCS en mode debug (même procédure que précedemment)



En mode release, les paramètres d’optimisation sont bien plus puissant que le programme effectué en assembleur. Comme je n’ai pas optimisé l’assembleur, le nombre de cycles est supérieur au language C et assembleur linéaire. Il ne faut pas oublier qu’en mode Release on a très peu d’informations sur le debug (ce qui est normal car on est censé avoir debugé avant…)

