Programmer/debugger les contrôleurs ARM avec Eclipse et GCC

Introduction

Le présent document explique comment, pas par pas, configurer et utiliser un IDE de classe professionnelle totalement gratuit. L'exemple simple montre comment compiler et debugger un programme avec Eclipse, GCC (GNU Compiler Collection) et OpenOCD sur une cible ARM (LPC2103 de NXP).
Une chaine non « propriétaire » et non limitée :

- en taille de code,
- en optimisations,
- en temps,
- convenant à tous les processeurs ARM supportés par GCC (arm7, arm9 et variantes), de toutes marques (NXP, Atmel, STR,...).

Comme matériel, j'ai utilisé une carte de développement LPC2103 bon marché, avec une sonde JTAGkey tiny de Amontec (http://www.amontec.com/jtagkey-tiny.shtml). Pour ceux qui souhaient encore réduire le coût et qui ont un PC avec un port parallèle, construire un « Wiggler » tel que celui ci : http://wiki.jelectronique.com/projets/wiggler_clone/wiggler_clone

La description suivante est basée sur les denières versions d'Eclipse (3.3, version 2007-fall2 « Europa »), d'OpenOCD (r247 – 30/12/2007) et de GCC (4.2.2)/GDB (6.7.1 du 31/12/2007). Une des difficultés, l'écriture d'un makefile, est évitée.

En préalable, s'assurer que le PC est bien équipé d'un runtime « Java » de version supérieure à 1.5.0 (ce qui est le cas de la grande majorité des PC récents). Ceci peut se vérifier en tapant *java -version* dans une fenêtre DOS. Sinon, le télécharger de http://www.java.com/fr .

Il faudra également télécharger et installer sur le PC les « GNU CoreUtils », qui interprètent les commandes des shell Linux sous Windows :
http://gnuwin32.sourceforge.net/packages/coreutils.htm ou bien
http://sourceforge.net/projects/gnuwin32/

Remarque générale : J'ai personnellement tout installé dans un répertoire qui s'appelle D:\arm. Dans les explications qui suivent, il faudra donc préciser le répertoire utilisateur approprié.

Installation de l'IDE Eclipse

L'installation de la chaine de développement nécessite d'installer :

- L'IDE eclipse; qui devra être complété par « CDT » (C Developpement Toolkit) et par « Zylin embedded CDT » (appelés « plugins »).
- Le compilateur GCC pour ARM (arm-elf-gcc)

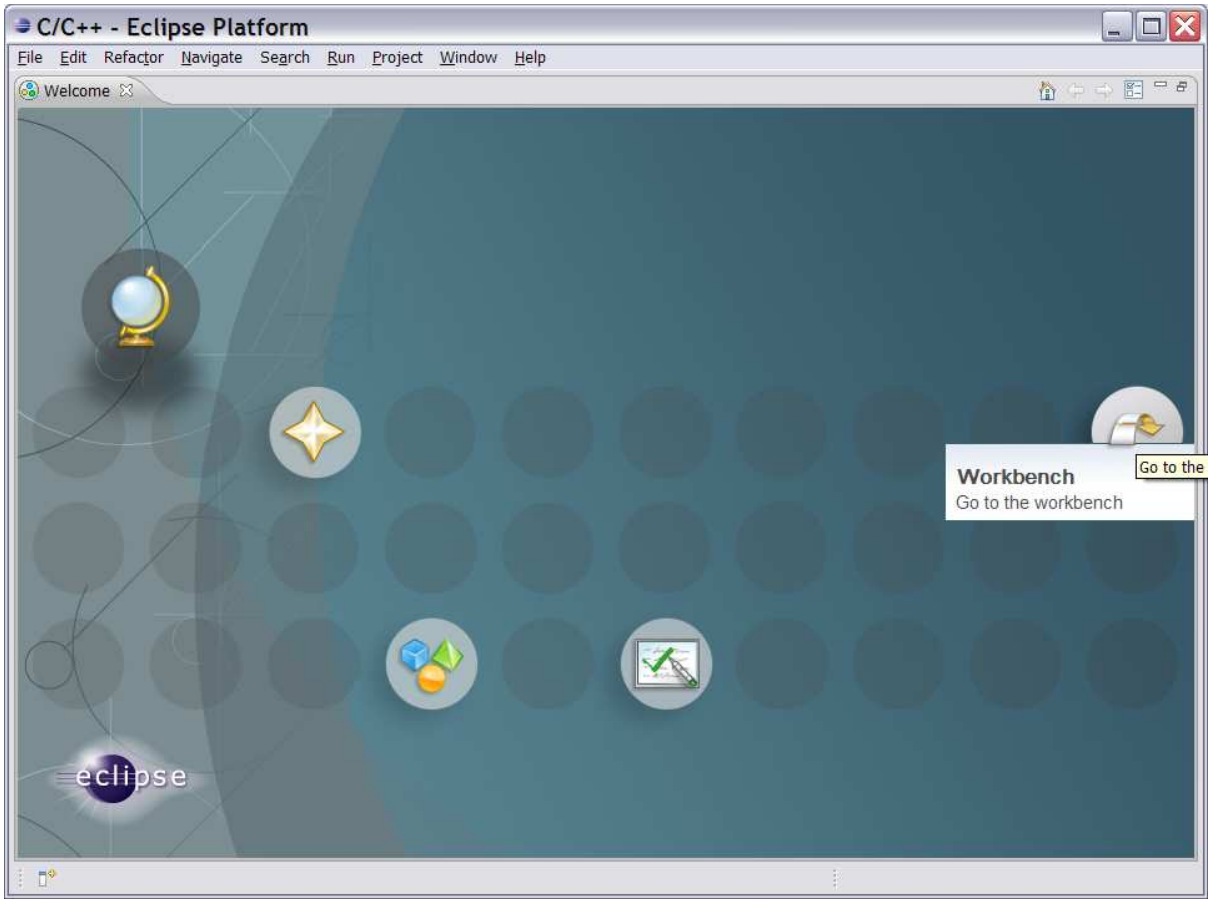
Se connecter à http://www.eclipse.org , y télécharger « Eclipse IDE for C/C++ developpers » :

Le fichier qui sera téléchargé s'appelle : **eclipse-cpp-europa-fall2-win32.zip**

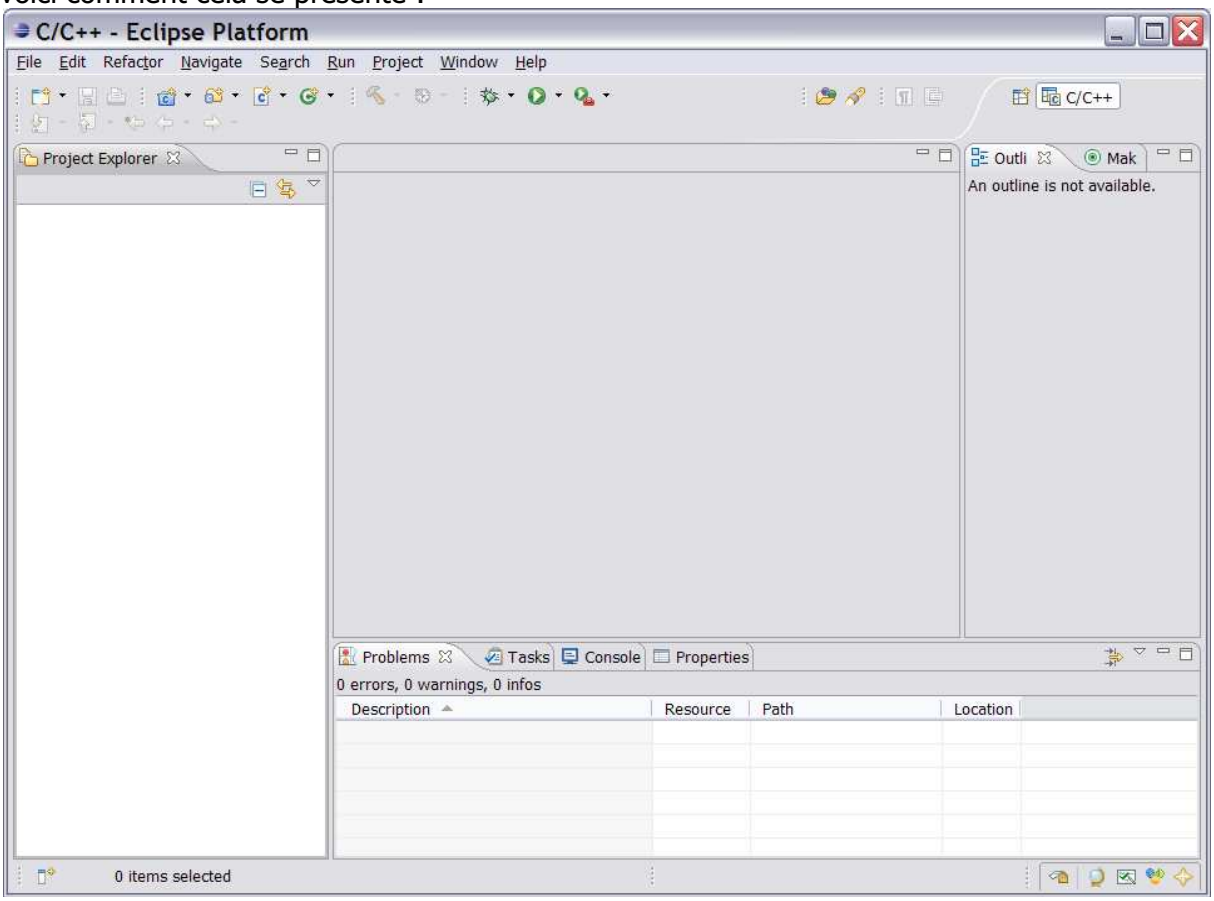
L'installation est simple : Ce zip contient un répertoire **eclipse** qu'il il suffit de décompresser à un endroit approprié.
L'étape suivante consiste maintenant à installer CDT et Zylin Embedded CDT.
Il y a d'autres méthodes que celle que je vais décrire, mais j'ai eu des problèmes d'incompatibilité entre versions; celle-ci évite ces ennuis.

Il faut lancer éclipse, simplement en cliquant sur « eclipse.exe »
Il proposera d'abord un « workspace », c'est le dossier de travail. Cliquer « OK » si celui qui est proposé convient, sinon choisir un autre.

Voilà l'écran d'accueil ; cliquer « goto the workbench »



Et voici comment cela se présente :



Cette vue, telle qu'elle apparaît à l'écran s'appelle une « **Perspective** ».
Ici, il s'agit de la « Perspective » « C/C++ »

Aller directement dans le menu : Help ->Software Update -> Find and Install ; sur l'écran qui appraît, choisir : « Search for new features to install »

Je ne vais pas décrire la suite des es manipulations en détail, la manière de procéder est parfaitement décrite ici :
http://subclipse.tigris.org/install.html (Ne pas installer subclipse, c'est simplement pour la manière de procéder)

Les URL des « new remote sites » desquels CDT et Zylin CDT seront téléchargés sont les suivants:

Pour CDT, le site est **http://download.eclipse.org/tools/cdt/releases/europa**

Pour Zylin embedded CDT, le site est **http://www.zylin.com/zylincdt**

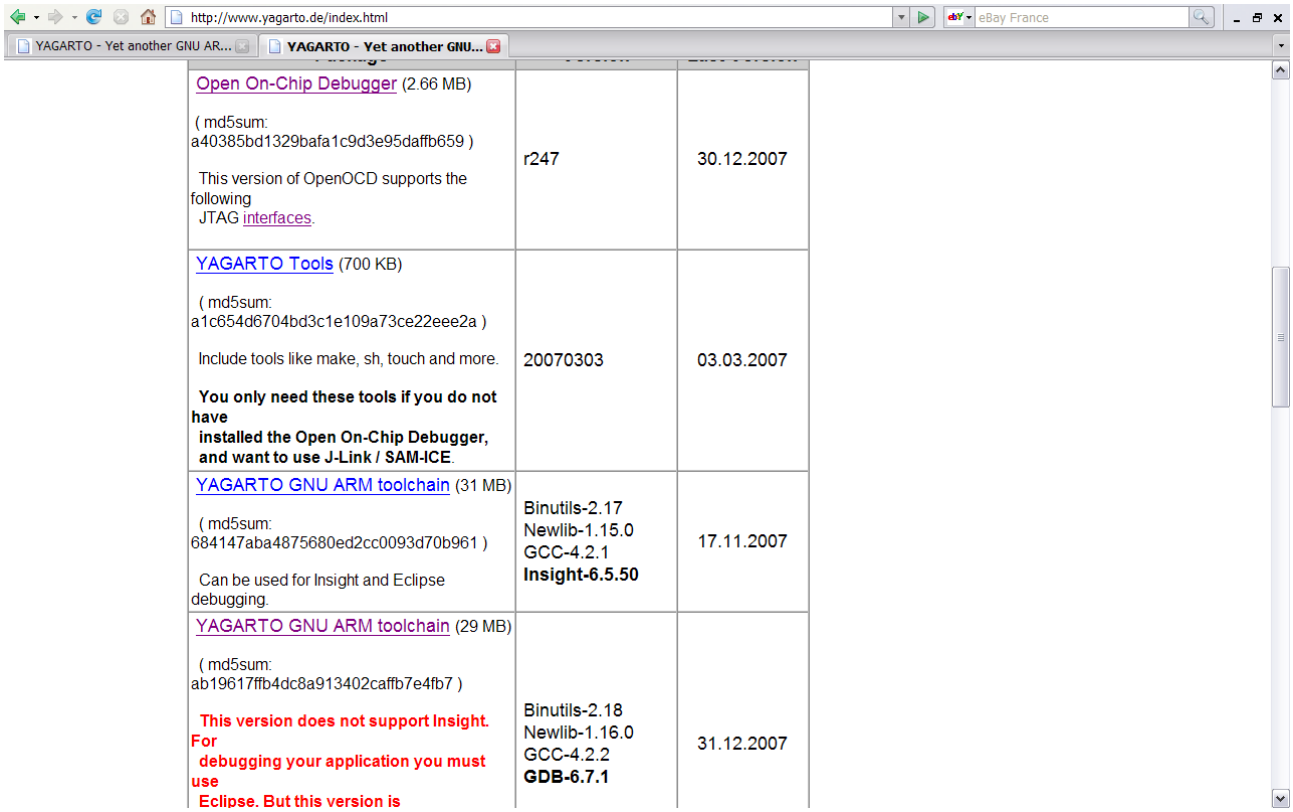
Comme « name », on peut mettre ce que l'on veut (par exemple CDT et ZylinCDT)
Sélectionner les updates trouvés, accepter les licences, et « OK » pour les menus.
Installer l'un et puis l'autre, eclipse se redémarrera après chaque installation.

Après ces deux installations, Eclipse est configuré.

Installation de la chaine GCC/GNU :

Elle est disponible gratuitement ici :

http://www.yagarto.de/index.html



Open On-Chip Debugger (2.66 MB) (md5sum: a40385bd1329bafa1c9d3e95daffb659) This version of OpenOCD supports the following JTAG interfaces.	r247	30.12.2007
YAGARTO Tools (700 KB) (md5sum: a1c654d6704bd3c1e109a73ce22eee2a) Include tools like make, sh, touch and more. **You only need these tools if you do not have installed the Open On-Chip Debugger, and want to use J-Link / SAM-ICE.**	20070303	03.03.2007
YAGARTO GNU ARM toolchain (31 MB) (md5sum: 684147aba4875680ed2cc0093d70b961) Can be used for Insight and Eclipse debugging.	Binutils-2.17 Newlib-1.15.0 GCC-4.2.1 **Insight-6.5.50**	17.11.2007
YAGARTO GNU ARM toolchain (29 MB) (md5sum: ab19617ffb4dc8a913402caffb7e4fb7) **This version does not support Insight. For debugging your application you must use Eclipse. But this version is**	Binutils-2.18 Newlib-1.16.0 GCC-4.2.2 **GDB-6.7.1**	31.12.2007

Télécharger « YAGARTO GNU ARM toolchain »

Le fichier téléchargé d'appelle :
yagarto-bu-2.18_gcc-4.2.2-c-c++_nl-1.16.0_gdb-6.7.1_20071231.exe

De même, télécharger « Open On-Chip Debugger » (OpenOCD), version 30.12.2007
Le fichier téléchargé d'appelle : **openocd-r247-20071230.exe**

Cette version est utilise directement GDB. Il n'est plus nécessaire d'utiliser Insight.

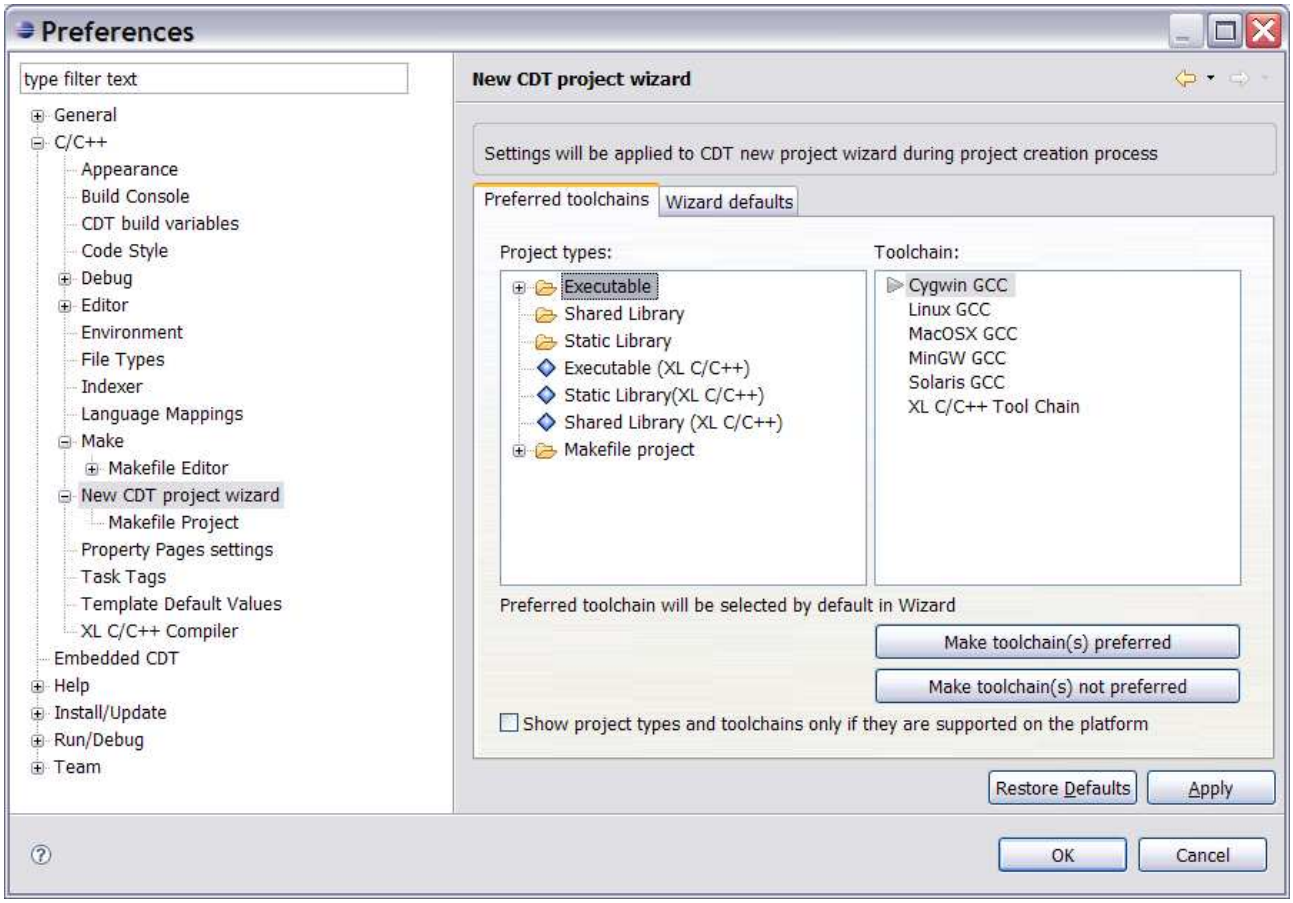
Exécuter ces fichiers afin de les installer dans le répertoire adéquat. Vérifier que le cmpilateur est fonctionnel : dans une fenêtre DOS, taper *arm-elf-gcc -v* . Il répondra par les option disponnibles et la version.

Si la carte d'essai ARM possède un jumper de sélection RUN/DEBUG, ne pas oublier de positionner celui-ci sur « DEBUG »

Le premier programme

Lancer Eclipse,

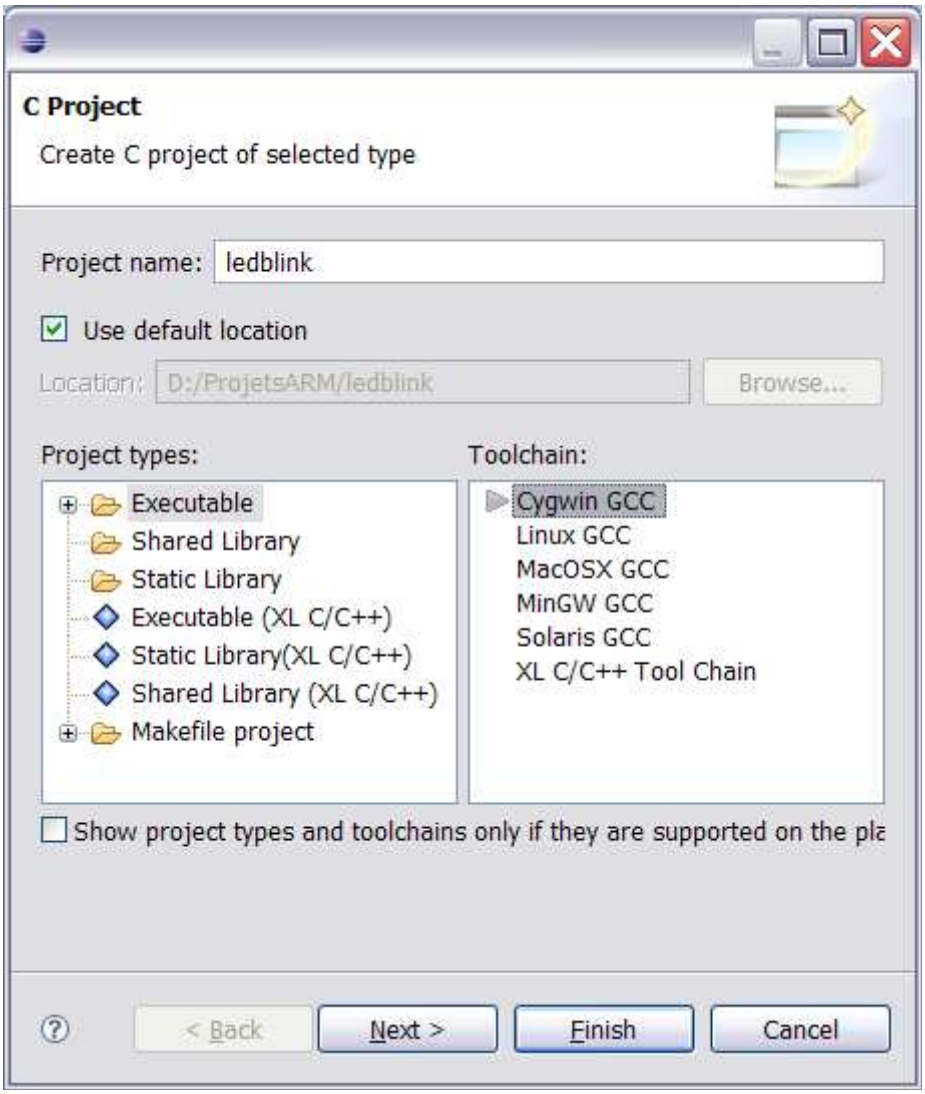
1ère étape importante : aller dans le menu : Window -> Preferences.. cliquer sur « New CDT project Wizzard » déselectionner le « Show projects types and toolchains only if they are supported on the plattform » , puis cliquer sur « Cygwin GCC » et « Make toolchain preferred »



Le premier projet :

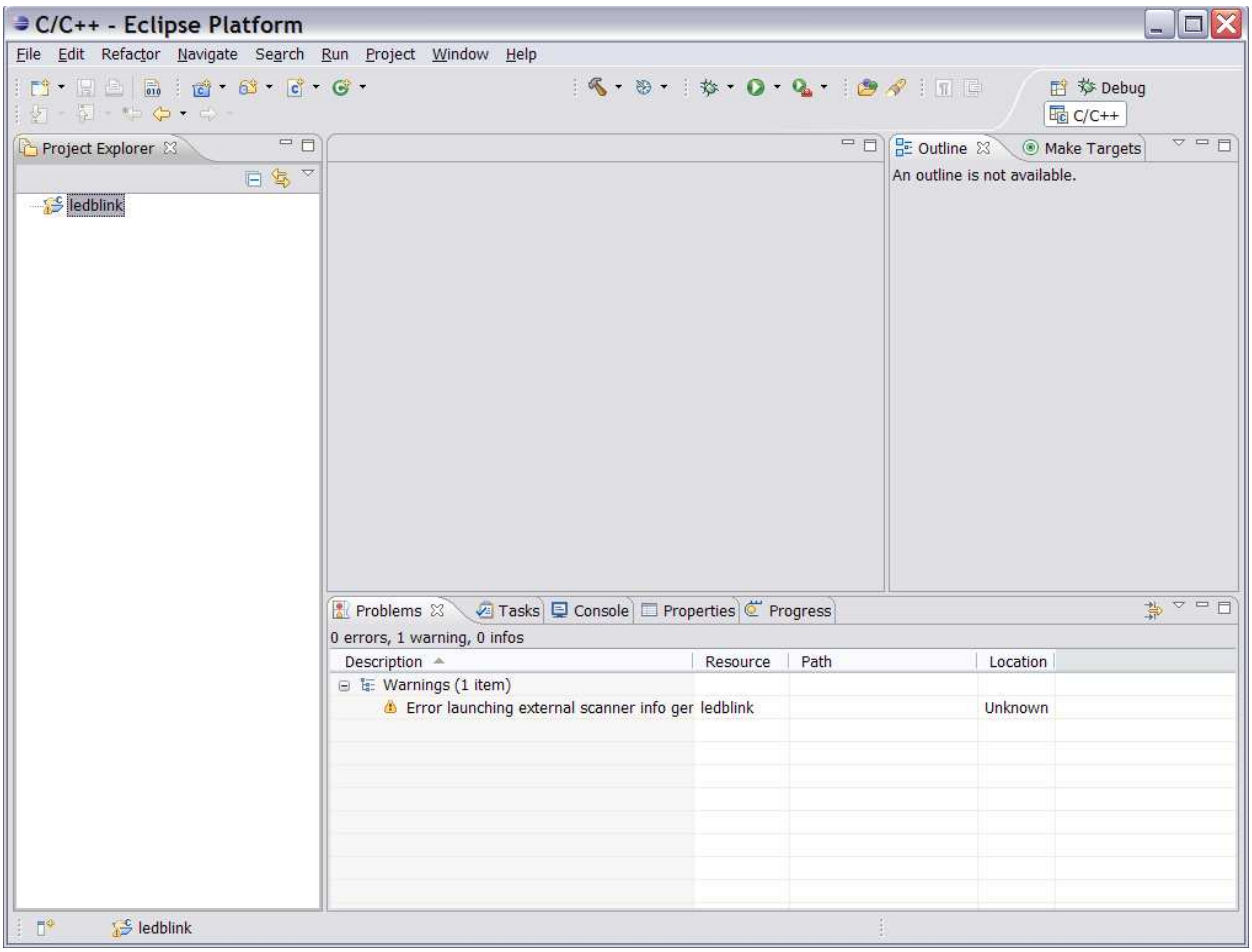
Aller dans le menu : File -> New -> C project

Dans le wizzard, taper le nom souhaité : *ledblink* ; puis comme précédemment : déselectionner le « Show projects and toolchains only if they are supported on the plattform » , puis cliquer sur « Cygwin GCC »

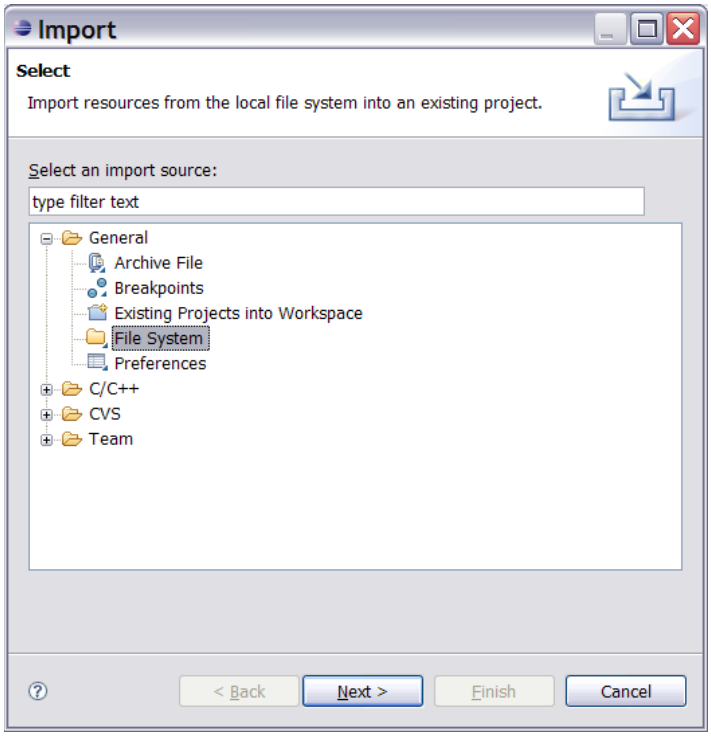


Cliquer sur « Finish » L'écran se présente alors comme ceci :

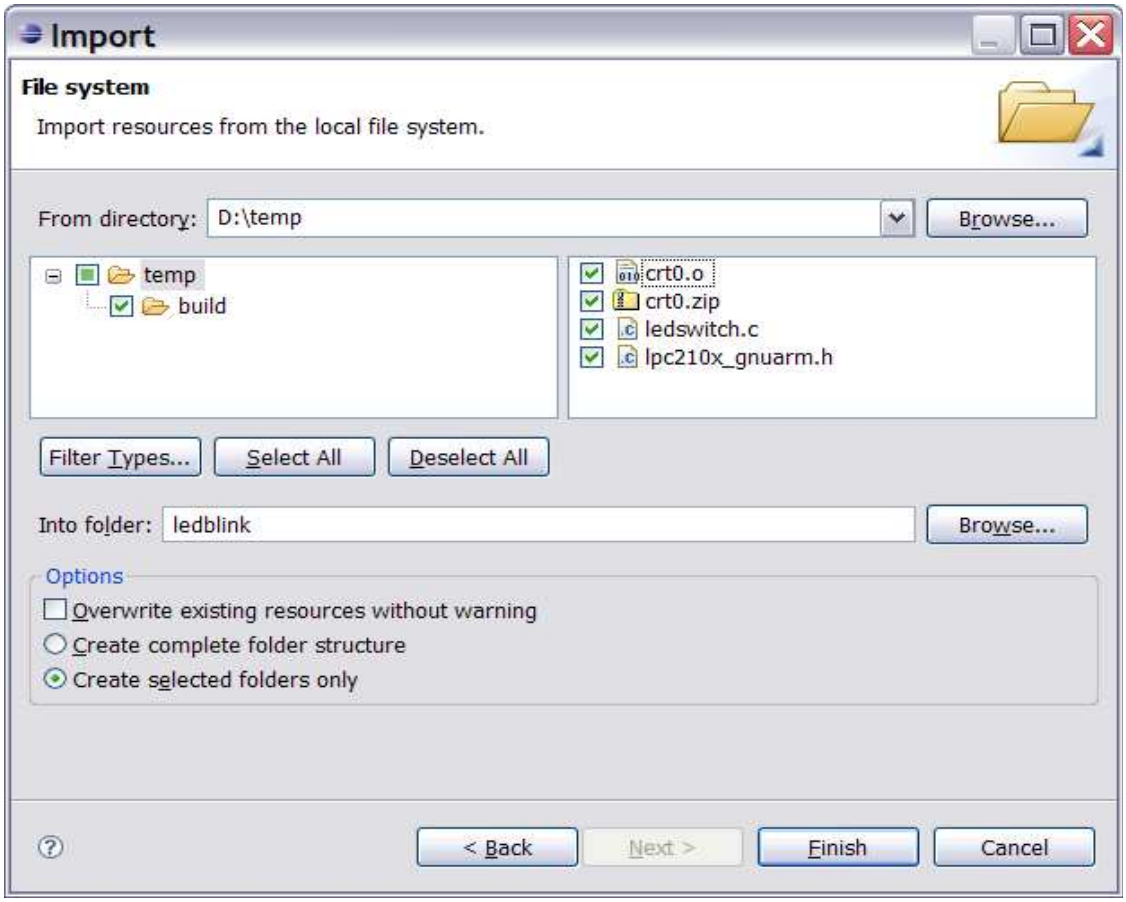
Un warning, mais ce n'est pas grave, il suffit de décocher l'option « Build Automatically » dans le menu « Project »



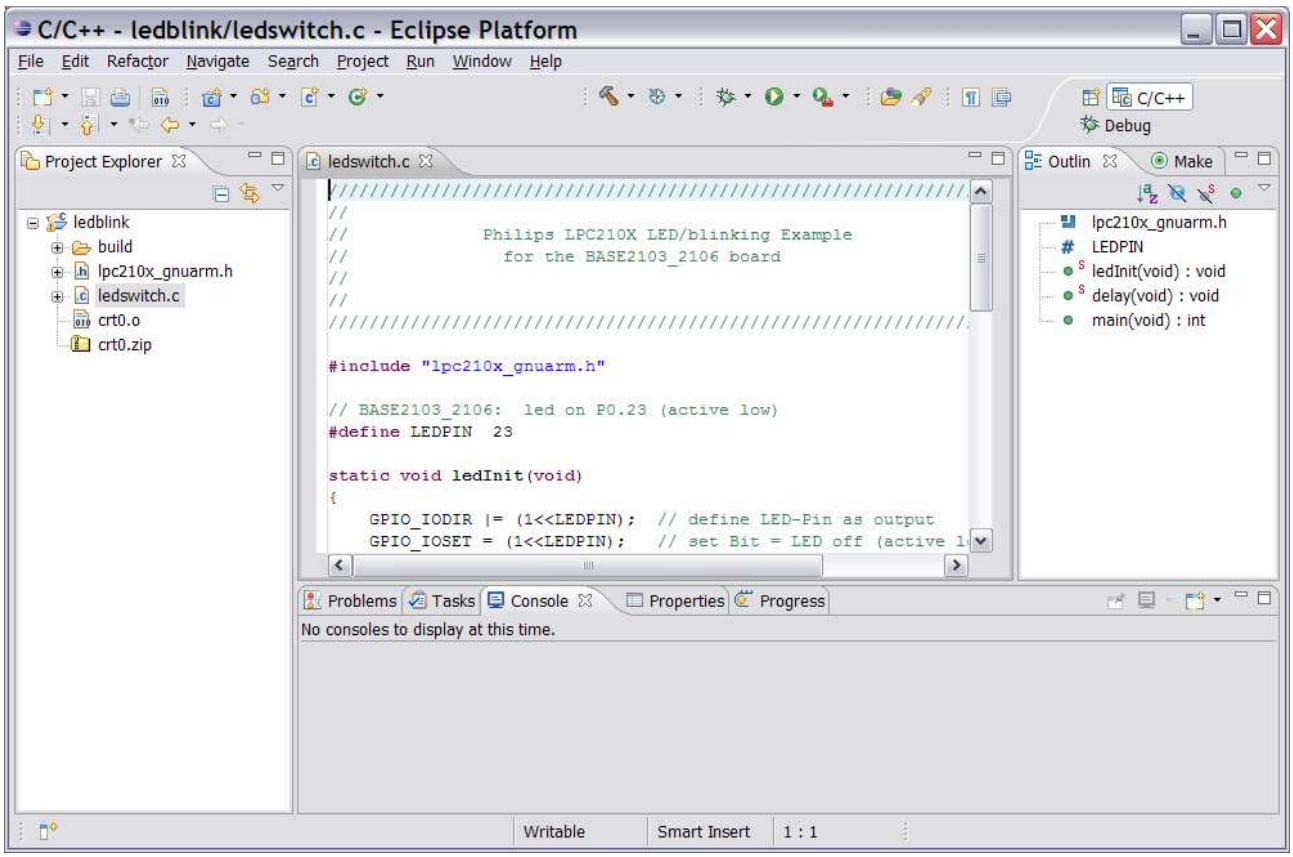
Il faut maintenant ajouter les sources C :
Cliquer sur le nom du projet (ledblink), puis par un clic sur le bouton droit de la souris dans le menu contextel : Import -> File system



Sélectionner ensuite (Next) le dossier où est décompressé l'exemple joint à cet article :
Choisir les fichiers / Folders à importer; l'écran se présente alors comme ceci :



Cliquer sur « Finish », Eclipse se présente alors comme suit, (après avoir double-cliqué sur « ledswitch.c », pour faire apparaître le code) :



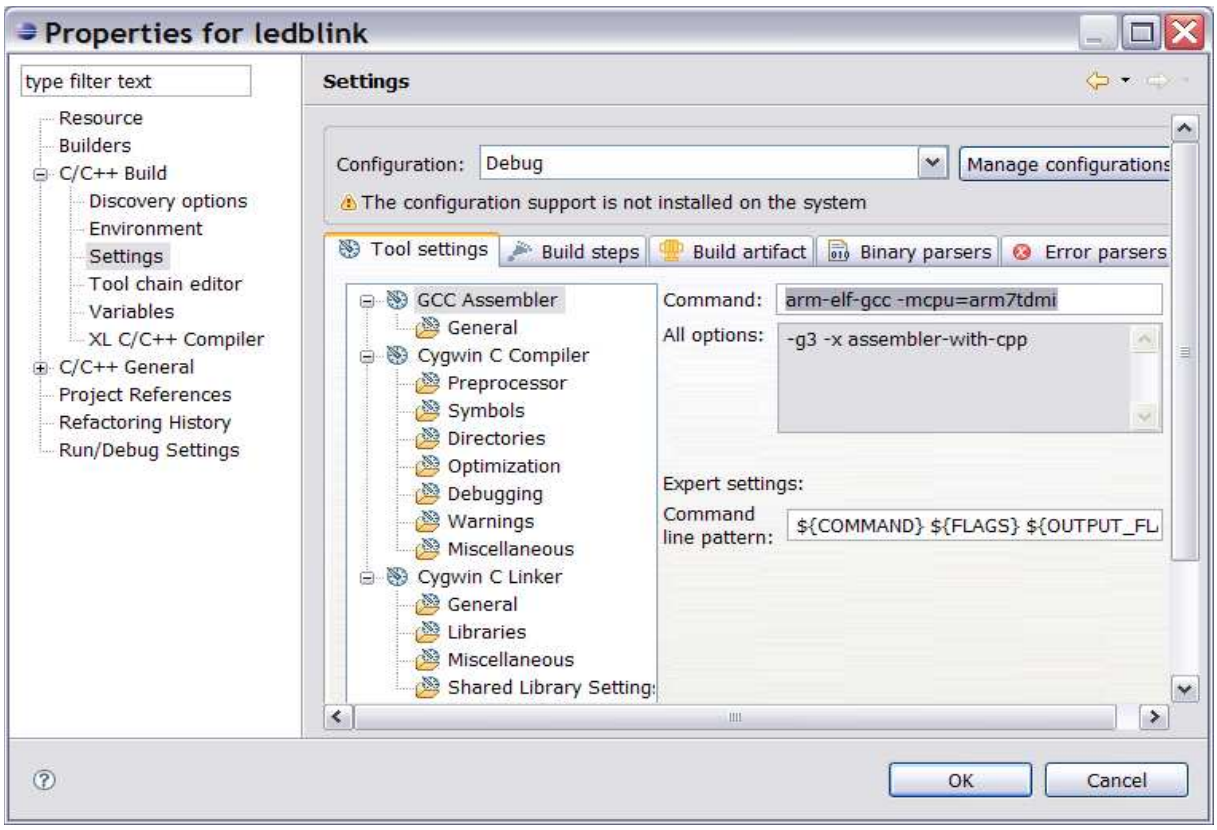
Attention, toute modification effectuée dans la fenètre centrale (édition) doit être suivie d'un save, sinon, elle n'est pas prise en compte. Ne pas oublier donc d'effectuer cette opération avant toute compilation (« Build »).

Commence alors une délicate étape de configuration, mais elle n'est répétée qu'une seule fois (par projet) et permet d'éviter l'écriture d'un Makefile assez complexe, qui sera généré automatiquement : cliquer sur le nom du projet, puis par un clic sur le bouton droit de la souris dans le menu contextuel : **Properties**

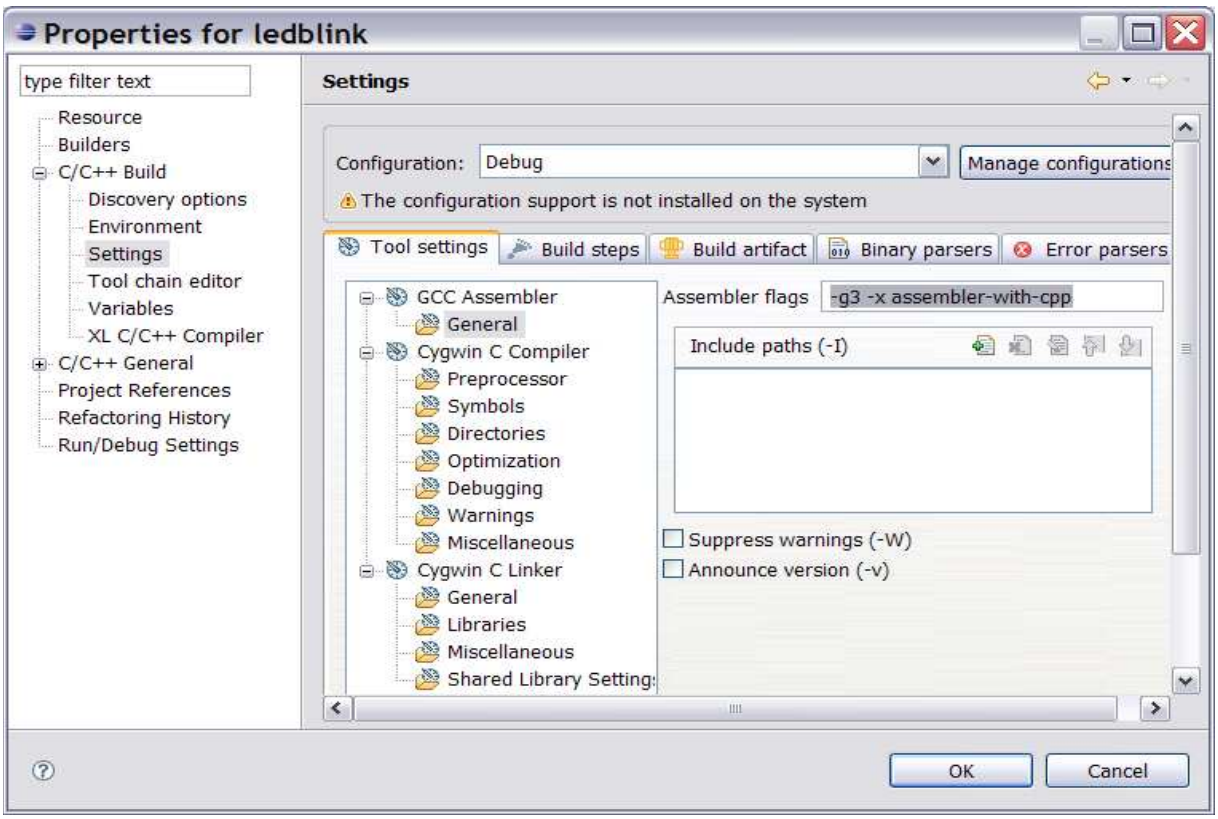
Cliquer sur le menu **C/C++ build** :

1. Cliquer maintenant sur « Settings » de C/C++ build, dans l'onglet « tools settings »
Il faut modifier les invocations de l'assembleur, du compilteur et du linker :

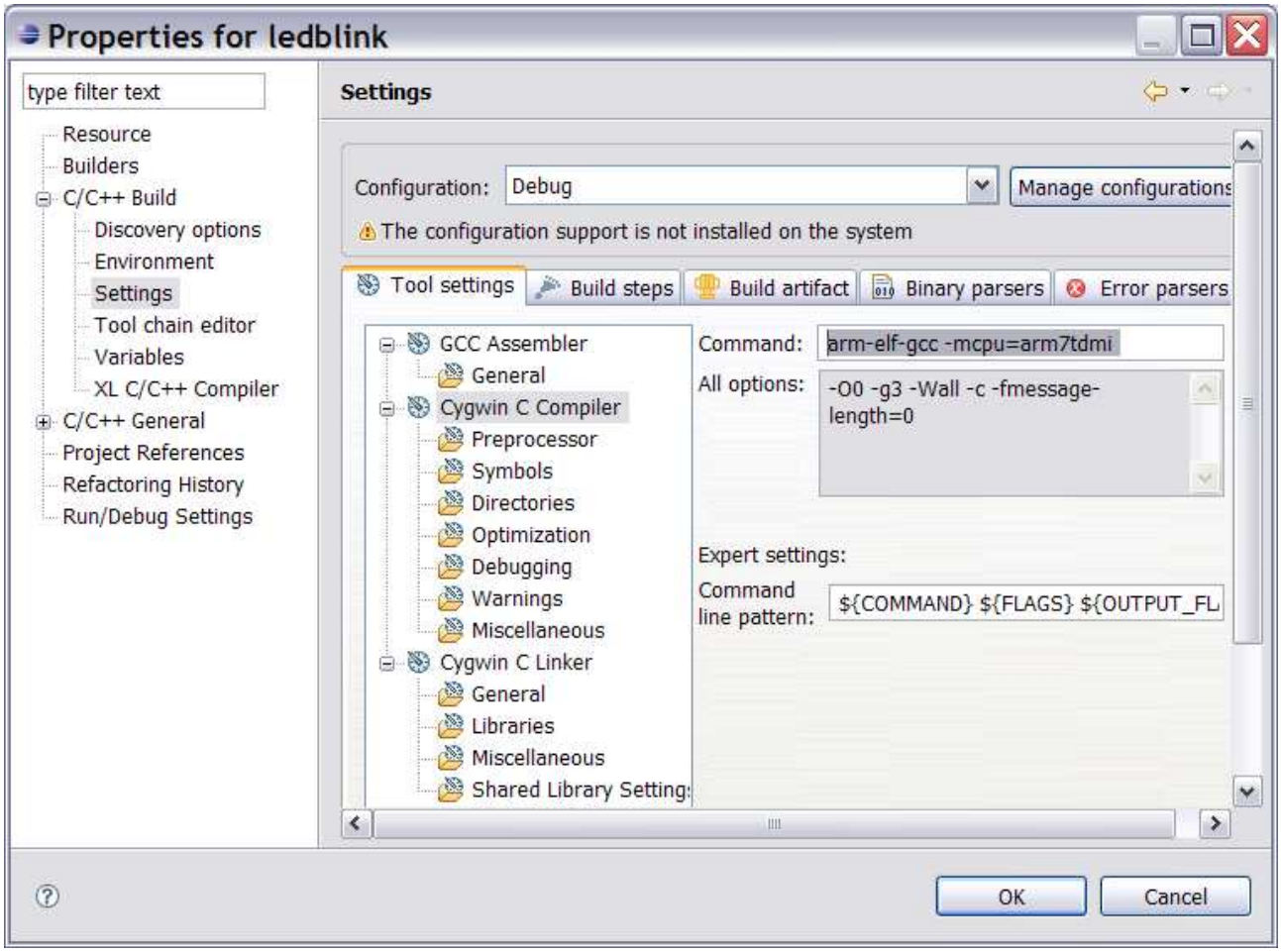
Modifier l'invocation de **l'assembler** « as »
par «**arm-elf-gcc -mcpu=arm7tdmi** »



Dans le menu « General » juste en dessous de « GCC assembler, rajouter come « Assembler Flags » : **-g3 -x assembler-with-cpp** Ce Flag apparaîtra en grisé dans la fenêtre « All Options » de l'assembler



2. De la même manière, Cygwin C **compiler** toujours de l'onglet « Tools settings » Command :



Modifier l'invocation du compilateur par **arm-elf-gcc -mcpu=arm7tdmi**

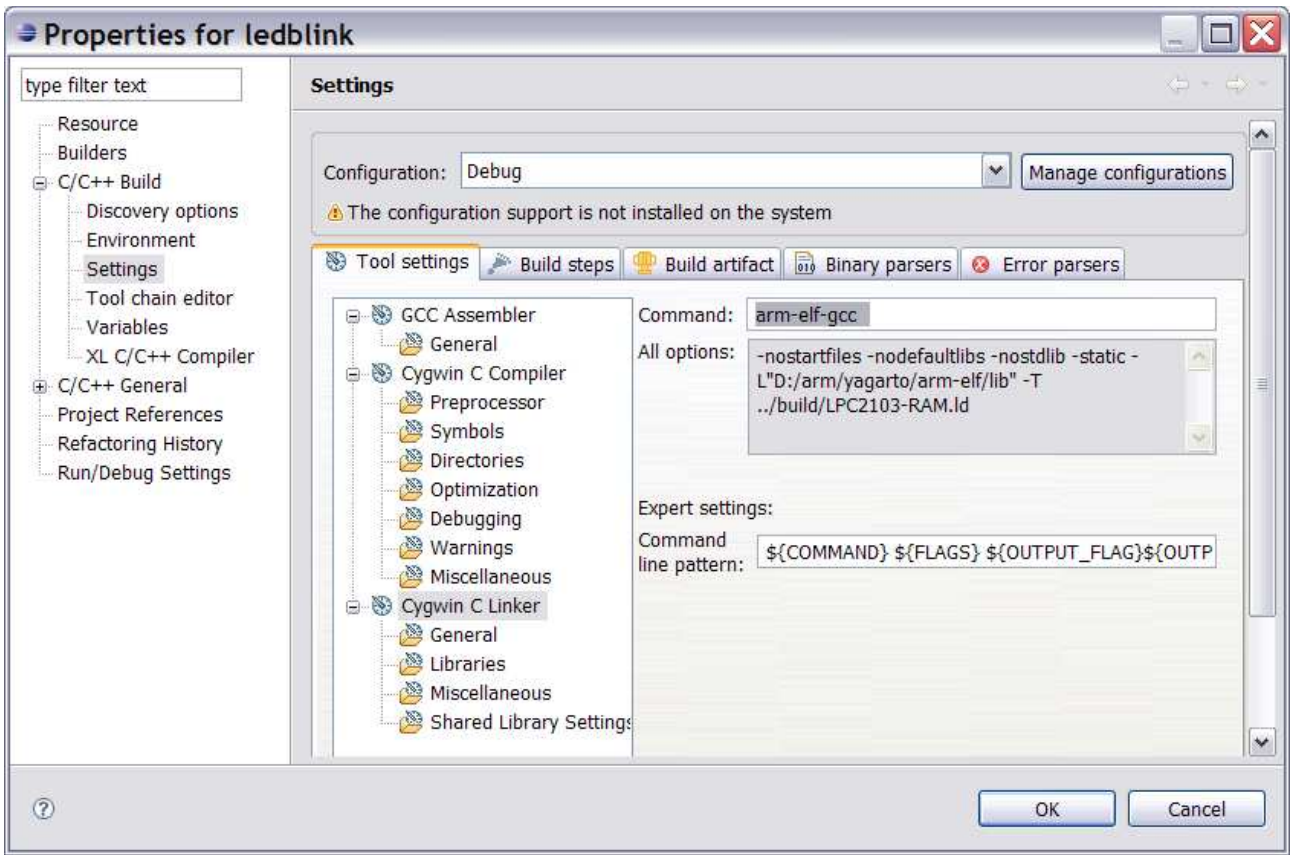
3. La configuration du **Linker** : essentiellement trois choses :

- Modifier l'invocation

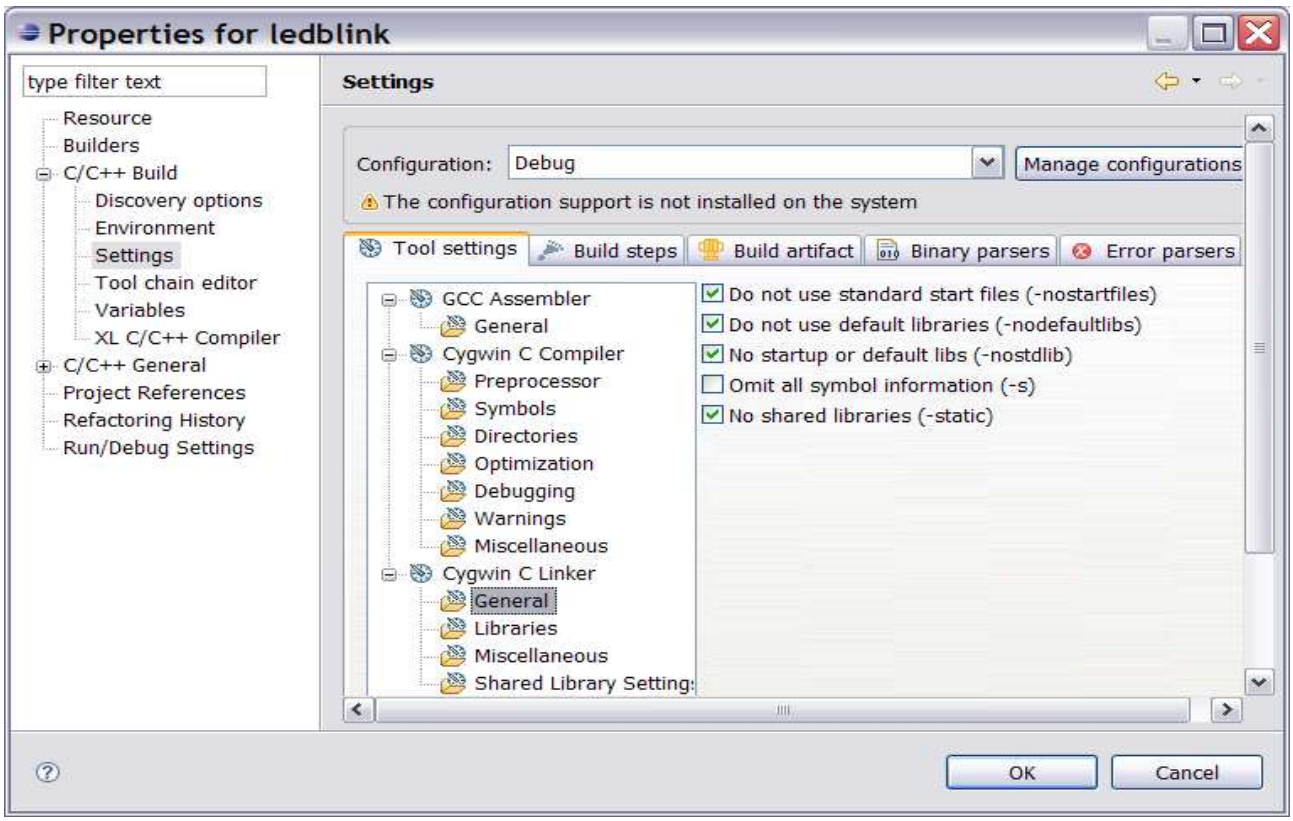
- Préciser le script : ici **LPC2103-RAM.ld** .
  Pour un exécutable en Flash, on choisira LPC2103-ROM.ld. Ces scripts sont facilement modifiables en fonction du processeur : simplement changer la taille de la Flash et de la Ram.

- Ajouter la routine de bas niveau « crt0.o » : Cette routine est celle qui est donnée dans l'exemple joint (crt0.S) en zip, que j'ai compilée en ligne de commande dos. J'ai essayé d'utiliser (sans succès) celles fournies dans la distribution GCC. J'ai donc opté pour cette solution.

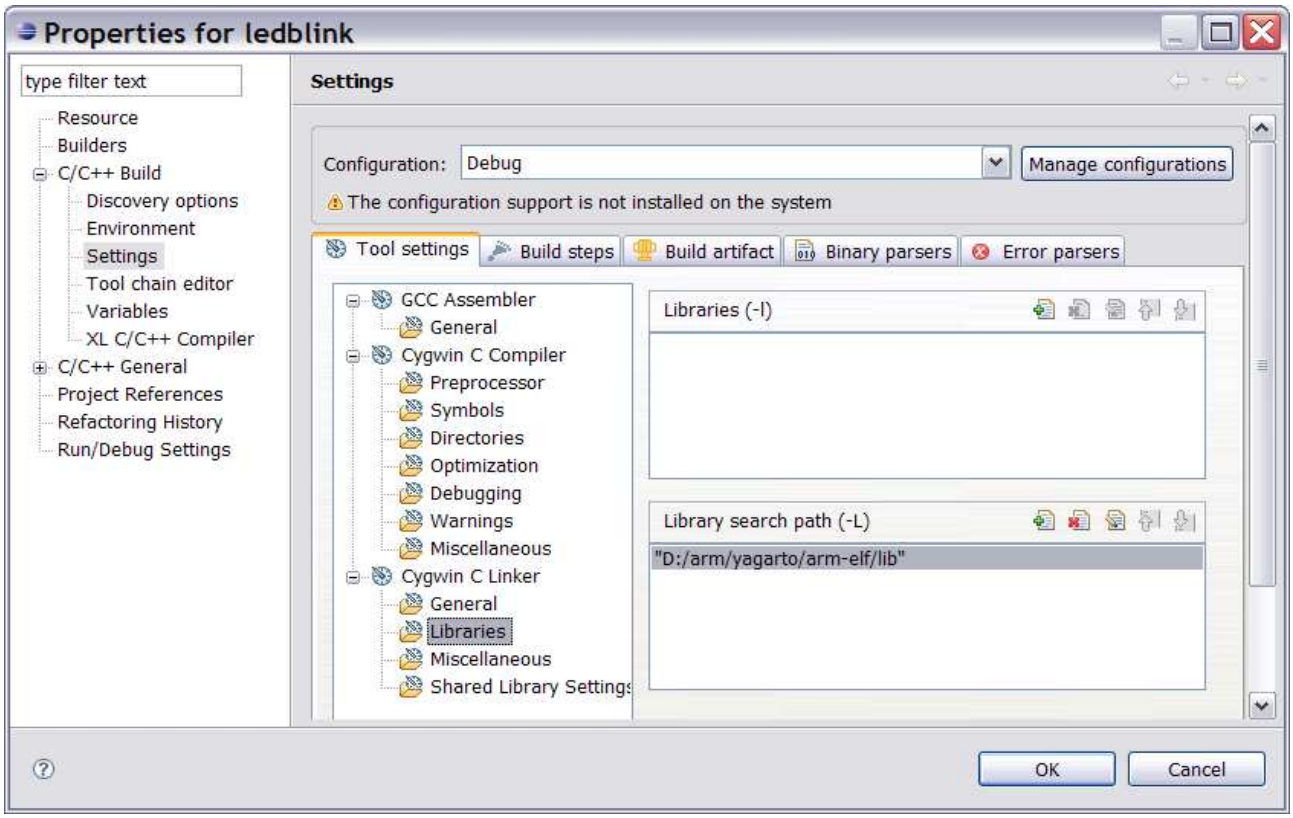
L'invocation du linker : **arm-elf-gcc**



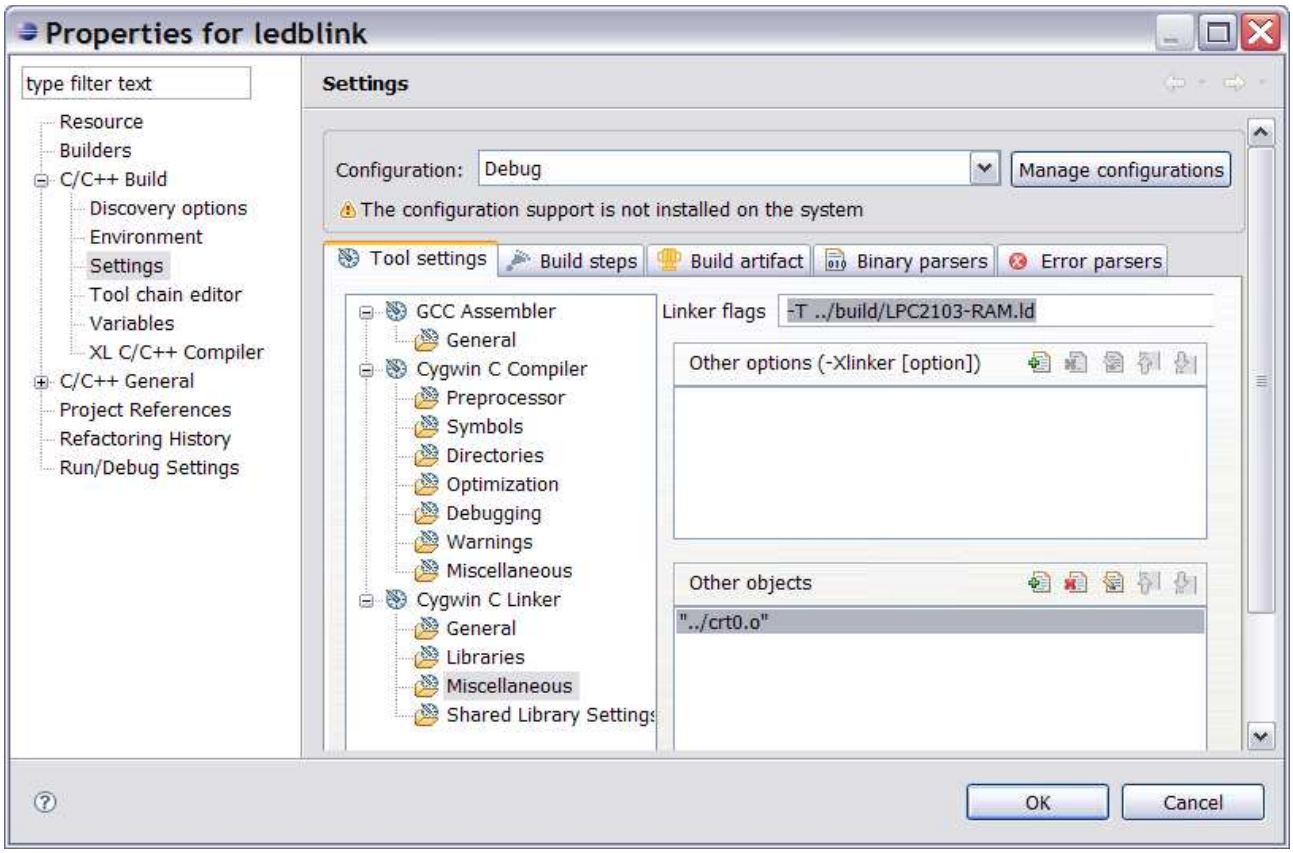
Dans menu « General », spécifier les options comme suit :



Dans le menu « Librairies » (j'ai installé Yagarto dans le folder « D:\arm – modifier le cas échéant)



Le menu « Miscellaneous »



Rajouter **-T ../build/LPC2103-RAM.ld** dans « Linker Flags ».

Si c'est pour générer un exécutable en FLASH, spécifier LPC2103-ROM.ld

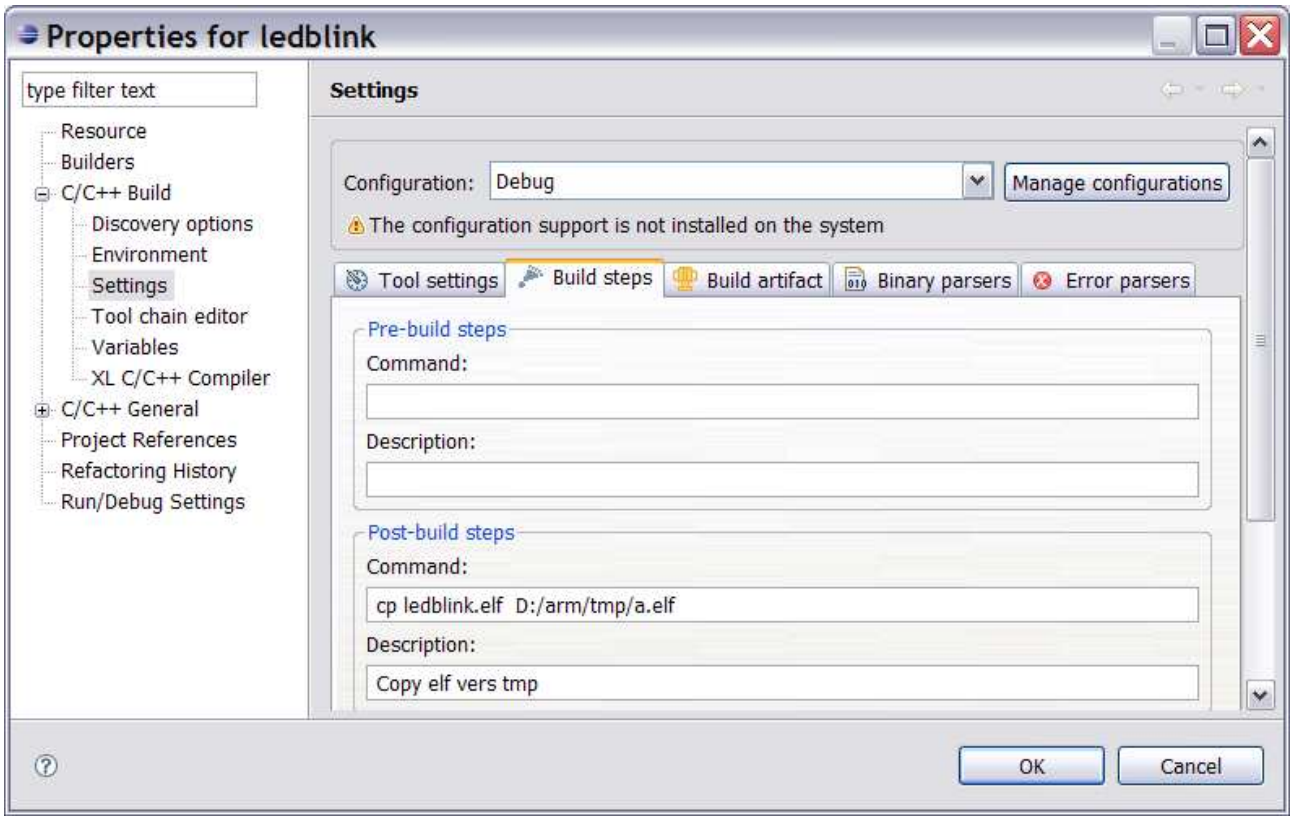
A l'aide des boutons :

rajouter **../crt0.o** dans « Other Objects »

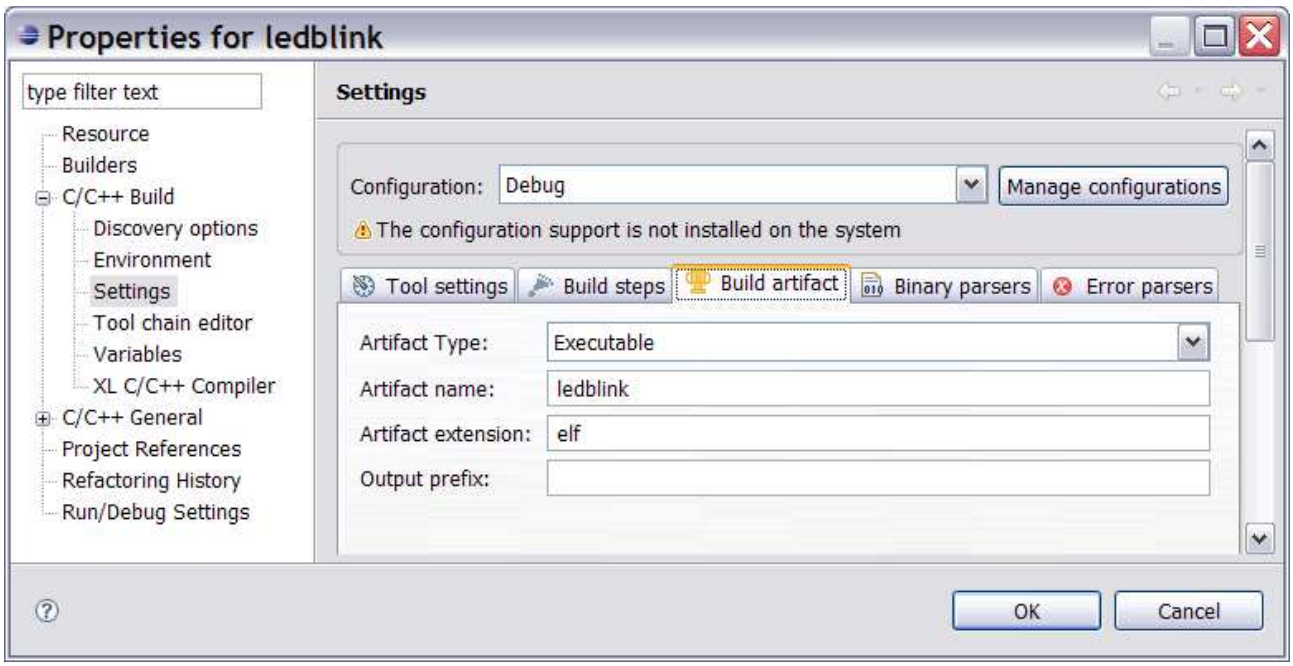
Le « ../ » s'explique du fait que le makefile est exécuté à partir de « Debug ».

Attention, YAGARTO a certaines reminescences Linux : Dans certains cas, mettre « \ » (Folder Windows) à la place de « / » (Folder Linux) provoquera une erreur (file not found).

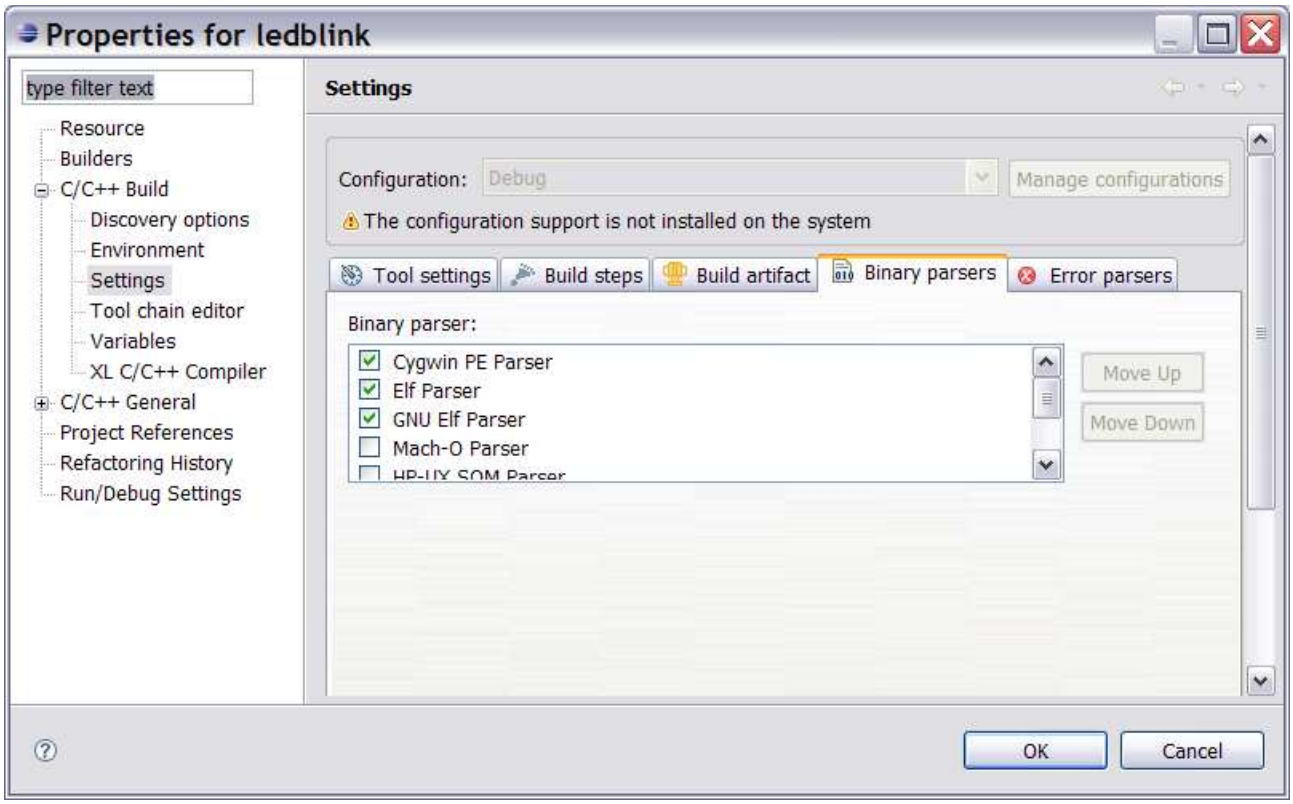
Dans l'onglet « Build steps » rajouter la commande « **cp ledblink.elf D:/arm/tmp/a.elf »** et un commentaire approprié comme « Description »qui servira à copier notre exécutable ledblink.elf vers un répertoire temporaire (ici, j'ai pris D:/arm/tmp) pour la programmation flash.



Dans l'onglet suivant «Build artifact », remplacer l'Artifact extension (.exe), qui peut poser des problèmes, par **.elf** :



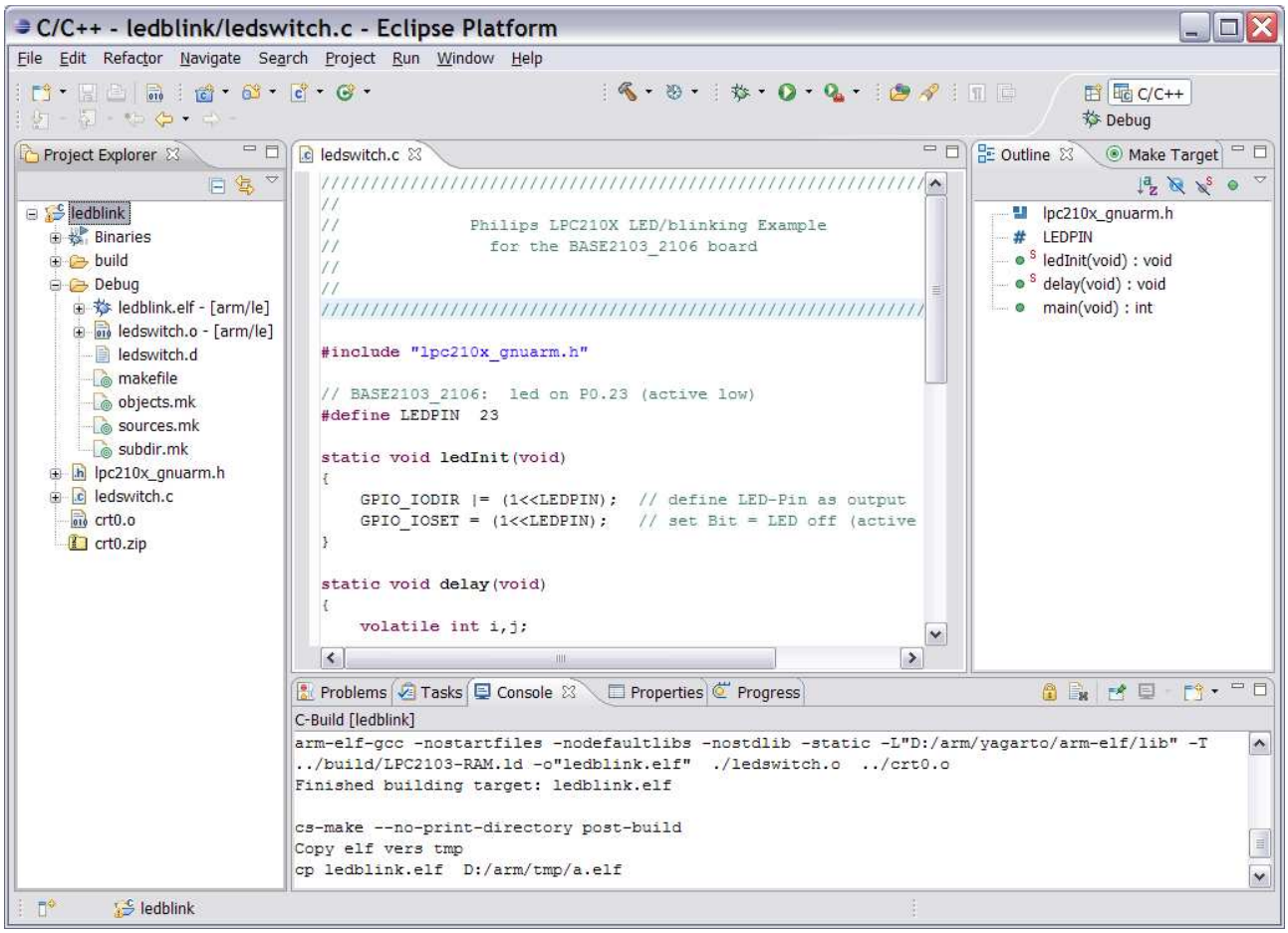
Finalement, l'écran « Binary parsers » doit ressembler à ceci » (cocher les options appropriées):



Cliquer sur « OK » pour fermer l'écran, le projet est alors prêt à être compilé. La création d'un exécutable s'appelle un « build ».

Dans l'écran principal, aller dans le menu « Project » puis « Clean.. », s'assurer que l'option « start a build immediately » est bien coché puis « OK ».

Après avoir cliqué sur « OK », 'écran se présente comme ceci :



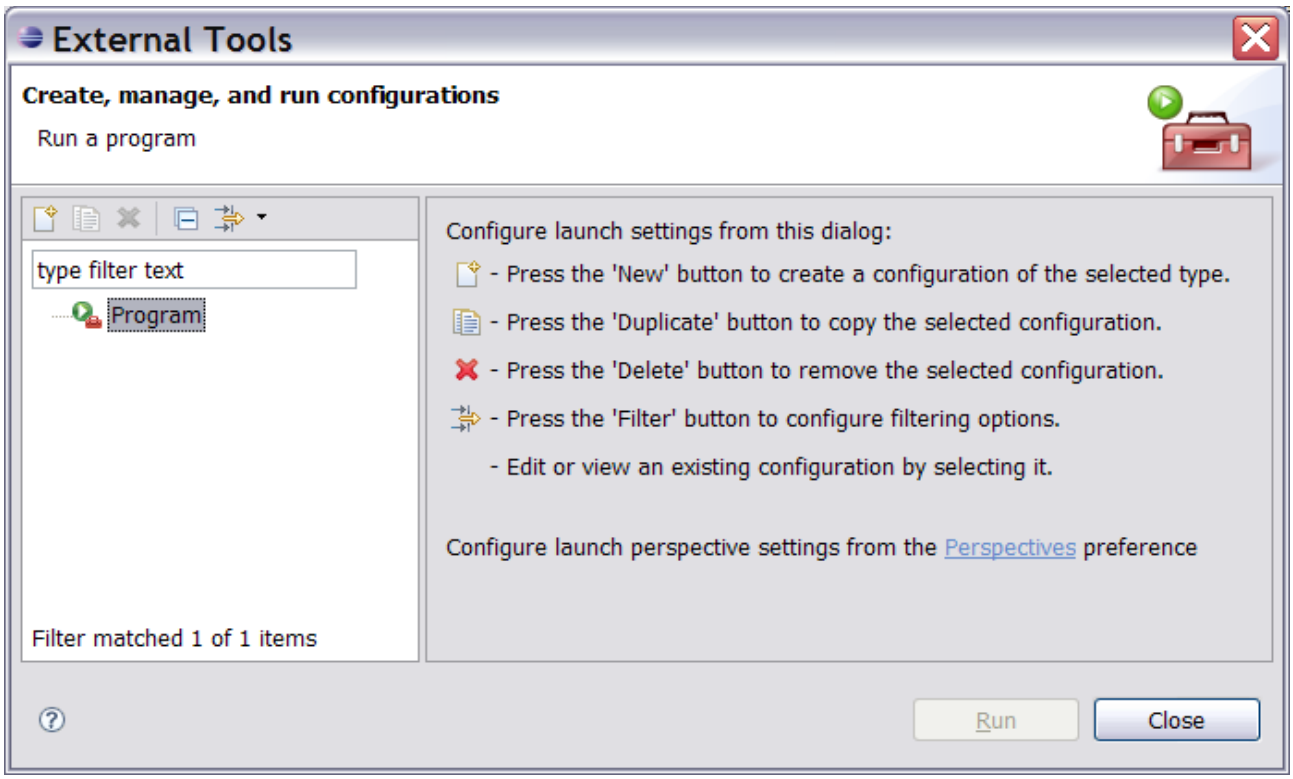
On peut constater que les différents « makefile » (et .mk) ont été rajoutés automatiquement (dans un nouveau folder « Debug »).
Un fichier exécutable « .elf » est présent.

L'étape suivante est l'intégration d'OpenOCD dans Eclipse

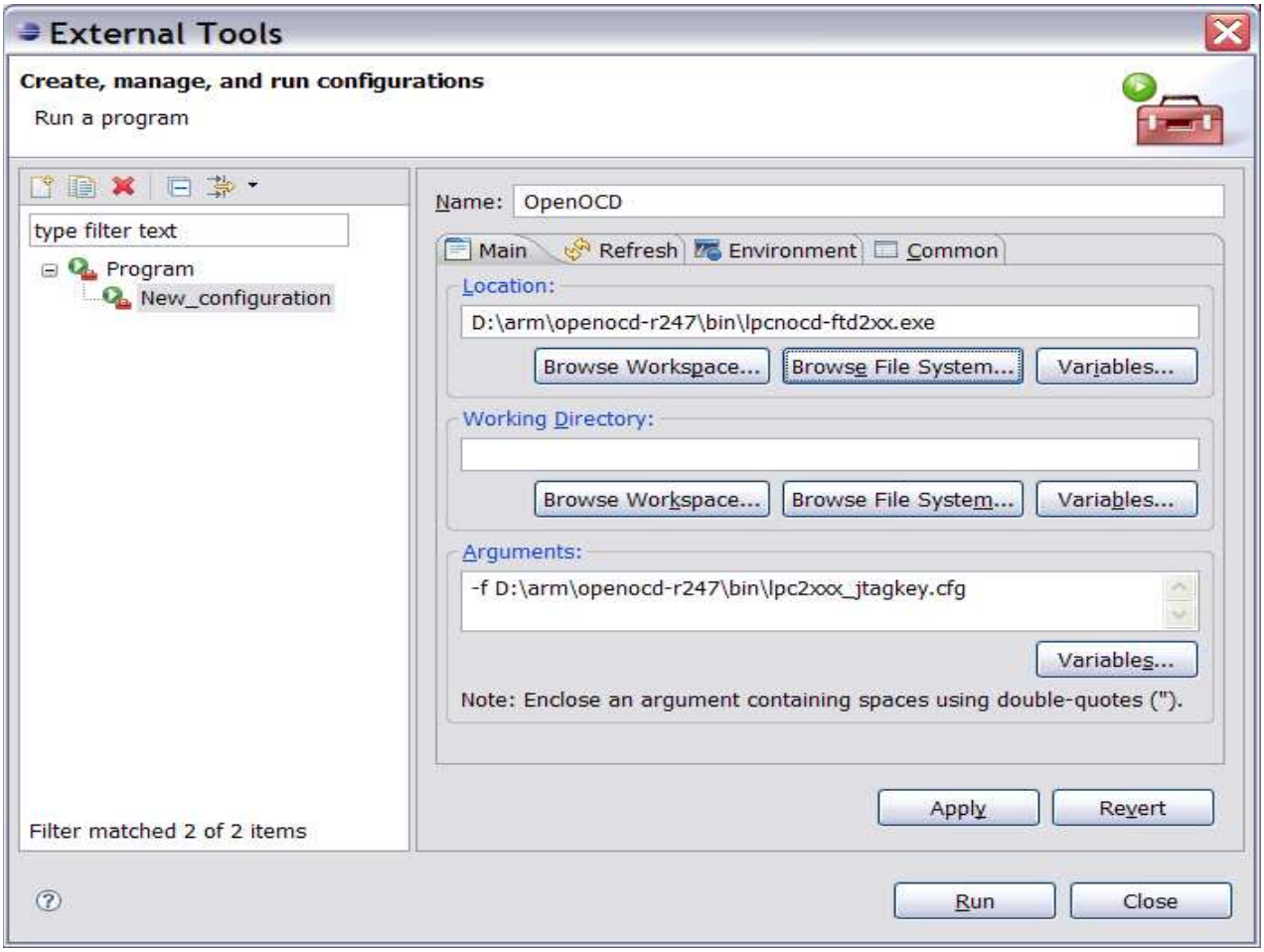
Cette manip n'est pas stritement indispensable (On peut aussi démarrer OpenOcd dans une fenêtre de commande DOS), mais l'installation telle que je vais la décrire apporte beaucoup de confort d'utilisation.

Tout d'abord, pour installer un « server » OpenOcd :

Aller dans le menu « Run » -> « External Tools » -> « Open External Tools Dialog ». Ce qui apparaît ressemble à ceci :



Cliquer une (ou deux fois) sur « Program », il apparaîtra une « New_configuration » qu'il convient de remplir de la manière suivante :



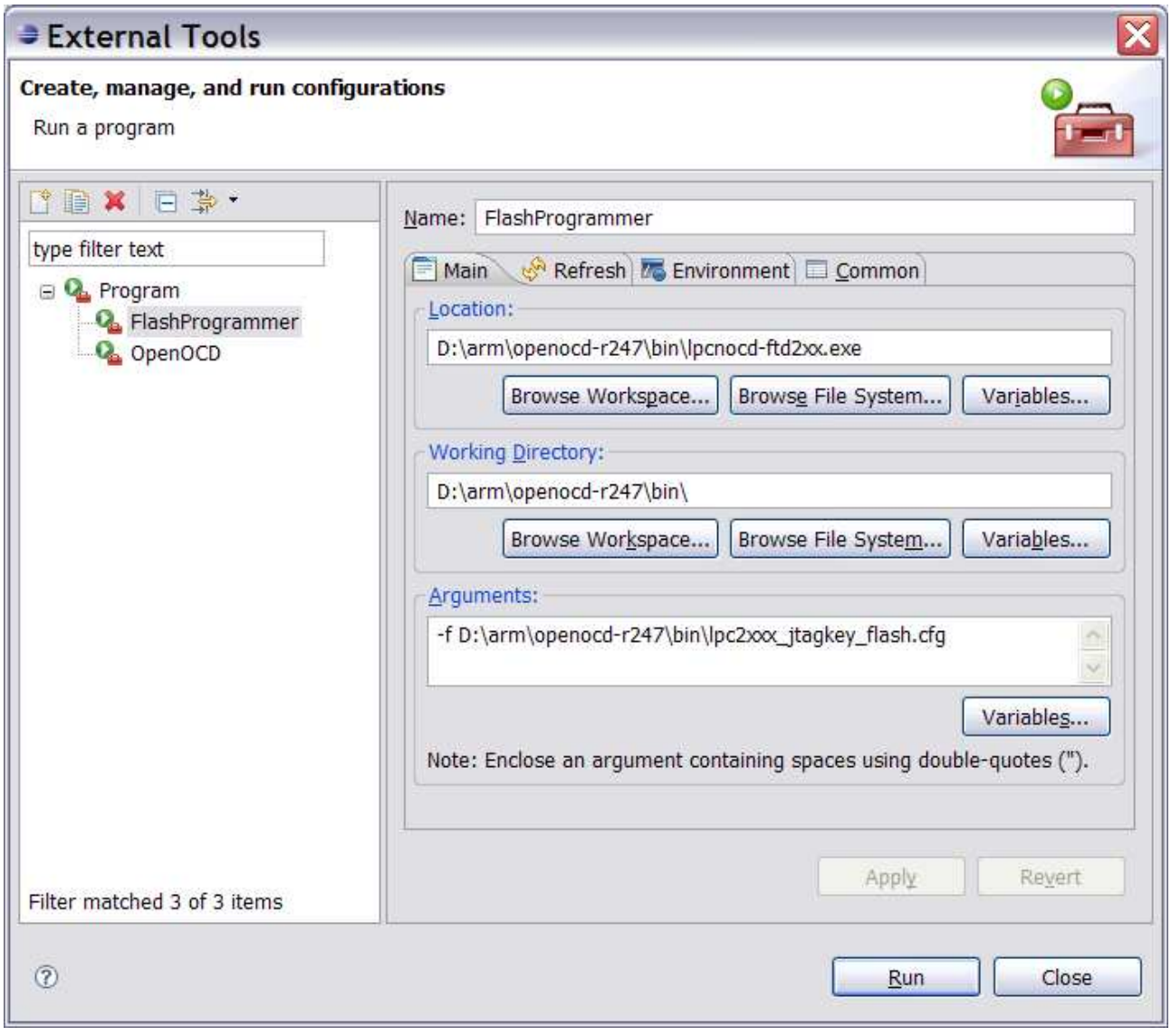
C'est à dire :
Dans « Name », taper « OpenOCD » (la New_confihuration prendra ce nom après clic sur « apply))
Dans « Location », taper le chemin de l'exécutable OpenOCD
Dans « Arguments », introduire -f et le nom avec le chemin du fichier de configuration d'OpenOCD » (ici lpc2xxx_jtagkey.cfg), qui dépend de la sonde (Jtag ou Wiggler) utilisée.

Notes :
J'ai presonnellement installé la chaine GCC/OpenOCD dans mon répertoire **D:\arm** Il conviendra de préciser celui qui est adéquat.

1. J'utilise une sonde Jtag USB (avec FTDI232), il faudra choisir l'exécutable et le fichier de configuration qui convient (différents si Wiggler sur port parallèle)
2. Ma carte ARM est à base de LPC2103. Il faudra également choisir l'exécutable et le fichier de configuration (.cfg) approprié.

Cliquer alors sur « Apply » puis « Close »

Répéter l'opération pour créer un « External Tools » qui programme la Flash (toujours avec OpenOCD) :



C'est apparemment le même que le précédent, sauf que il fait appel à un fichier de configuration différent : lpc2xxx_jtagkey_flash.cfg

Ce fichier est similaire au précédent (lpc2xxx_jtagkey.cfg) sauf :
- qu'il fait appel à un script qui programme le « a.elf » qui se trouve dans D:\arm\tmp
- qu'il coupe le serveur (shutdown) après l'opération

Il contient la ligne supplémentaire :

```
target_script 0 reset D:/arm/openocd-r247/bin/flash.script
```

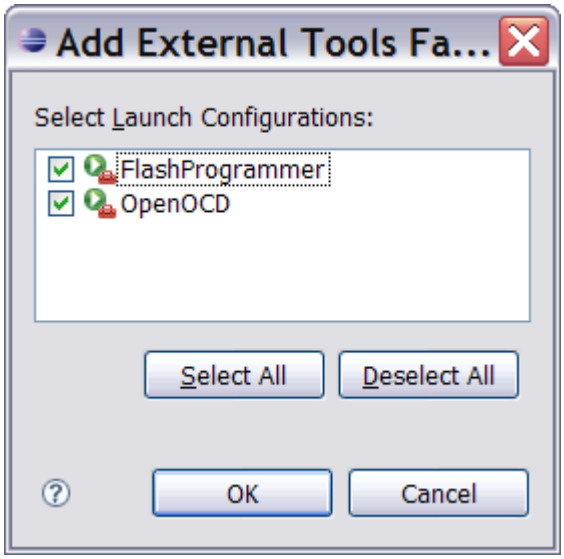
« flash.script » est un fichier qui contient les commandes OpenOCD suivantes

```
arm7_9 dcc_downloads enable
wait_halt
sleep 10
poll
flash probe 0
# erase first bank only:
flash erase 0 0 0
# erase all banks: #26 secteurs pour LPC2138; 8 pour LPC2103
# flash erase 0 0 26

flash auto_erase on
flash write_image D:/arm/tmp/a.elf 0x0 elf

# flash write is deprecated but still available.
# Update to flash write_binary 0 a.bin 0x0
reset run
sleep 10
shutdown
```

Retourner maintenant dans le menu « « Run » -> « External Tools » « Organize Favorites »; dans la fenêtre qui s'ouvre, cliquer « Add », puis dans la nouvelle fenêtre qui s'ouvre, sélectionner OpenOcd et FlashProgrammer :



Cliquer sur « OK » (2 fois).

Femer et relancer Eclipse : OpenOCD et « FlashProgrammer » appraîssent alors directement dans le menu « Run » -> « External Tools »

Il faut maintenant configurer OpenOCD et GDB

Il faut d'abord préparer deux fichiers, qui serviront de fichiers de commandes d'initialisation de GDB :

Un premier, que l'on pourra appeler ***gdbinit_ram*** : contenant

```
target remote localhost:3333
monitor reset
monitor sleep 500
monitor poll
monitor soft_reset_halt
monitor arm7_9 sw_bkpts enable
load
br main
continue
```

Un second, qui servira à debugger des programmes en Flash, que l'on pourra nommer ***gdbinit_rom***, contenant :

```
target remote localhost:3333
monitor reset
monitor sleep 500
monitor poll
monitor soft_reset_halt
monitor arm7_9 force_hw_bkpts enable
```

Ce qui appelle quelques commentaires :

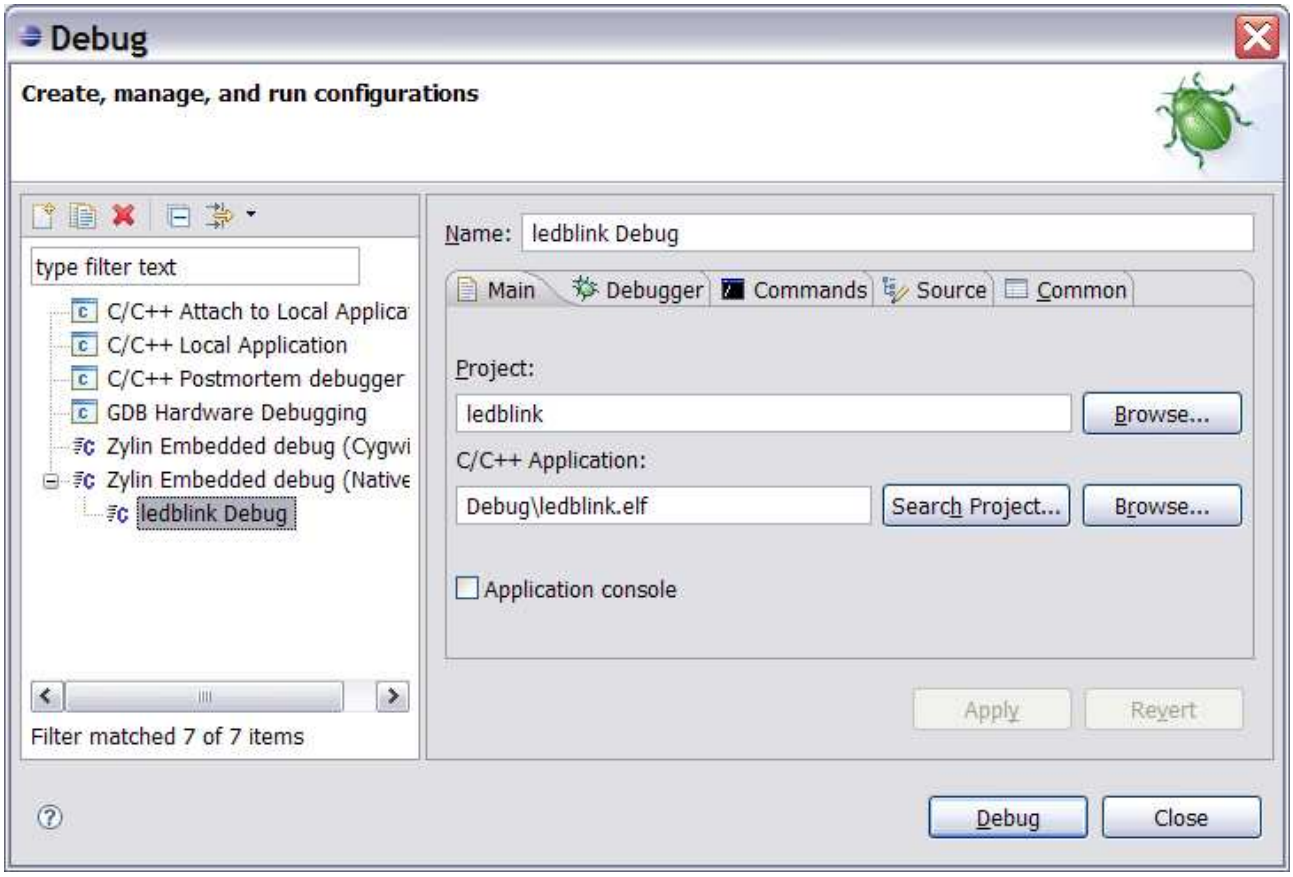
target remote localhost:3333 indique à GDB qu'il doit communiquer avec OpenOCD via le port local 3333.

Les lignes suivantes qui commencent par « monitor » sont des commandes OpenOCD qui lui sont envoyées par GDB

load : charge le programme en RAM
br main : met un « breakpoint dans à la fonction main »
continue : commence l'exécution

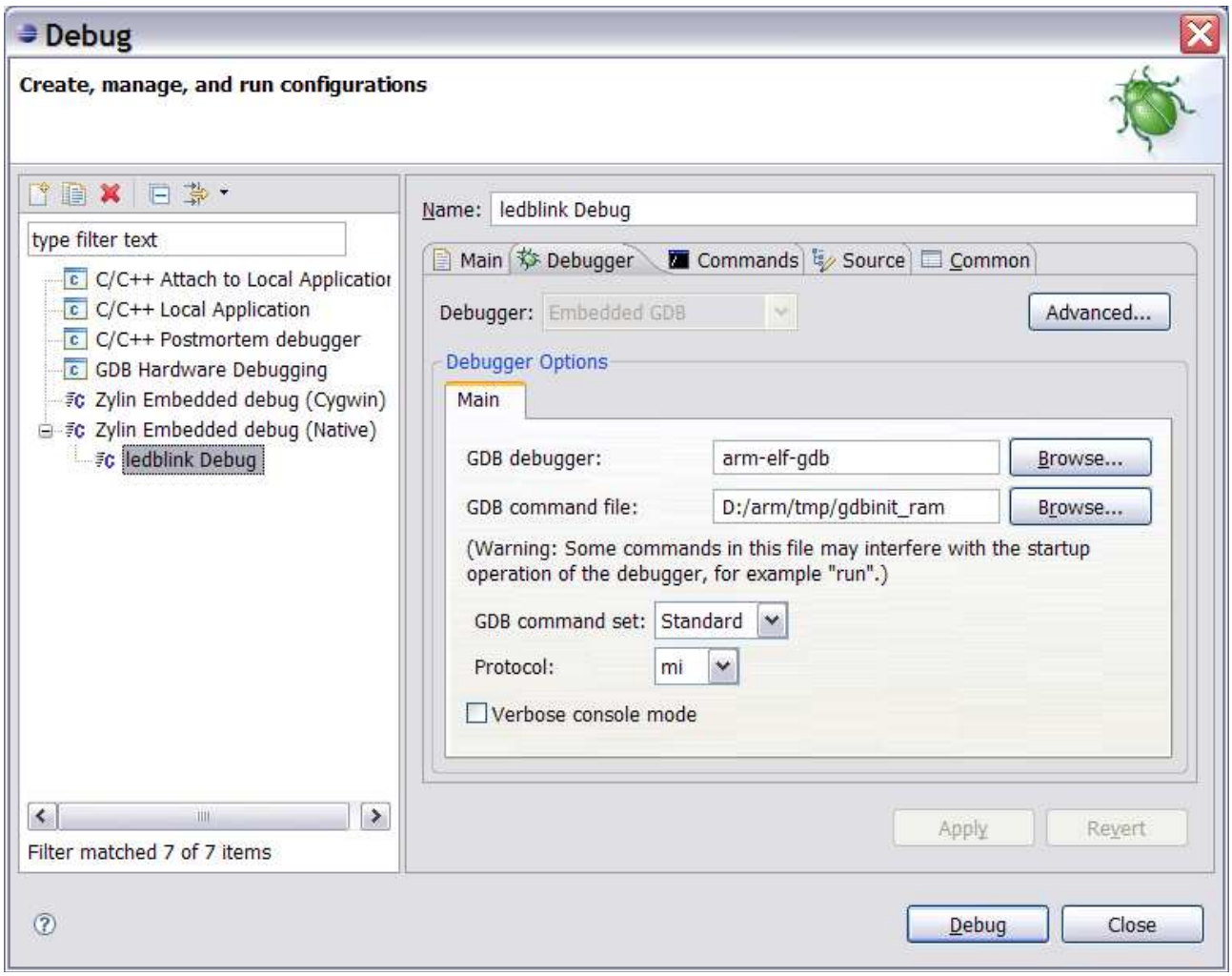
Ces deux fichiers seront placés dans un répertoire approprité (D:\arm\tmp dans mon cas).

Dans le menu principal « Run » ou en cliquant sur la petite flèche (vers le bas) à côté de l'insecte vert, on choisit « Open debug dialog », puis en déroulant (cliquant sur) « Zylin Embedded debug (Native) », il y a une configuration «ledblink debug ».



Debug en RAM :

Il faut maintenant configurer le Debugger , dans l'onglet approprié:



Le debugger s'appelle **arm-elf-gdb**.

Remplacer le « GDB command files » proposé par défaut par le fichier créé précédemment (avec le chemin complet – attention / et pas \ ) : gdbinit_ram .

GDB est maintenant configuré : cliquer sur « Close »

Aller dans le menu « Windows » -> « Open Perspective » -> « Debug »

Maintenant :

- Brancher la sonde JTAG (en supposant, si c'est une sonde USB, que les drivers auront été installés au préalable)
- Mettre la carte d'essai ARM sous tension

Effectuer un RESET de celle-ci (en appuyant sur le bouton approprié)

Lancer OpenOCD : aller dans le menu « Run » -> « External Tools » -> « OpenOCD »

Voici ce qui apparaît dans la console :

```
Info:    openocd.c:93 main(): Open On-Chip Debugger (2007-12-30 17:00 CET) svn:
247
Info:    openocd.c:94 main(): $URL:
http://svn.berlios.de/svnroot/repos/openocd/trunk/src/openocd.c $
Info:    jtag.c:1291 jtag_examine_chain(): JTAG device found: 0x4f1f0f0f
(Manufacturer: 0x787, Part: 0xf1f0, Version: 0x4)
Warning: arm7_9_common.c:742 arm7_9_assert_reset(): srst resets test logic, too
```

OpenOCD a bien trouvé la sonde JTAG. Si échec, ré-appuyer sur le bouton reset de la carte ARM et recommencer.

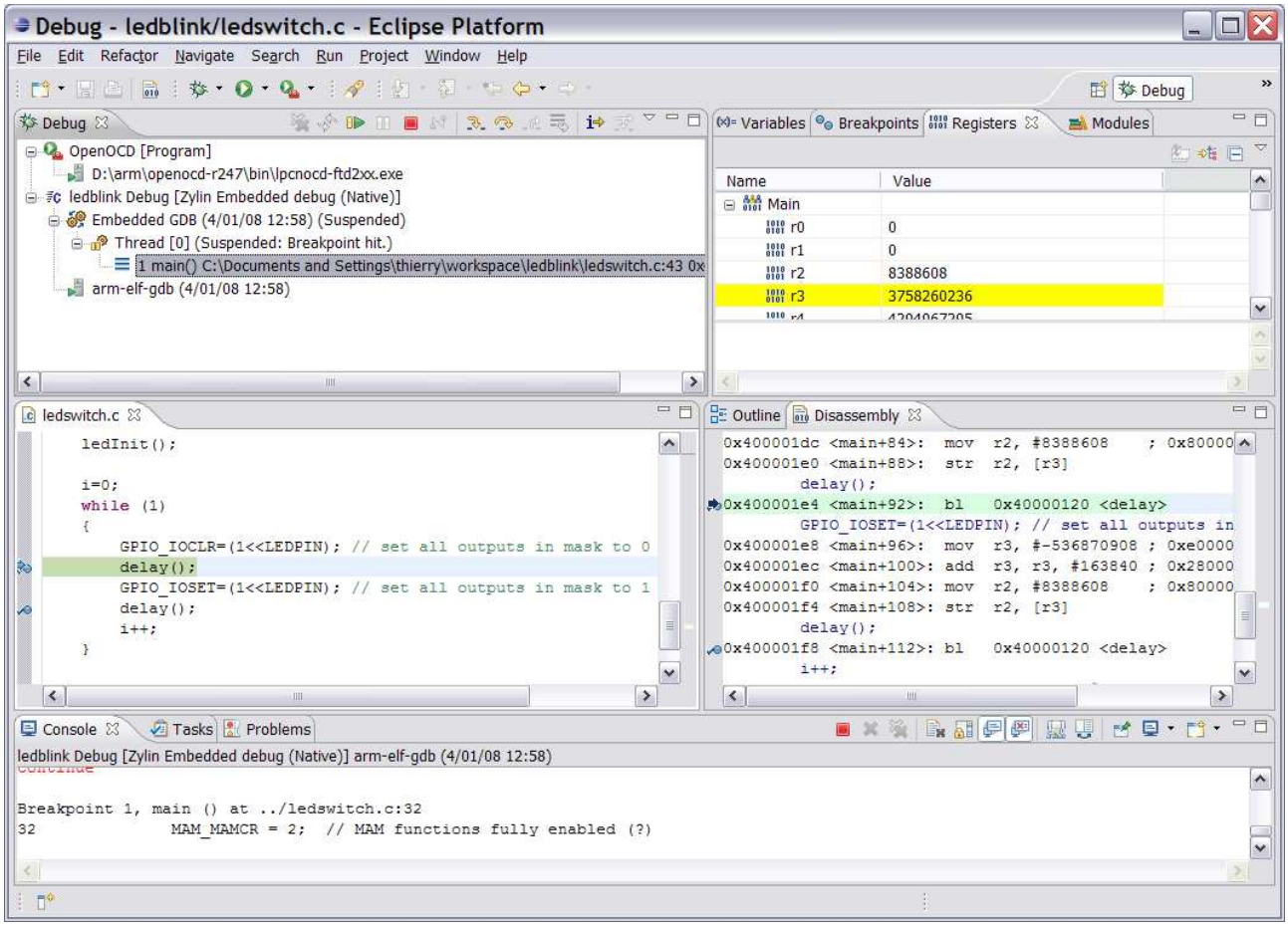
Retourner dans le menu « Run » -> « Open Debug Dialog » et cliquer maintenant sur le bouton « **Debug** », afin de lancer GDB

Voici ce qui apparaît ensuite dans la « Console » (c'est GDB qui effectue le chargement du programme en RAM) :

```
source D:/arm/tmp/gdbinit_ram
0x0000014c in ?? ()
target state: running
requesting target halt and executing a soft reset
software breakpoints enabled
Loading section .text, size 0x214 lma 0x40000000
Start address 0x40000050, load size 532
Transfer rate: 16 KB/sec, 532 bytes/write.
Breakpoint 2 at 0x400001a0: file ../ledswitch.c, line 32.

Breakpoint 2, main () at ../ledswitch.c:32
32          MAM_MAMCR = 2;    // MAM functions fully enabled (?)
```

La suite relève de l'utilisation du debugger GDB. Il est par exemple possible de mettre un « Breakpoint » en positionnant la souris devant une ligne C et en cliquant avec le bouton droit, dans le menu contextuel « toggle breakpoint », exécuter pas par pas, examiner (et modifier) les registres, les variables, la mémoire, l'asm, ... avec les différents boutons ou même d'introduire directement des commandes GDB dans la « console ». La sonde JTAG réagit instantanément.



Les différentes fenêtres s'appellent des Vues « Views ». Il est possible d'en ajouter (ou enlever) via le menu « Window » - « Show View ». On peut également les redimentionner (agrandir), et finalement, arranger l'écran comme l'on veut.

Pour arrêter le debug, cliquer sur «ledblink debug » puis, avec le menu qui apparaît avec un clic du bouton droit : « Terminate and remove ».

Pour repasser à l'écran C du compilateur, aller dans le menu « Window --> Open Perspective -> C/C++. Si, par mégarde, on ferme une ou plusieurs fenêtres de la « Perspective », toujours dans le même menu Window, effectuer : « Reset Perspective ».

Debug d'un programme en Flash :

Ne pas oublier de changer le script du linker avant de faire le build

L'étape supplémentaire consiste à flasher d'abord le µC :

Une fois le buid effecté, aller dans le menu : « Run » - « External Tools » « FlashProgrammer »

Ceci apparaît dans la « console » :

```
Info:    openocd.c:93 main(): Open On-Chip Debugger (2007-12-30 17:00 CET) svn:
247
Info:    openocd.c:94 main(): $URL:
http://svn.berlios.de/svnroot/repos/openocd/trunk/src/openocd.c $
Info:    jtag.c:1291 jtag_examine_chain(): JTAG device found: 0x4f1f0f0f
(Manufacturer: 0x787, Part: 0xf1f0, Version: 0x4)
Warning: arm7_9_common.c:742 arm7_9_assert_reset(): srst resets test logic, too
```

La programmation flash n'apparaît pas explicitement.
Switcher vers la Perspective Debug (Windows -> Open Perspective –> Debug)

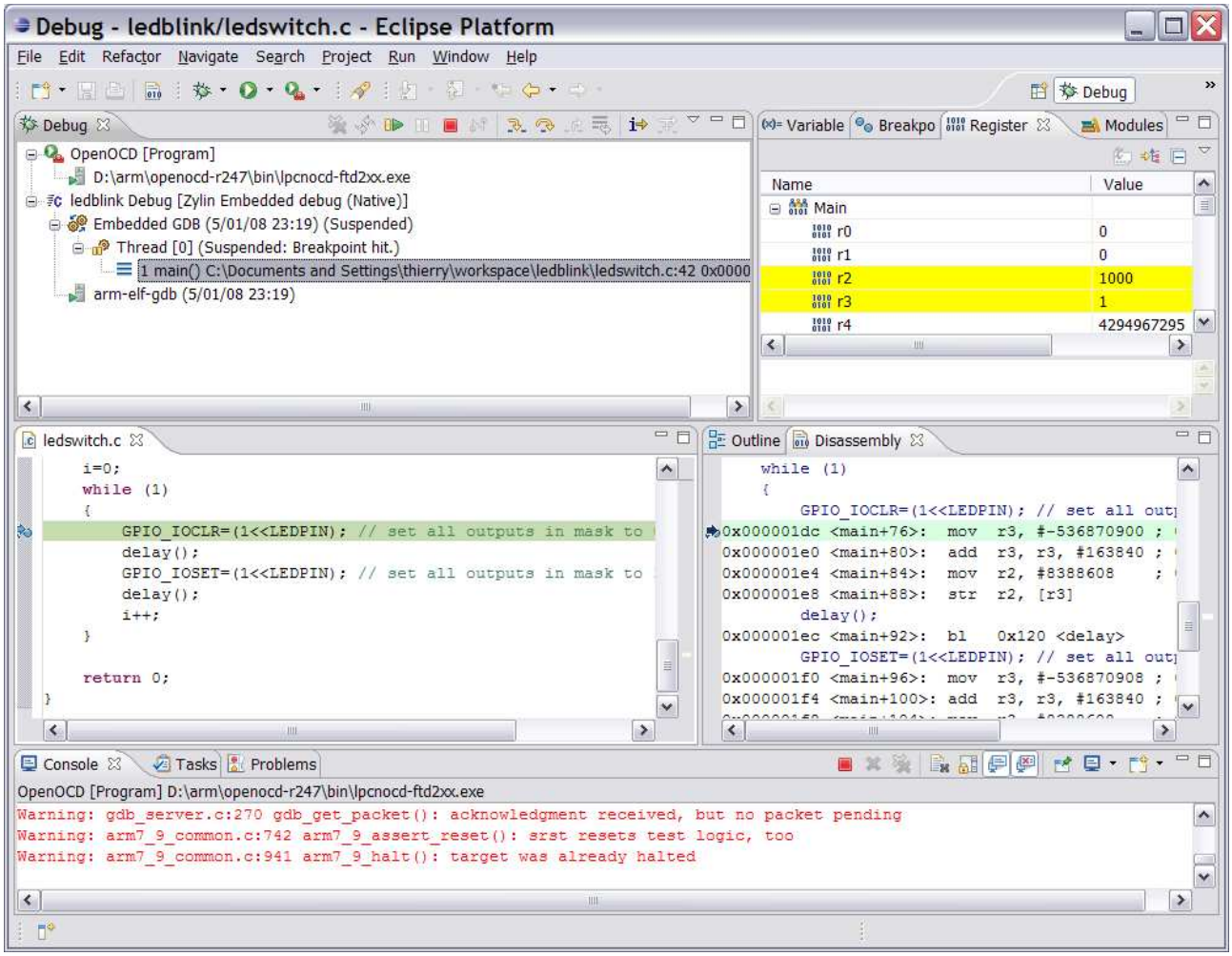
On voit le flash programmer qui est dans la fenêtre Debug. Cliquer sur celui-ci avec le bouton droit de le souris et dans le menu contextuel : « Terminate and Remove ».
Il faut effectuer cette opération avant de relancer OpenOCD (sinon, il mettra un message d'erreur comme quoi on ne peut pas utiliser deux sondes en même temps)

Comme précédemment, lancer maintenant le serveur OpenOCD : aller dans le menu « Run » -> « External Tools » -> « OpenOCD »

Ouvrir le menu « Run » -> « Open Debug Dialog », et dans l'onglet Debugger, changer gdbinit_ram par **gdbinit_rom**

Une fois que cela est fait : cliquer sur le bouton « Debug » , GDB se lance. Tout est OK :

Voir la copie d'écran ci-après : le programme est bien en Flash (adresses 0x0000.....).



Dernières remarques et conclusion

Tout ceci a l'air bien compliqué, mais avec un peu d'habitude, cela devient simple et évident.
Il doit être possible aussi de créer des projets « template », afin d'éviter toutes les étapes de configuration.

- Eclipse est écrit en Java, donc son exécution est assez lente. Il faut donc un PC puissant; pas la peine d'essayer avec un Pentium 2 à 400MHz.
- Tout ce que j'ai décrit ci-dessus est parfaitement réalisable sous Linux. Télécharger les versions appropriées.
- Il y a d'autres programmes inclus dans la distribution GCC :
  - arm-elf-objcopy : permet de convertir les exécutables en différents formats (Bin, Hex, S19 par exemple).
  - arm-elf-objdump : dump des symboles des objets, désassemblage,..
  - etc..
- La plupart des manipulations via les menus que j'ai décrites sont accessibles via des icones d'eclipse (par exemple le switch entre perspectives, le lancement du debug,..). Il faut cependant une certaine habitude, mais c'est nettement plus rapide.
- L'utilisation de arm-elf-gcc sans eclipse est parfaitement possible sous DOS, en ligne de commande et avec un éditeur standard pour les fichiers source.

- Eclipse peut être utilisé (simultanément) avec tout autre microcontrôleur supporté par GCC. J'utilise Eclipse/GCC (m68k-elf-gcc) simultanément pour des projets ColdFire (Freescale). Il suffira « juste » de paramétrer adéquatement le projet (et éventuellement utilser un « workspace » différent).

Pour approfondir GCC/GDB, beaucoup de docs sont librement téléchargeables sur internet : http://www.gnu.org/manual/

Pour comprendre le développement en C sous GCC, une compréhention basique de MAKE est utile.

Finalement, les pages précédentes résultent de longues semaines de galère et de recherche. Sans tutorial clair, Eclipse n'est pas simple ni intuitif, avec de très nombreux menus tout aussi tortueux les uns que les autres, mais il faut avouer que réussir à debugger son premier programme avec GDB rend aussi euphorique que le premier clignottement de led.

Liens ARM -GCC -OpenOCD, ou l'on pourra par exemple trouver modes d'emploi, différents fichiers de configurations OpenOCD pour différents ARM (LPC, ATMEL, STR7,..) ainsi que des projets exemples :

http://www.yagarto.de/ : Chaine GCC/GDB
http://openfacts.berlios.de/index-en.phtml?title=Open+On-Chip+Debugger OpenOCD
http://www.siwawi.arubi.uni-kl.de/avr_projects/arm_projects/ : ARM projects by Martin Thomas: La caverne d'ali-baba
http://www.siwawi.arubi.uni-kl.de/avr_projects/arm_projects/openocd_intro/index.html idem, accessing ARM controllers with OpenOCD.
http://wiki.jelectronique.com/at91/openocd : utilsation OpenOCD avec AT91 et Wiggler (français)

thierry