Using Open Source Tools for STM32 Cross Development under Ubuntu Linux.

Revision 1

Author:
Michel Catudal
mcatudal@comcast.net
February 14th 2009

Introduction

For those on a limited budget, use of open source tools to develop embedded software for the ARM devices is the only way. The goal of this document is to help those interested in doing development with Linux by making their installation of an **ARM** development system a bit easier. So far most of the information for ARM development is for windows while the Linux users are ignored. For development on Linux you are usually on your own.

The goal of this document is not to do everything for you but to make your life easier. You still need to be somewhat familiar with your Linux environment to use this document. This document describes an installation on **ubuntu 8.10 Linux.** The favorite development environment appears to be eclipse. Ubuntu 8.10 uses an old version of Eclipse. The latest release of eclipse is 3.4 and the latest development code is 3.5. This document will use version 3.5, either versions 3.3 or 3.4 could be installed. I do not provide plugins for 3.2 and have no plan to do so. I have plugins for the latest 3 versions.

For the debugging part you will need the Zylin plugin because even the latest CDT has some serious bugs in regard to debugging embedded ARM7 or Cortex devices. The Zylin source had to be patched because of the missing simulator would keep it from being compiled. I have removed the useless windows stuff.

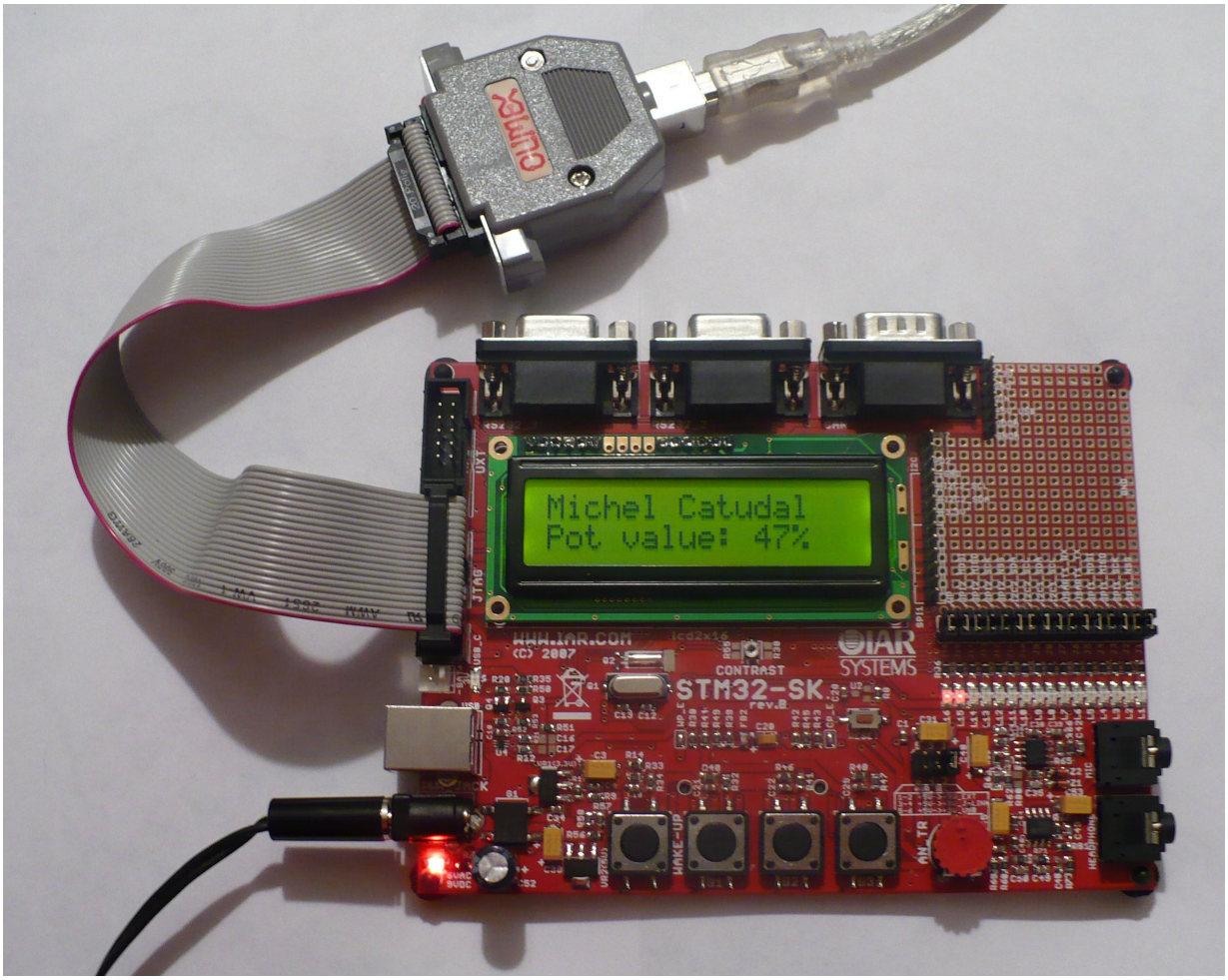
For development you will need a set of tools. The package that I provide fully supports ARM7TDMI and cortex-m3 devices such as STM32 and Luminary.

For this demo I use a board from Olimex designed for IAR called **STM32-SK**.

If you need to rebuild the packages you need to note that in order to compile GCC you need to have it installed already as well as newlib. A work around would be to create a bootstrap version first. This is a bit more complicated.

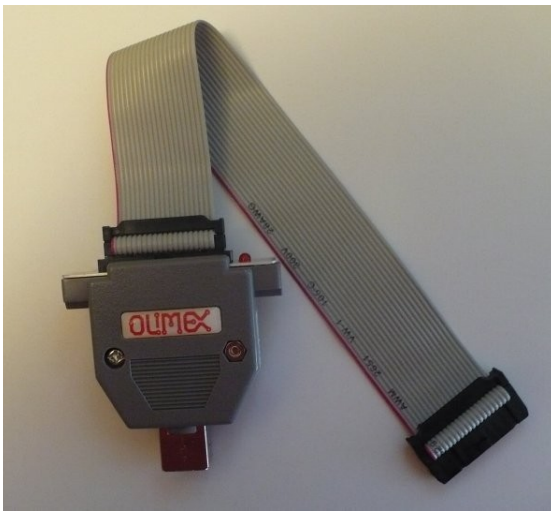
The binaries were compiled under the 32 bits version of ubuntu 8.10. They will work on any recent debian based system.

Hardware Information



As a hardware platform to exercise our ARM cross development tool chain, we will be using the **STM32-SK** evaluation board from Olimex at the cost of around $250.00. You can purchase this board at digikey , future or some other distributor of STMicros. Olimex sells a slightly different board which is somewhat cheaper. You can get some cheaper boards at MicroController Pros.as well. Type **STM32** in the search box. You will see several interesting boards using the **STM32** processor.
The **STM32-SK** board includes two serial ports, a USB port, 1 CAN port, 16 LEDs, a LCD display, a potentiometer connected to an A/D port, a connector for a microphone and one for a headphone. It also has a connector in the back for a flash SD drive. It has standard JTAG connector for debugging.
This board comes with a limited version of the **IAR** Compiler. The Full version is very good but very expensive. The **ARM GCC** Compiler compares well against **IAR** compiler and is open source. For commercial projects you should seriously look into buying **IAR**.

With the board comes a programmer from IAR, it works nicely with windows but the **OpenOCD** support for Linux is very new and operation is rather slow compared to the competitors. This is likely to change with time.
Olimex has a few **USB** devices that you can buy at **MicroController Pros**



The device used with this tutorial is the **ARM-USB-TINY** which costs $41.00. You could use something different like the **ARM-USB-JTAG** which costs $72.00. Unless you need the extra serial port and need the power supply this device has no real advantage over the tiny device.

Open Source Tools

To install an **ARM** cross development tool chain, we need the following components:

1. Eclipse IDE version 3.3, 3.4 or 3.5
2. Debugger plugin (derivative of zylin plugin)
3. Modified version of the gnuarm eclipse plugin.
4. Binutils, GNU C++/C Compiler and newlib for arm-elf targets
5. Latest OpenOCD for JTAG debugging

Here is a list of some of the files available at Catudal Software.

1. arm-elf-binutils-2.19_020109-1_i386.deb
2. binutils-cvs-020109.tar.bz2
3. arm-elf-gcc-4.4.0_013109-1_i386.deb
4. arm-elf-gcc-4.4.0-locales_013109-1_all.deb
5. gcc-svn-013109.tar.bz2
6. arm-elf-newlib-cvs_020109-1_i386.deb
7. newlib-cvs-020109.tar.bz2
8. arm-elf-gdb_6.8-1_i386.deb
9. arm-elf-gdb-6.8.tar.bz2
10. libftd2xx_0.4.16-4_i386.deb
11. openocd-0.1.0_0.1.0-1_i386.deb
12. openocd-0.1.0.tar.bz2
13. arm-elf-gcc-plugin-eclipse-3.3.2.tar.bz2
14. arm-elf-gcc-plugin-eclipse-3.4.tar.bz2
15. arm-elf-gcc-plugin-eclipse-3.5.tar.bz2

The locales file is only needed if you want the message to be in your native language if said language is not english. The plugin chosen will depend on which version of eclipse you use.

Installation of C Compiler, utilities and newlib

You need to open a terminal window. You can either su root or use sudo to install the applications.

You do a cd to the directory where the files have been downloaded and install them

```
sudo debpkg -i libftd2xx_0.4.16-4_i386.deb
sudo debpkg -i openocd-0.1.0_0.1.0-1_i386.deb
sudo debpkg -i arm-elf-binutils-2.19_020109-1_i386.deb
sudo debpkg -i arm-elf-gcc-4.4.0_013109-1_i386.deb
sudo debpkg -i arm-elf-gcc-4.4.0-locales_013109-1_all.deb
sudo debpkg -i arm-elf-newlib-cvs_020109-1_i386.deb
sudo debpkg -i arm-elf-gdb_6.8-1_i386.deb
```

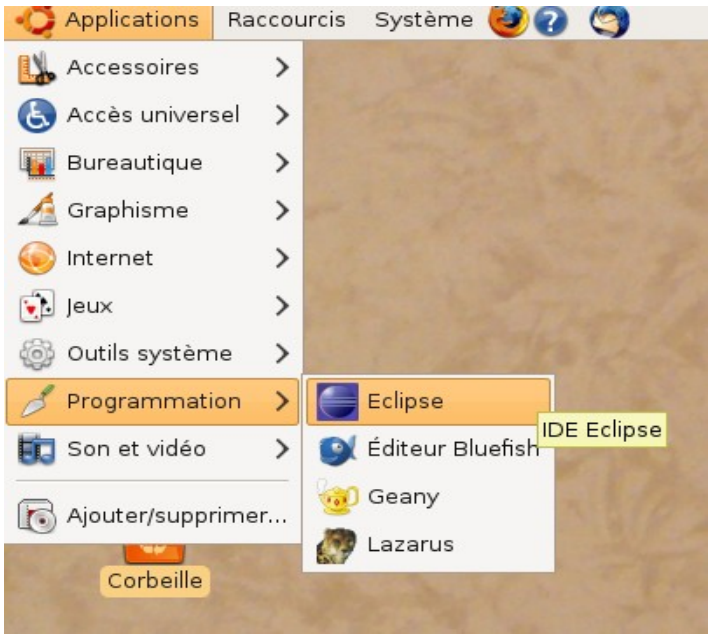
Installation of Eclipse

The version of eclipse that comes with ubuntu 8.10 is too old to be usable with the latest embedded development. You need to first remove the version that is installed. You could also choose to compile your own version. It should be at least version 3.3. The latest release is 3.4.

You can download version 3.4 here. The wed address is http://www.eclipse.org/downloads/

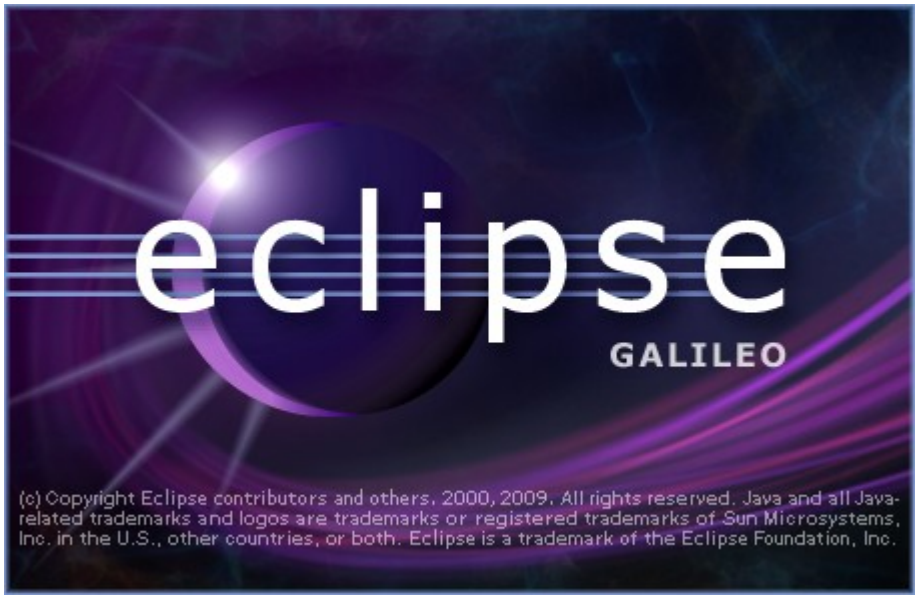
The version we will describe here is version 3.5. This version is perfect for C or C++ development. If you prefer a more complete version use the release version 3.4. Take note that any issues found in the C C++ plugin are more likely to have been resolved in 3.5.
If you choose a different version make sure that you pick the right plugins as they all have the the same names but are compressed in files with names that include the eclipse version numbers. The latest eclipse SDK as of Feb 8th is version 3.5M5 .
The web address is http://download.eclipse.org/eclipse/downloads/

Download and decompress the packages on the root your user account. If you have several users you may want to put it on /usr/local. The next step is to create an entry in the menu to make it easier to call it. The file to run is named eclipse, on the eclipse directory.

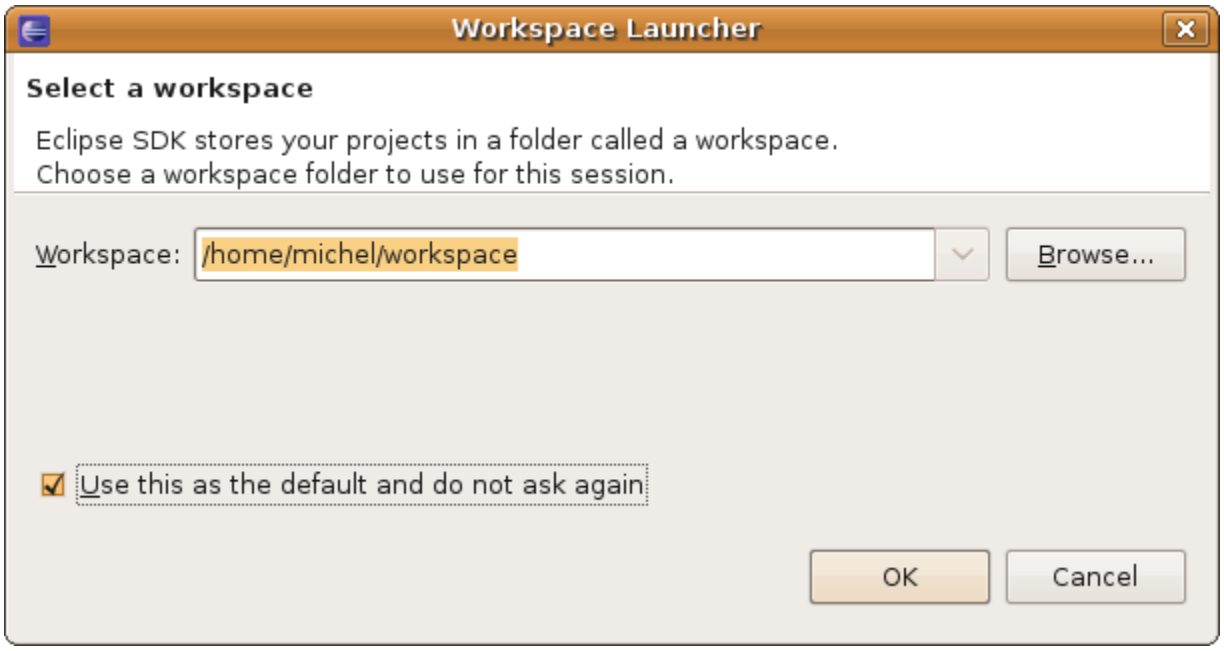


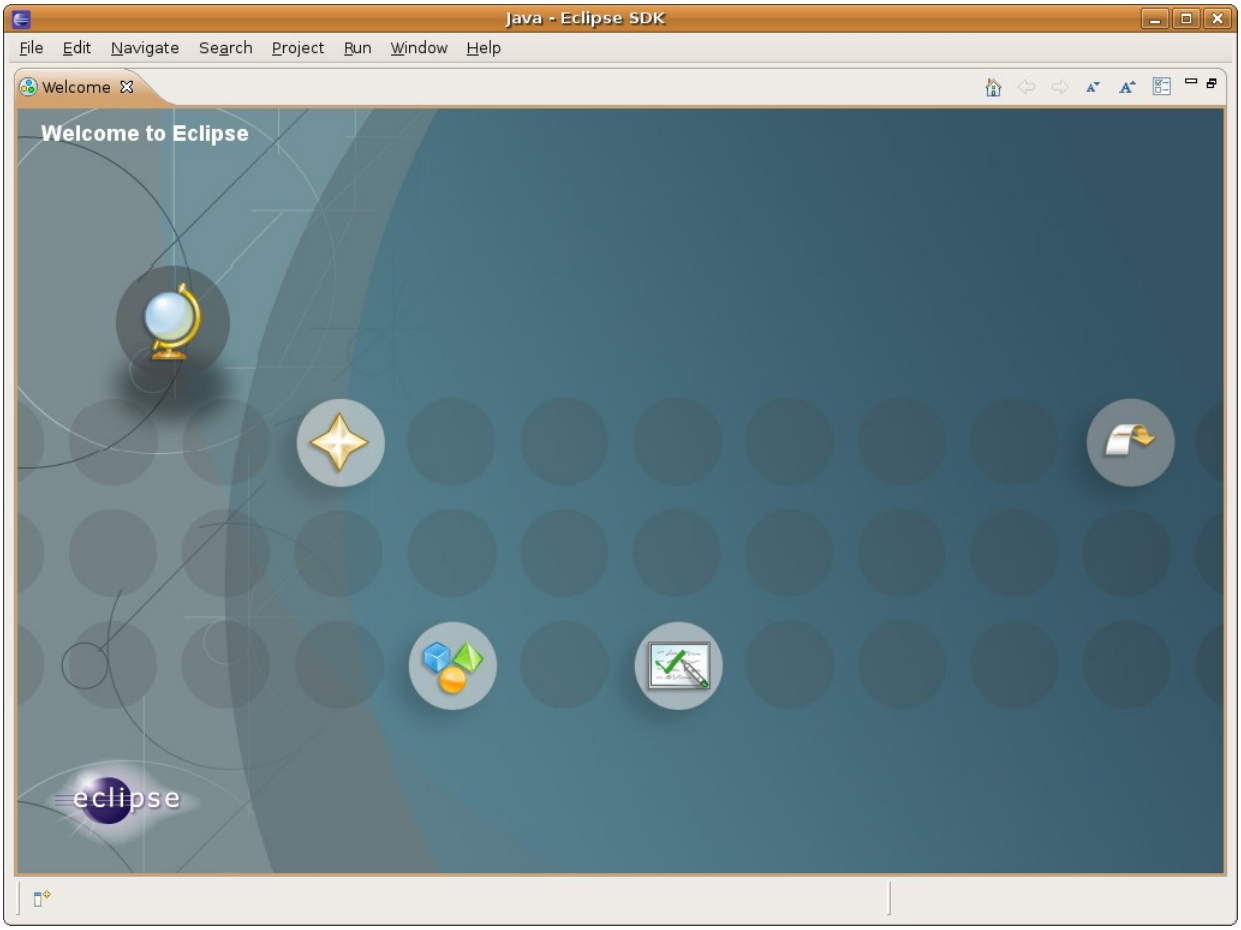
Start Eclipse using the menu.

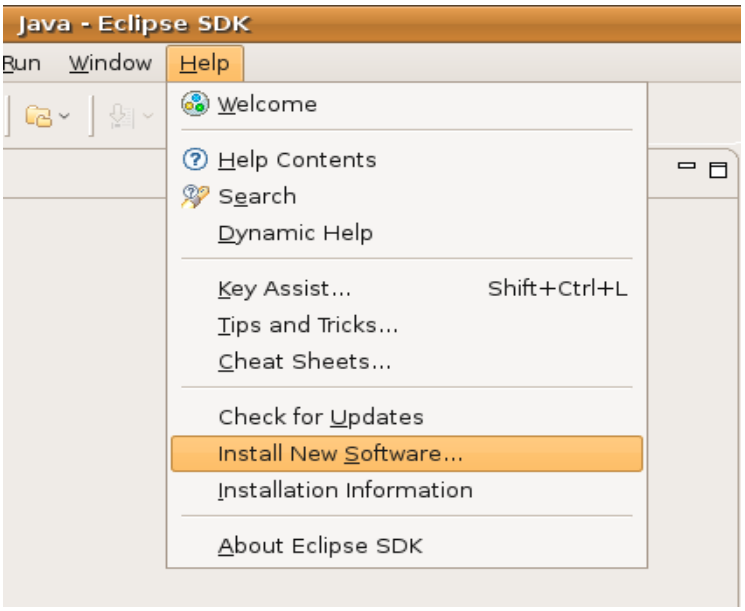
The Eclipse splash screen opens up as shown here.



The first time that you run eclipse or if you have never selected a default workspace it will bring this menu. To accept the default workspace select “Use this as the default and do not ask again”.

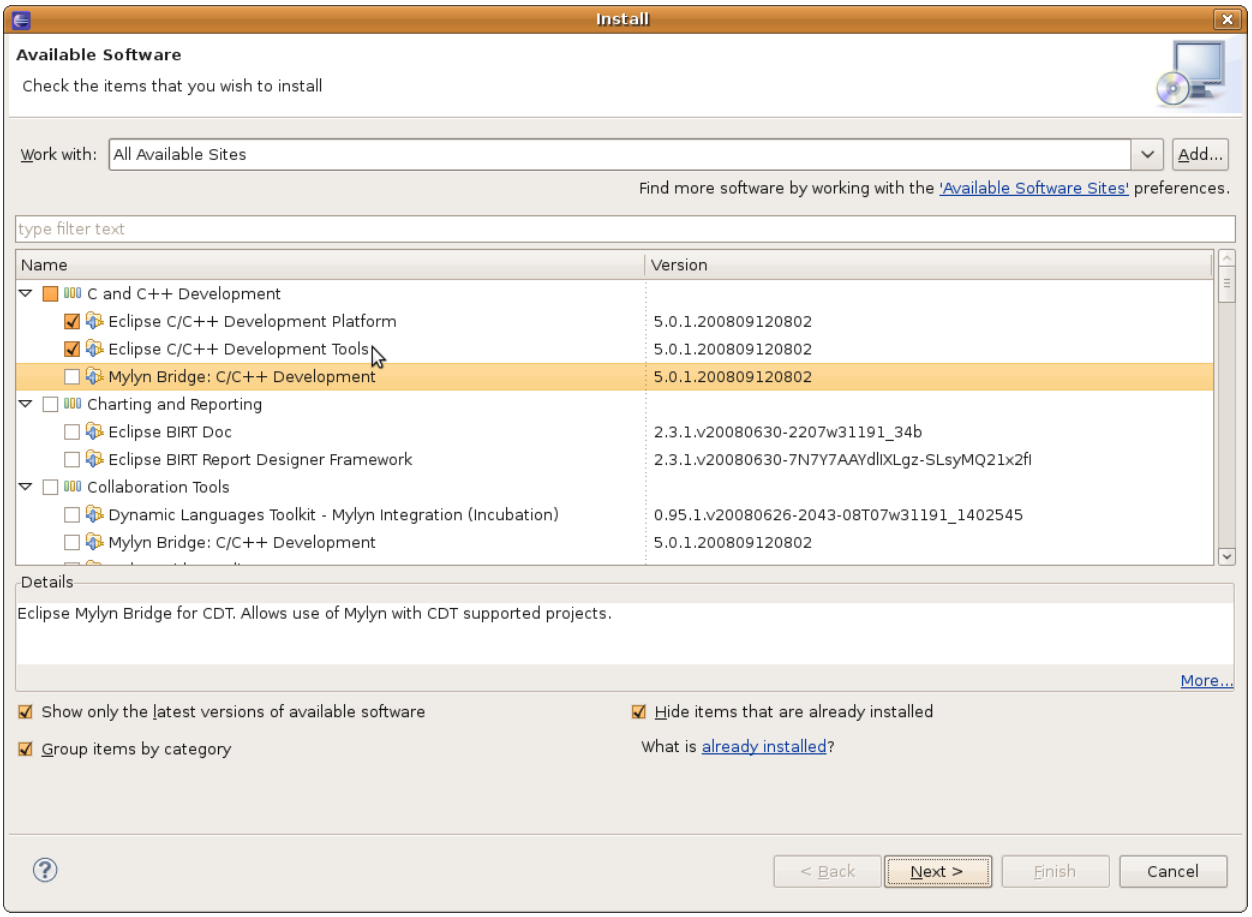




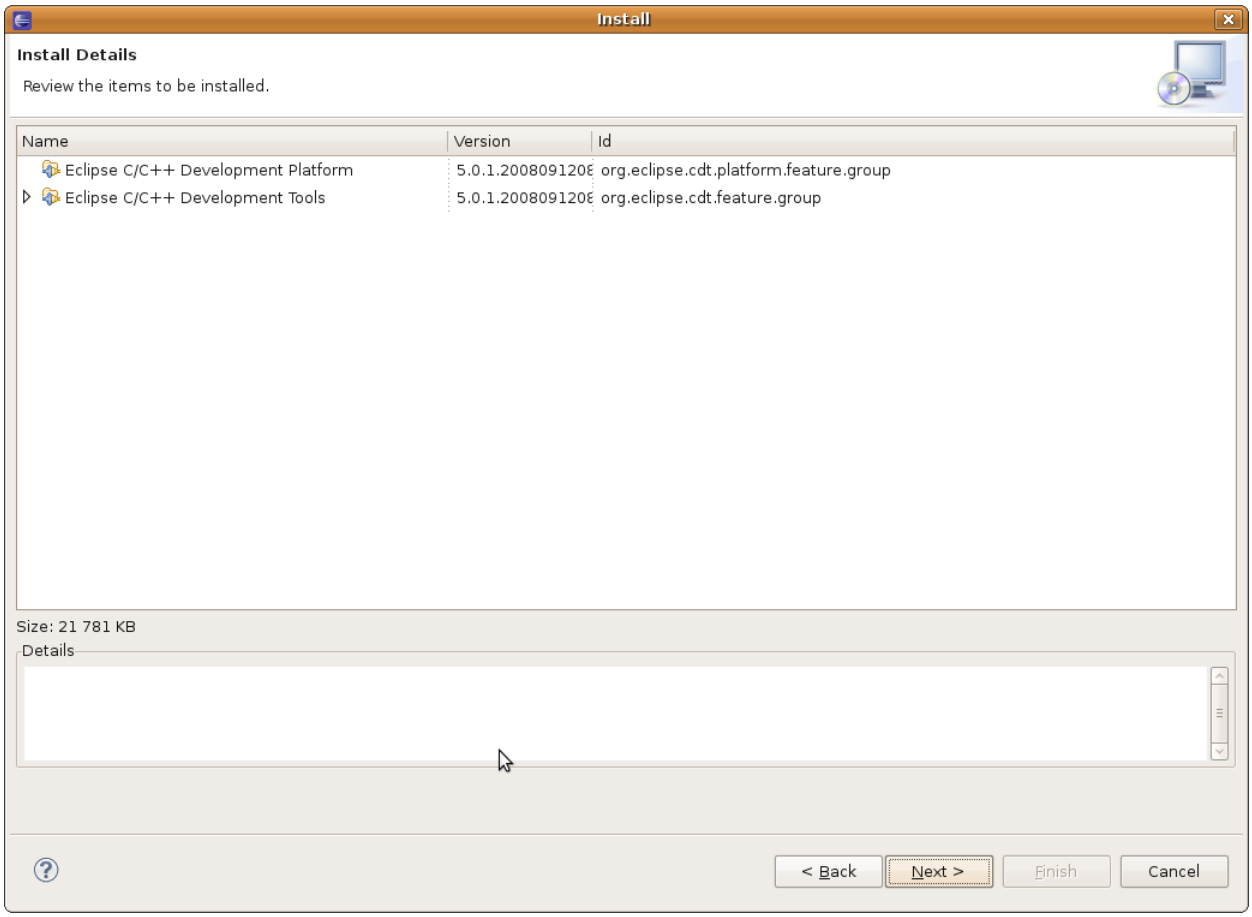


The default SDK doesn't have support for C and C++ so we need to install it.

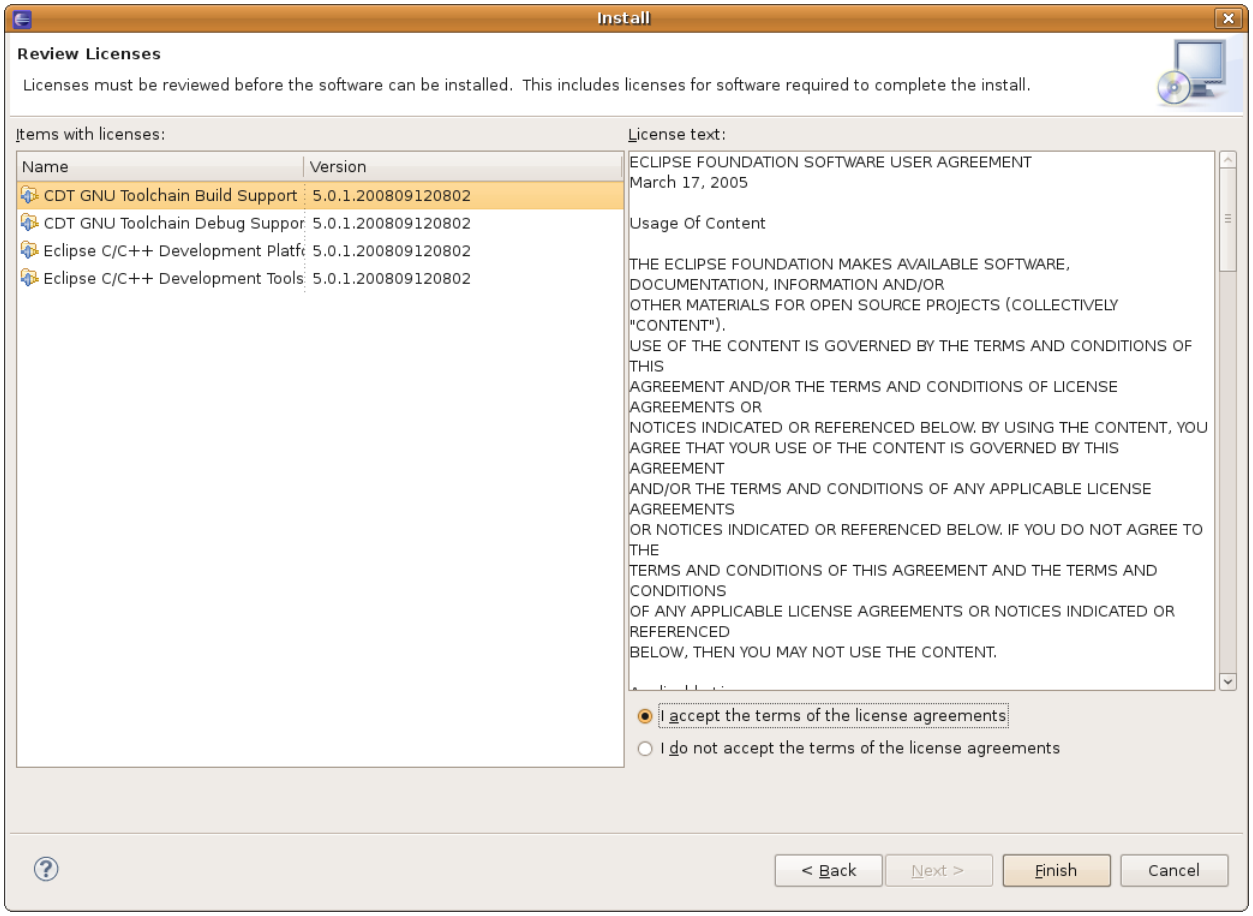
We only need to add support for C and C++. Support for java and plugin development is already installed. Click on Next

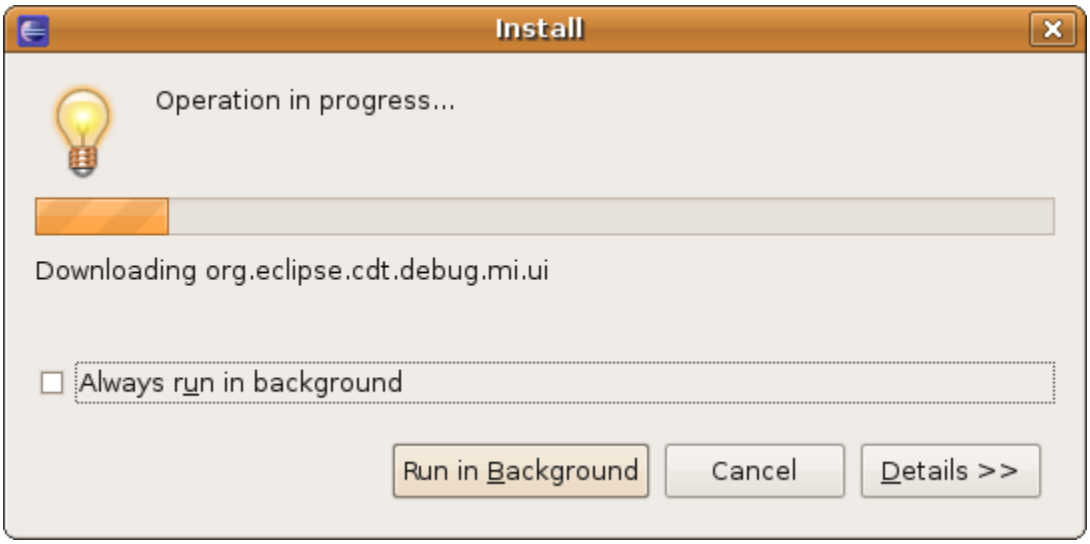


Click on Next to accept the installation.



It would be a good move to accept the terms. If not the install will be canceled. Click on Finish to complete the installation.

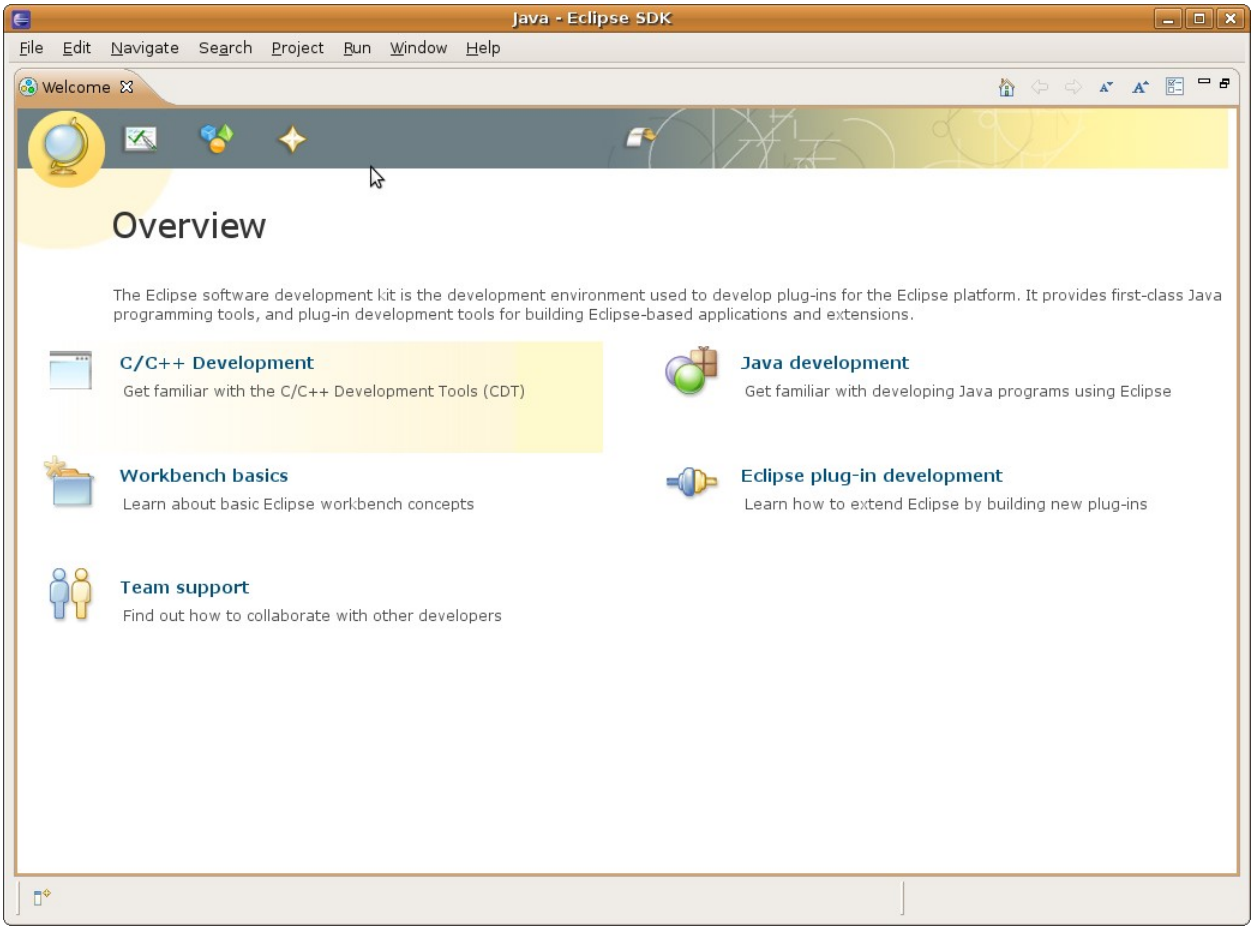




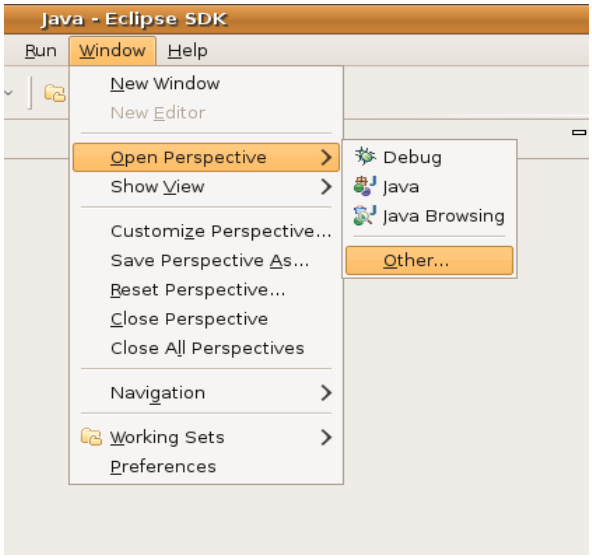
We now close Eclipse because we need to install a plugin which will handle the project creation that we will use for the arm-elf tools. It is important to choose the correct plugin for the Eclipse version installed. All three packages have files with identical names. The plugins compiled for one version of eclipse will not work in a different version.

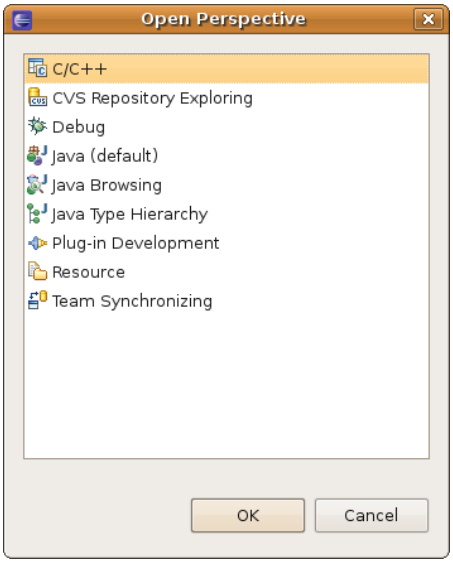
You need to copy the two plugins to the eclipse plugins directory.
The plugin package includes two plugins, one for ARM development and the other one a modified version of the Zylin plugin. The Zylin plugin is cleaned up of the windows and mac parts and fixed so it would actually work with Linux.

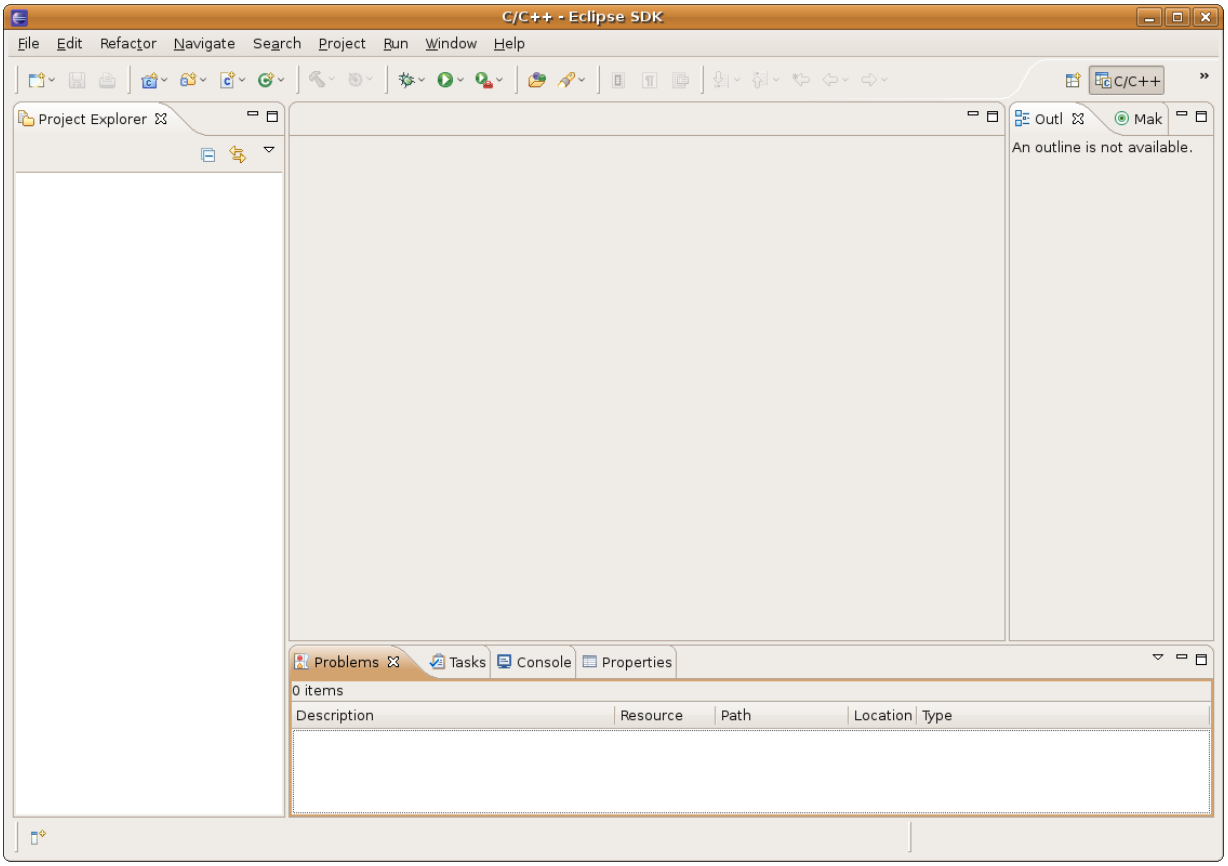
When you open eclipse again you will get an other welcome menu which you just have to remove.



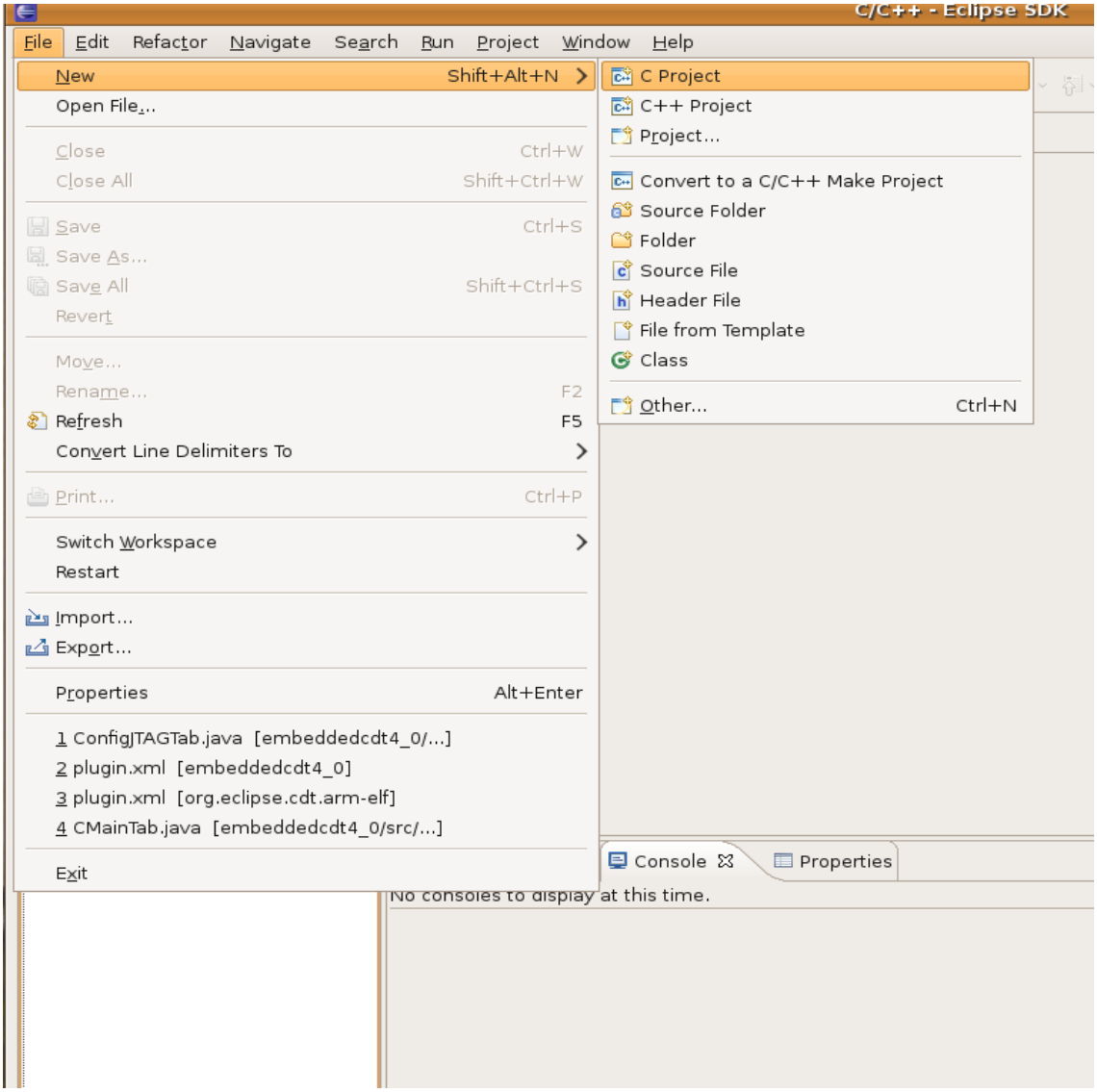
Click on “Window – Other”. You then get an option of which perspective you want to open. Choose C/C++.



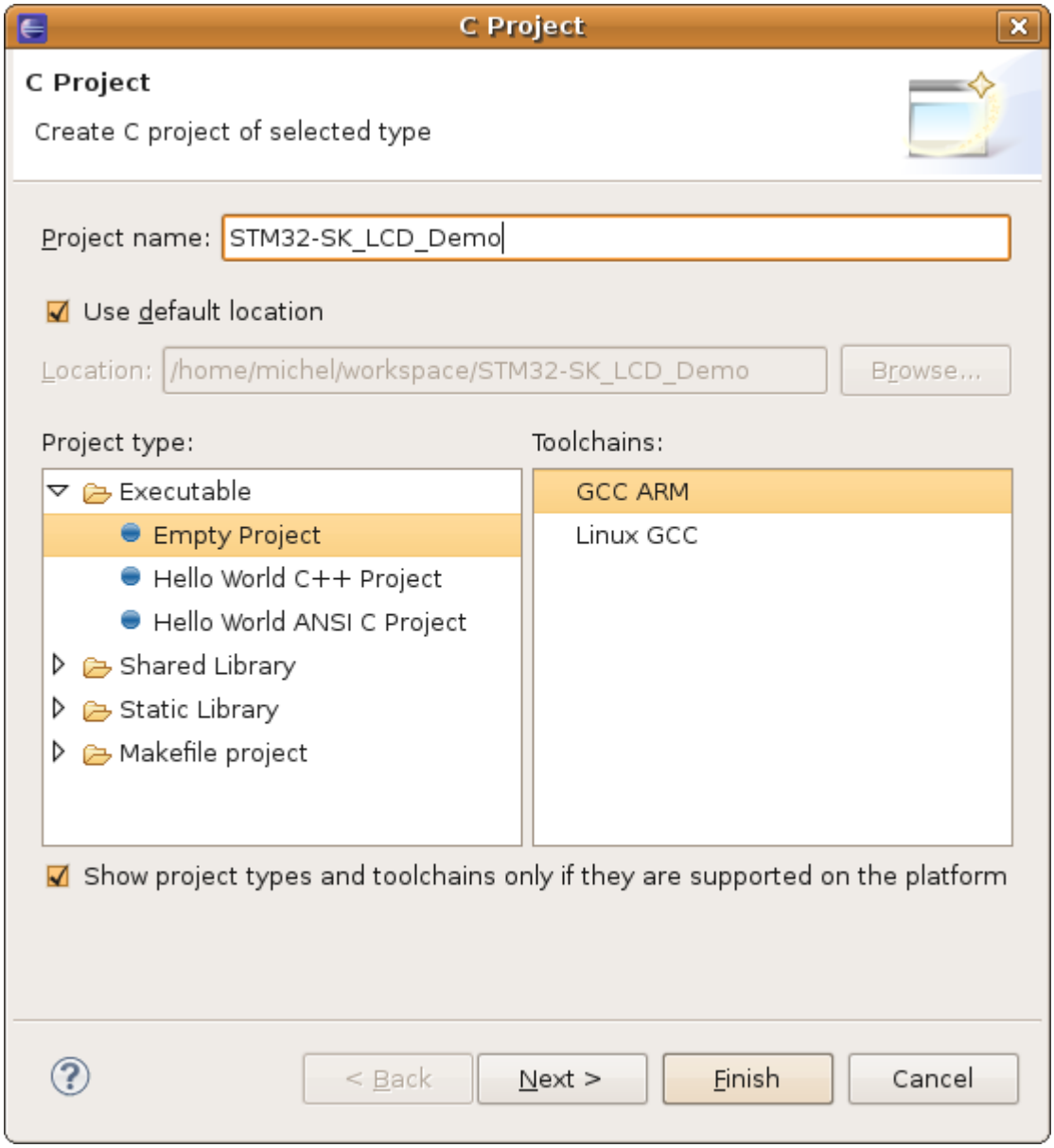




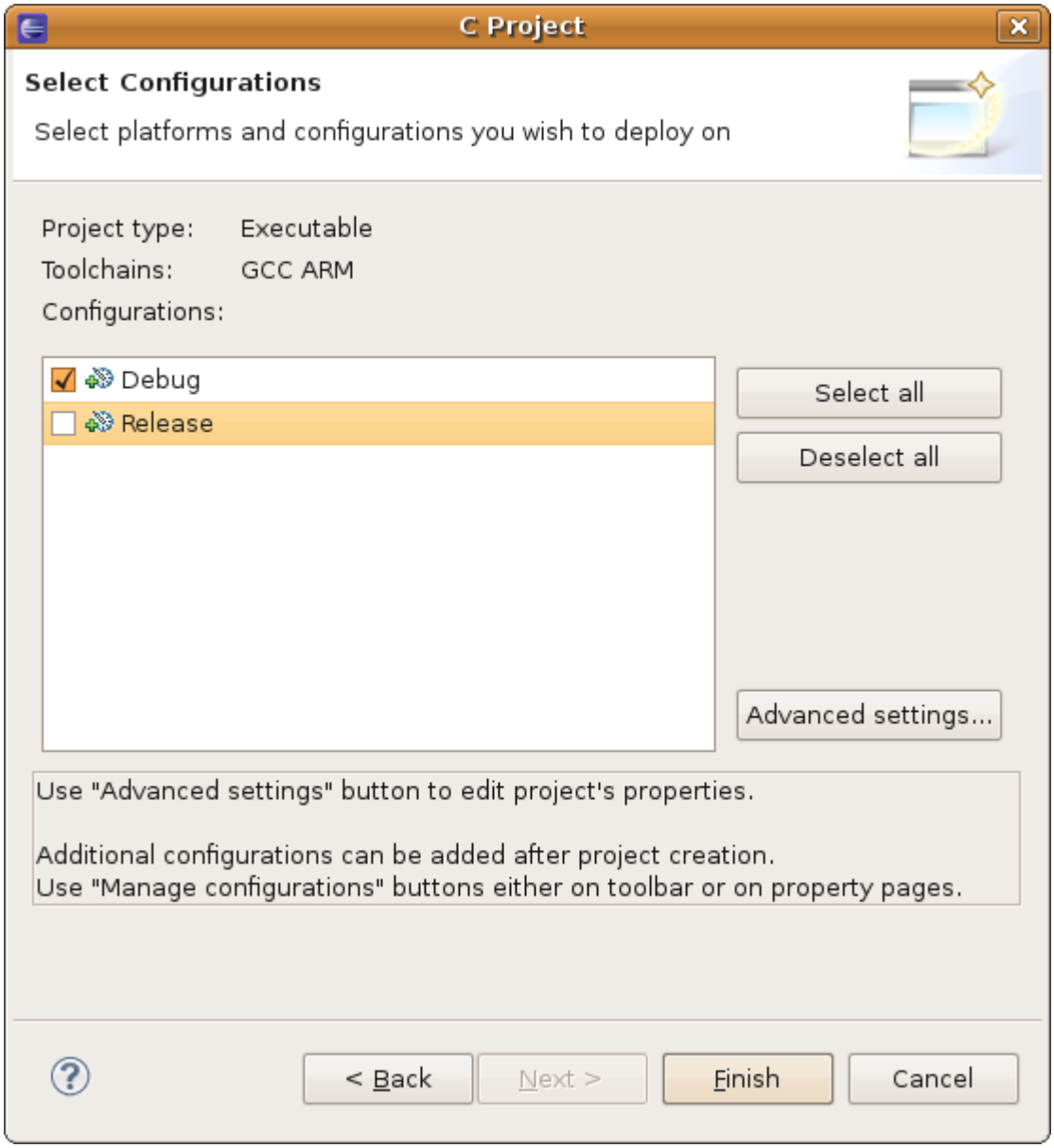
To test our installation we will create an empty project and copy the files from a project that we know works fine.



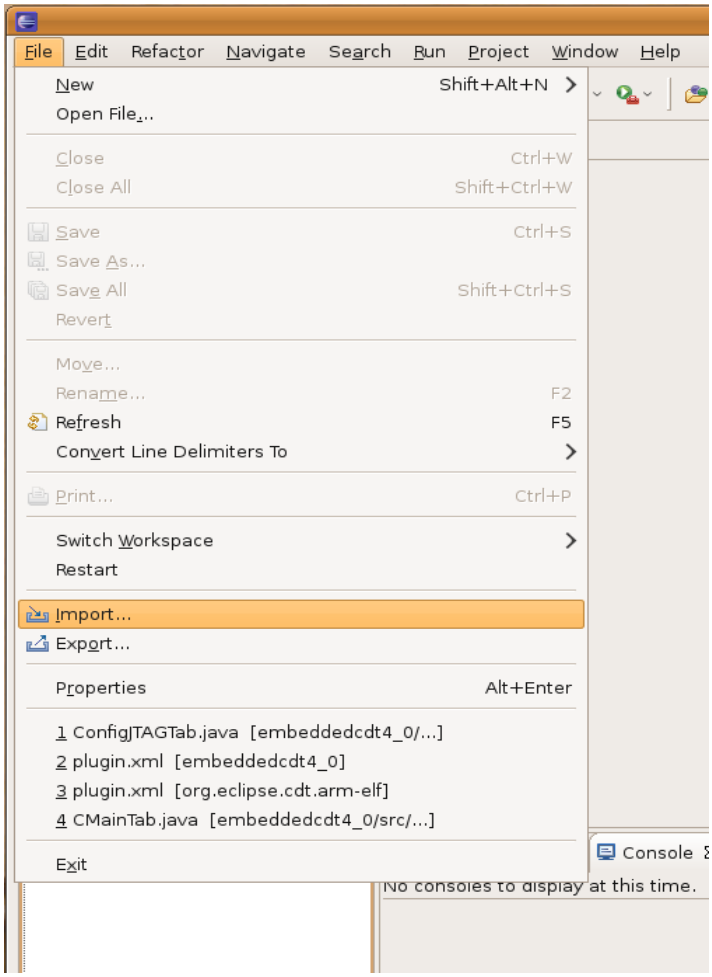
Enter a name for the project. We will use the default location for the project. It will be a GCC ARM project. The GCC ARM project uses the arm-elf-gcc compiler.



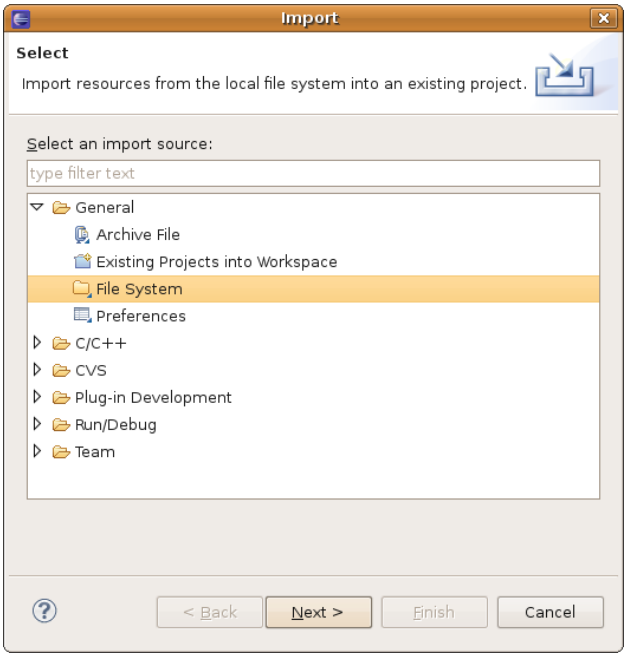
We will just create a debug environment for this test so we remove the mark for the Release section. Press on next.

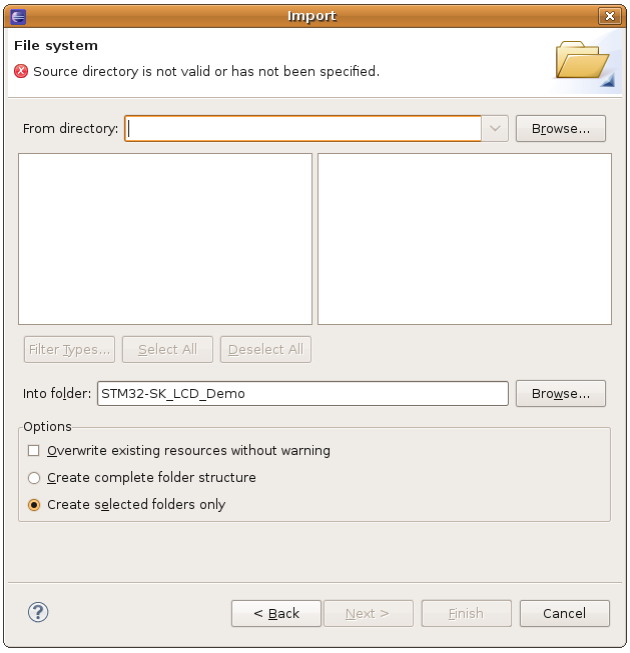


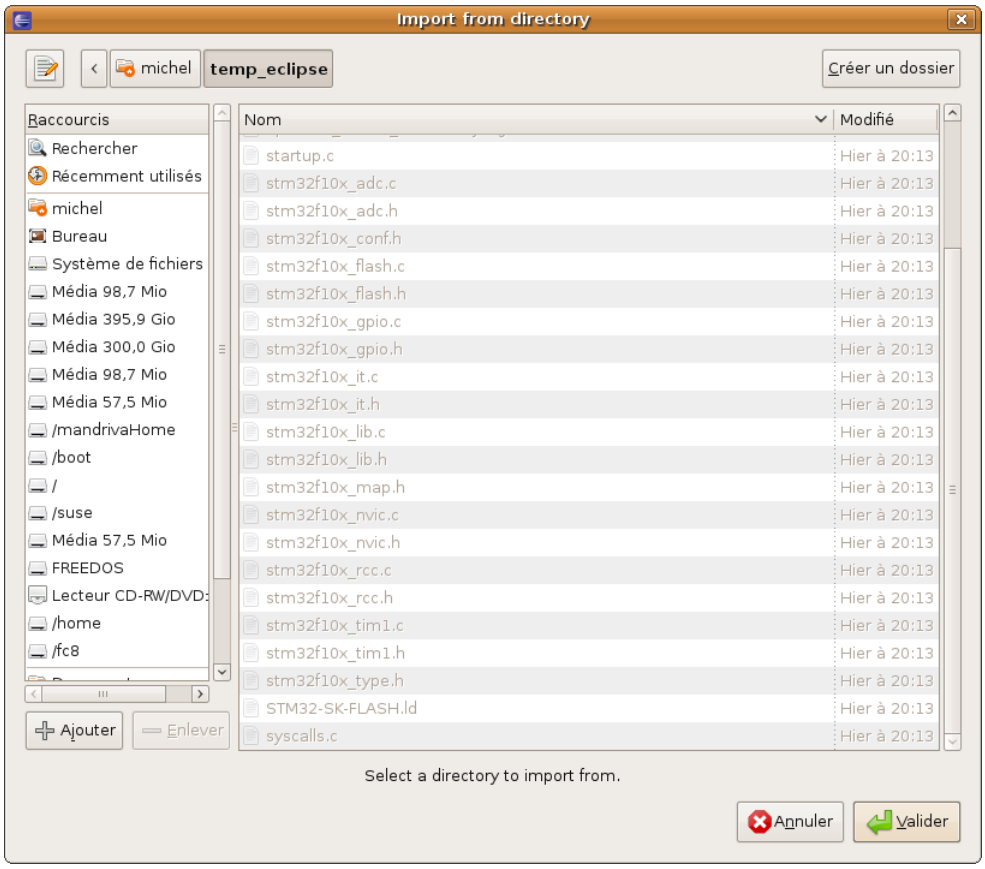
For now we just press on Finish. We will come back to the properties later on to fix the project configurations.



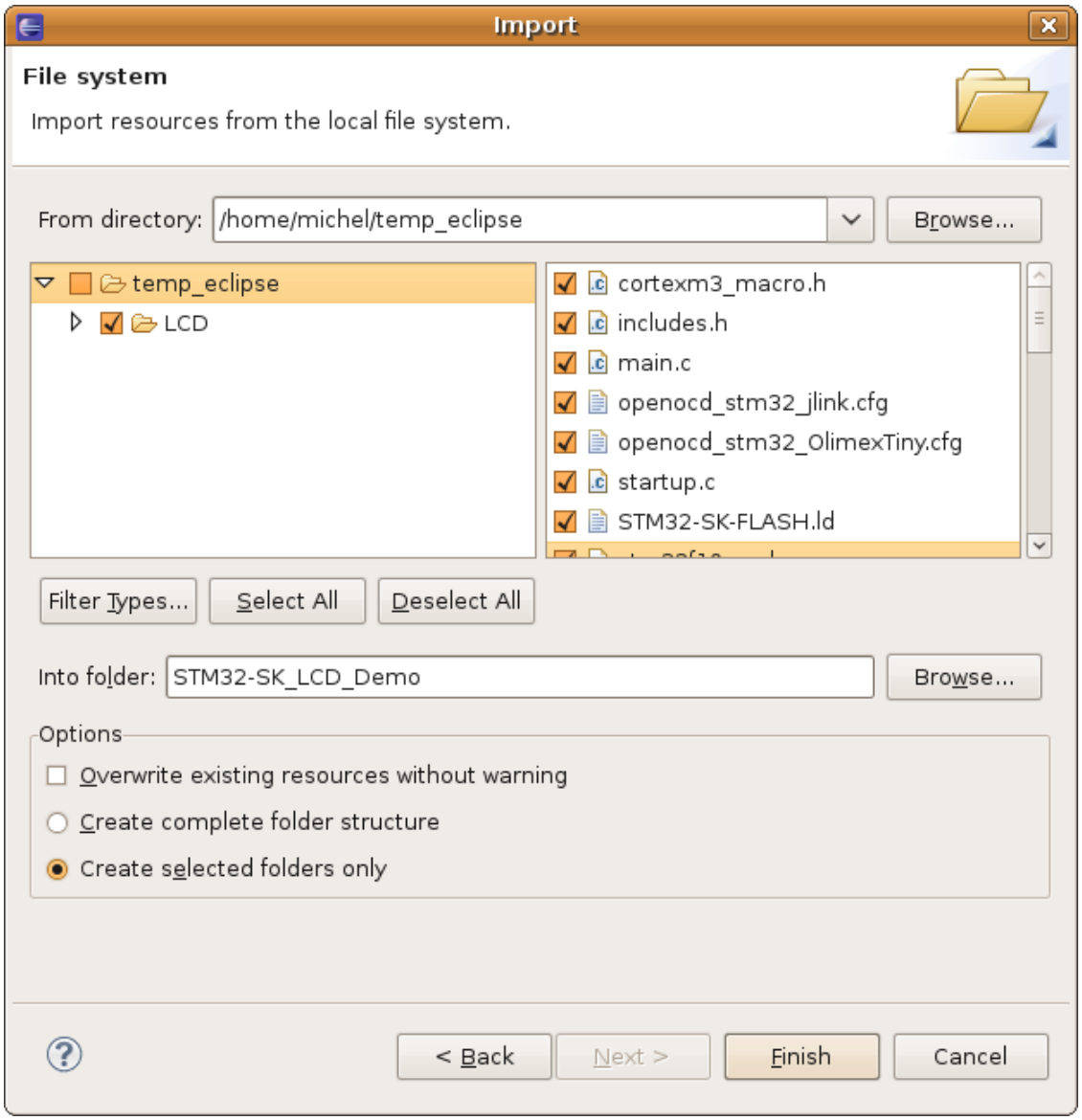
We will import some files from a working project. We don't bring any makefiles since the plugin will take care of creating the proper makefiles needed.


Import
Select
Import resources from the local file system into an existing project.
Select an import source:
type filter text
General
Archive File
Existing Projects into Workspace
File System
Preferences
C/C++
CVS
Plug-in Development
Run/Debug
Team
< Back
Next >
Finish
Cancel

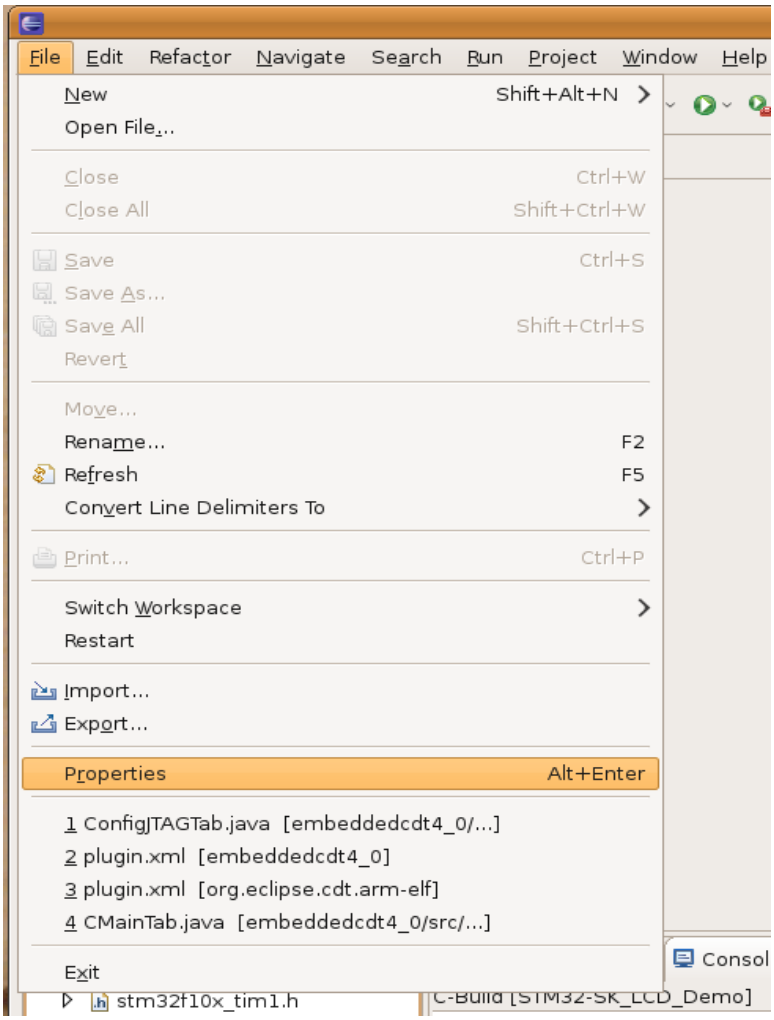

Import
File system
Source directory is not valid or has not been specified.
From directory:
Browse...
Filter Types...
Select All
Deselect All
Into folder:
STM32-SK_LCD_Demo
Browse...
Options
Overwrite existing resources without warning
Create complete folder structure
Create selected folders only
< Back
Next >
Finish
Cancel


Import from directory
michel
temp_eclipse
Créer un dossier
Raccourcis
Rechercher
Récemment utilisés
michel
Bureau
Système de fichiers
Média 98,7 Mio
Média 395,9 Gio
Média 300,0 Gio
Média 98,7 Mio
Média 57,5 Mio
/mandrivaHome
/boot
/
/suse
Média 57,5 Mio
FREEDOS
Lecteur CD-RW/DVD
/home
/fc8
Ajouter
Enlever
Nom
Modifié
startup.c
stm32f10x_adc.c
stm32f10x_adc.h
stm32f10x_conf.h
stm32f10x_flash.c
stm32f10x_flash.h
stm32f10x_gpio.c
stm32f10x_gpio.h
stm32f10x_it.c
stm32f10x_it.h
stm32f10x_lib.c
stm32f10x_lib.h
stm32f10x_map.h
stm32f10x_nvic.c
stm32f10x_nvic.h
stm32f10x_rcc.c
stm32f10x_rcc.h
stm32f10x_tim1.c
stm32f10x_tim1.h
stm32f10x_type.h
STM32-SK-FLASH.ld
syscalls.c
Hier à 20:13
Select a directory to import from.
Annuler
Valider

We pick all the files on the root directory of ~/temp_eclipse/STM32-SK_LCD_Demo as well as the LCD directory. All the files will be copied to the ~/workspace/STM32-SK_LCD_Demo directory. Do not mark the STM32-SK_LCD_Demo directory.



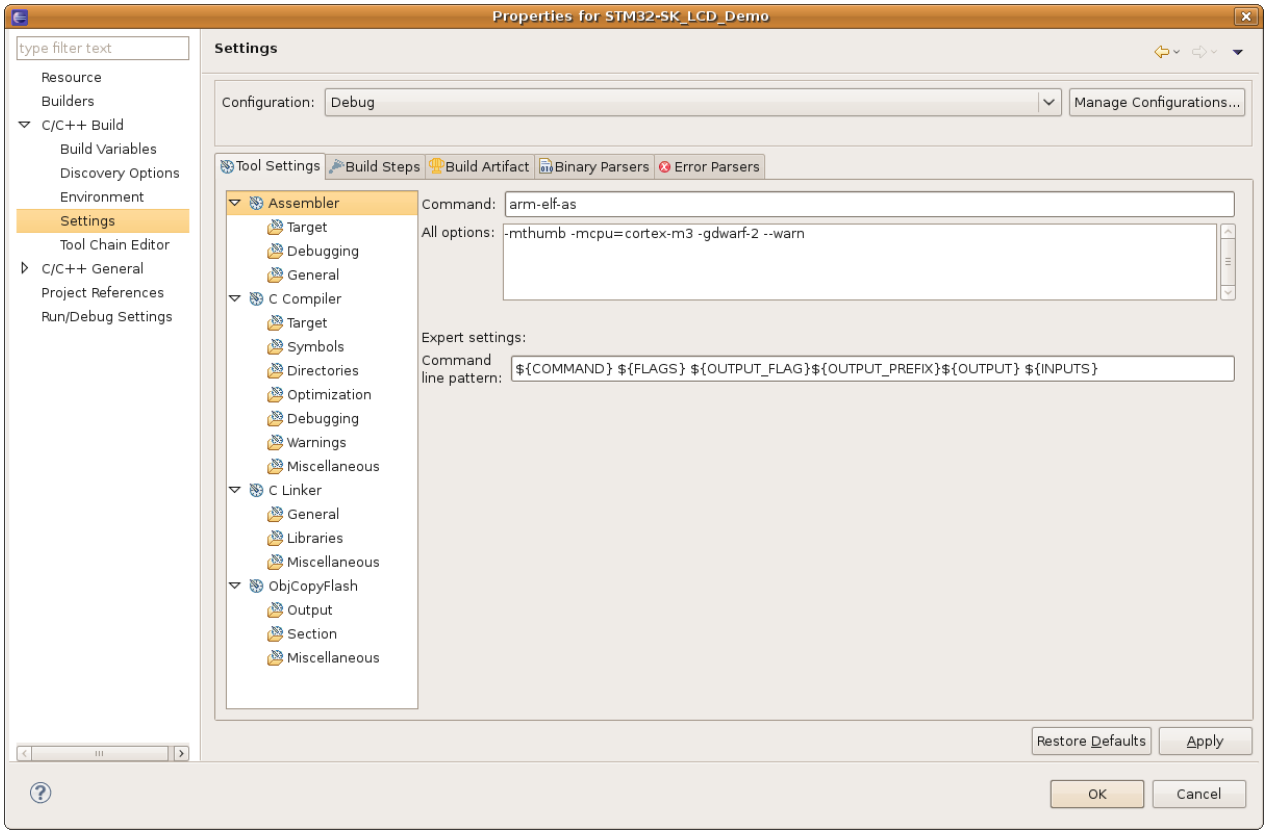
Press finish when all the files have been selected



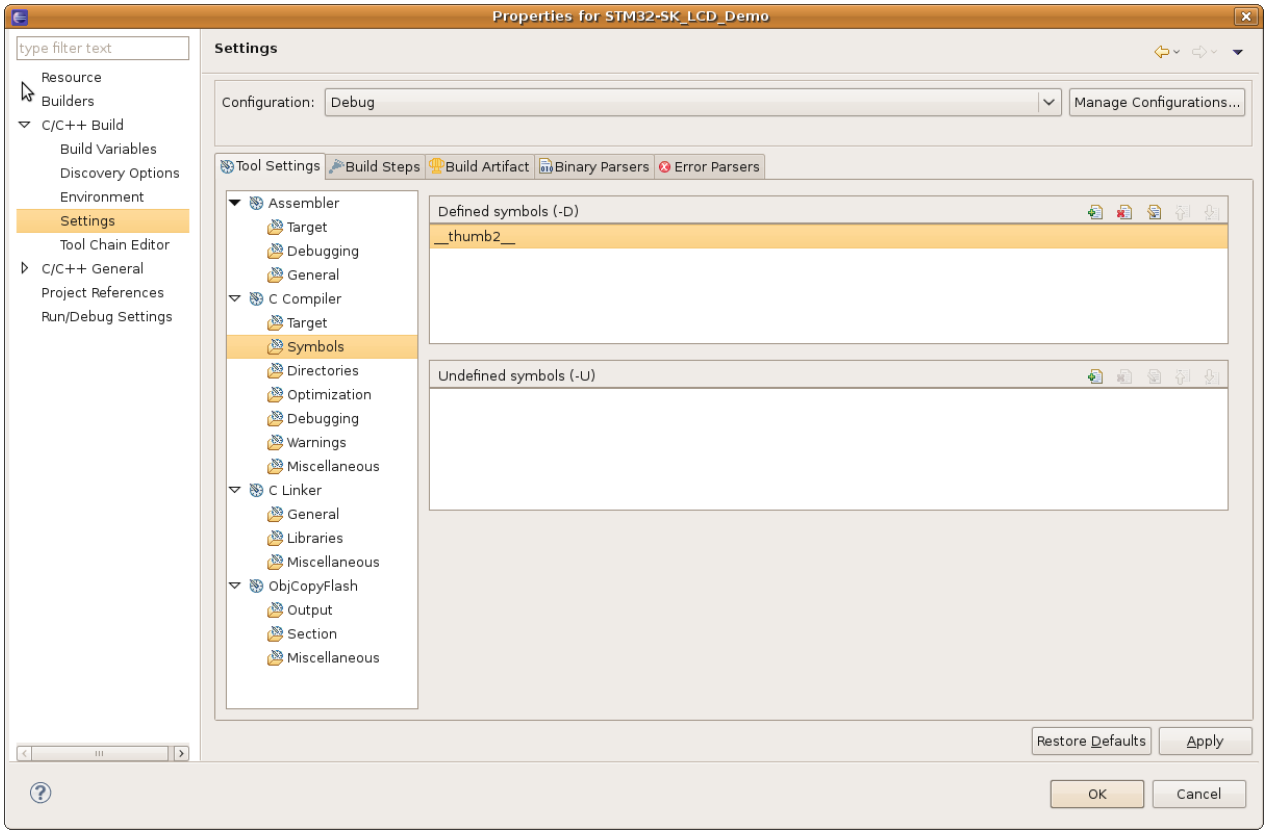
The project is almost ready. Before we try to compile though we need to adjust the properties.

Right click on STM32-SK_LCD_Demo.
On the bottom click on Properties.

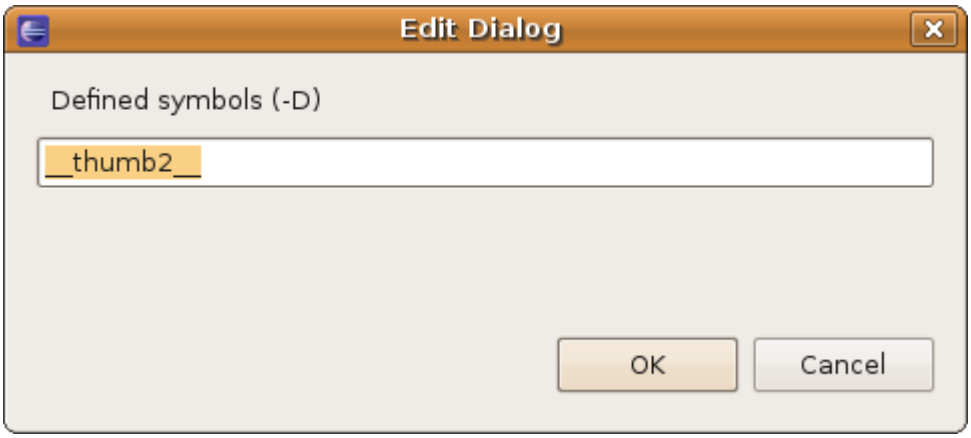
You may need to expand C/C++ Build settings if not already done. Pick the Settings option.



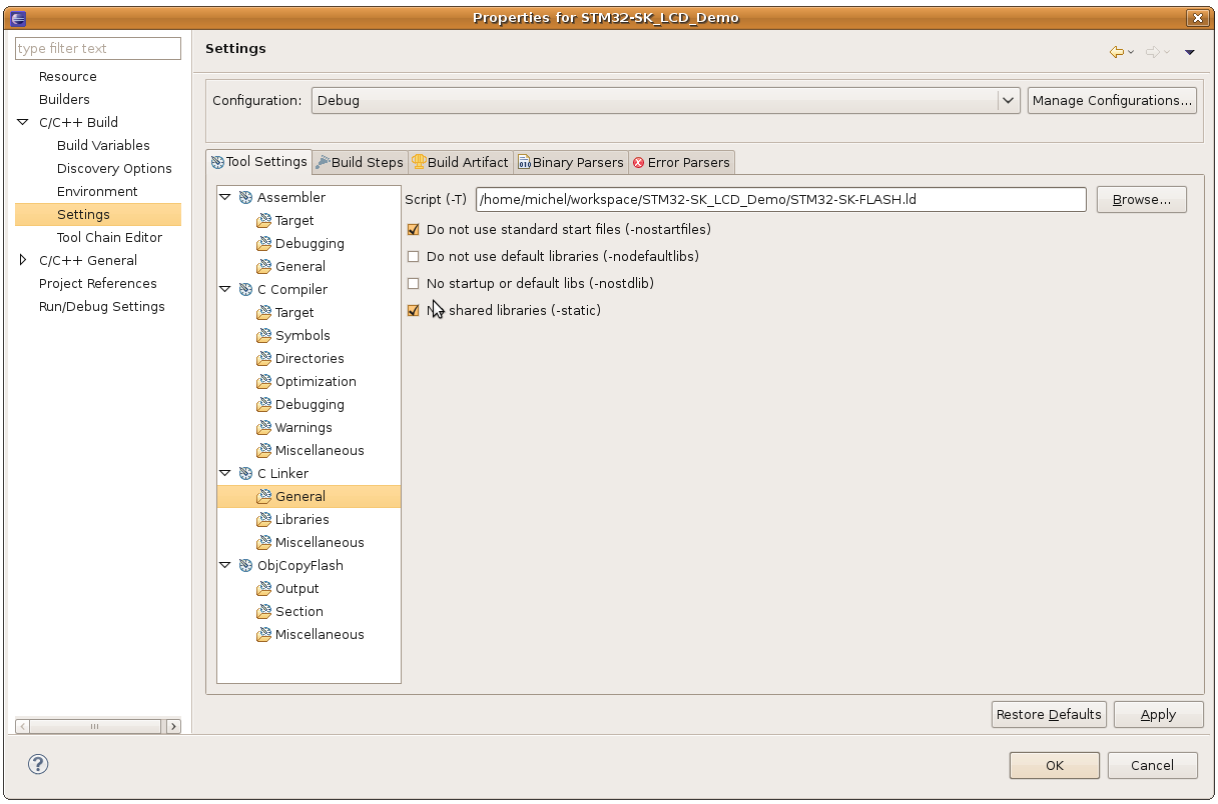
We first start with the Symbols. We need to add a definition. Click on the add button.

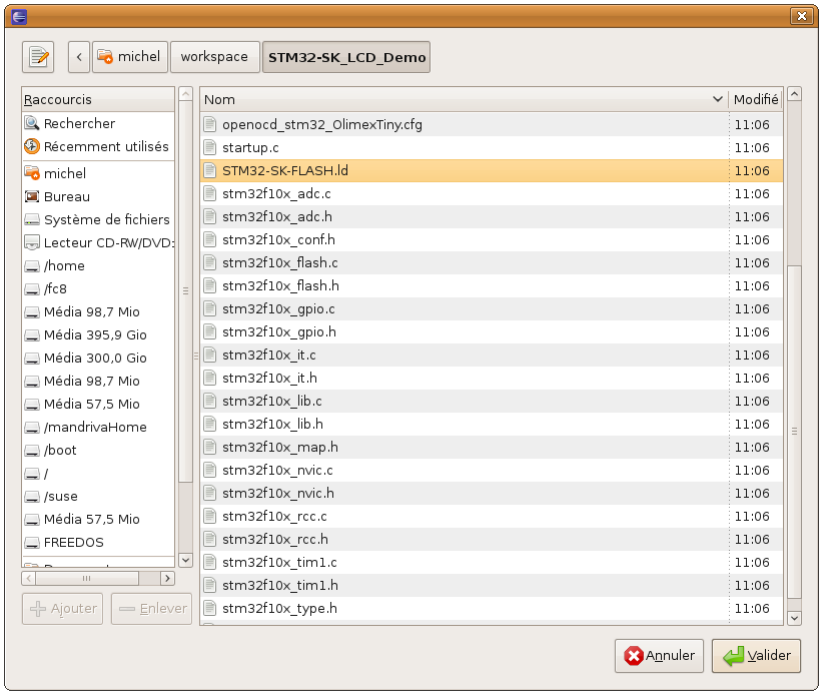


To enter the definition on new and type __thumb2__ and press on OK.

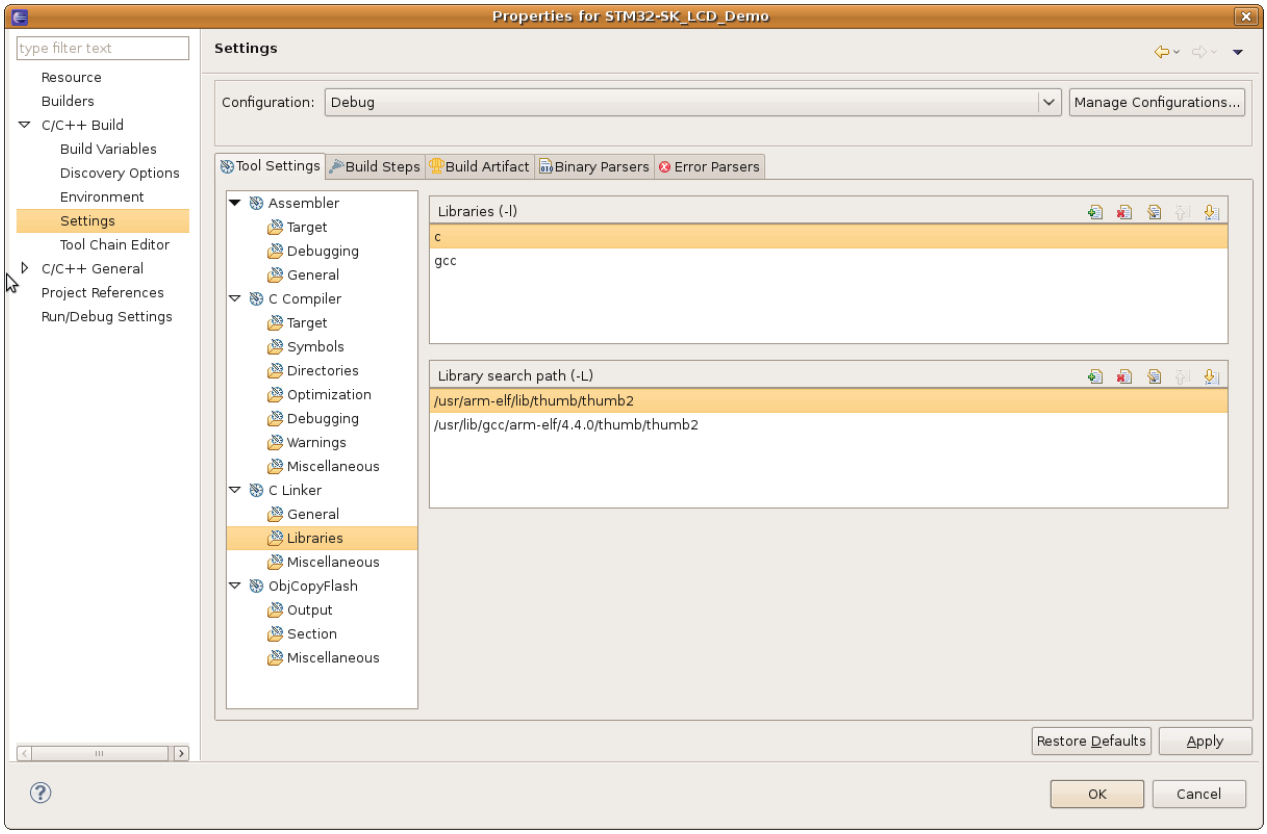


We need to identify the linker script. Press on Browse and pick the file STM32-SK-FLASH.ld on the project directory.

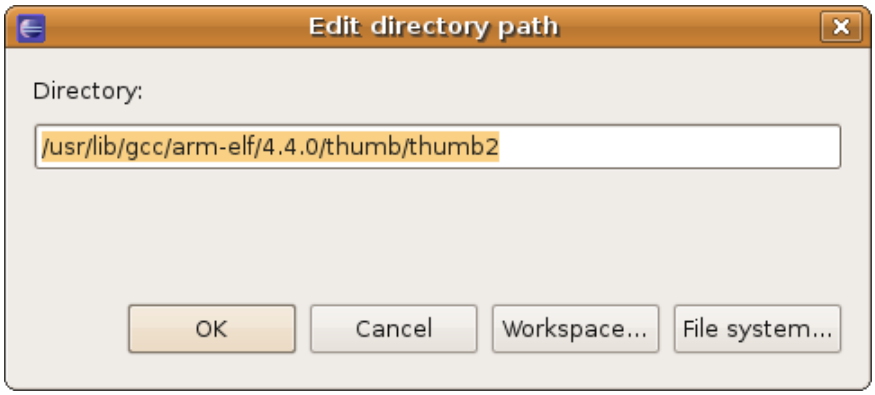




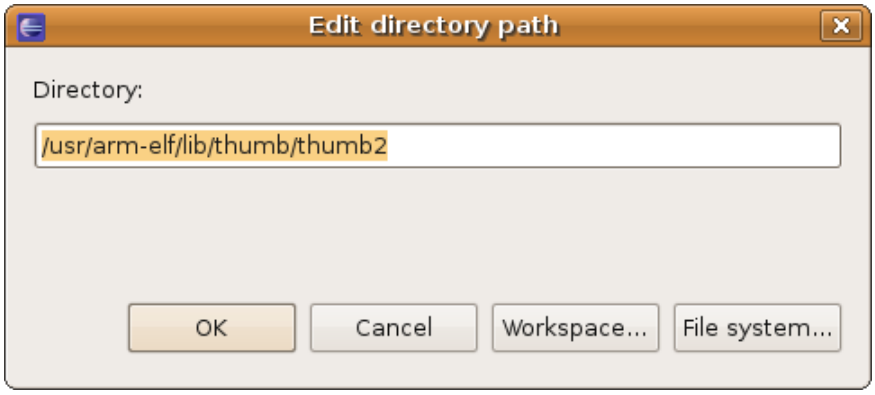
The next step will be to set the library paths and libraries used. Click on add in the libraries section and enter c and gcc.



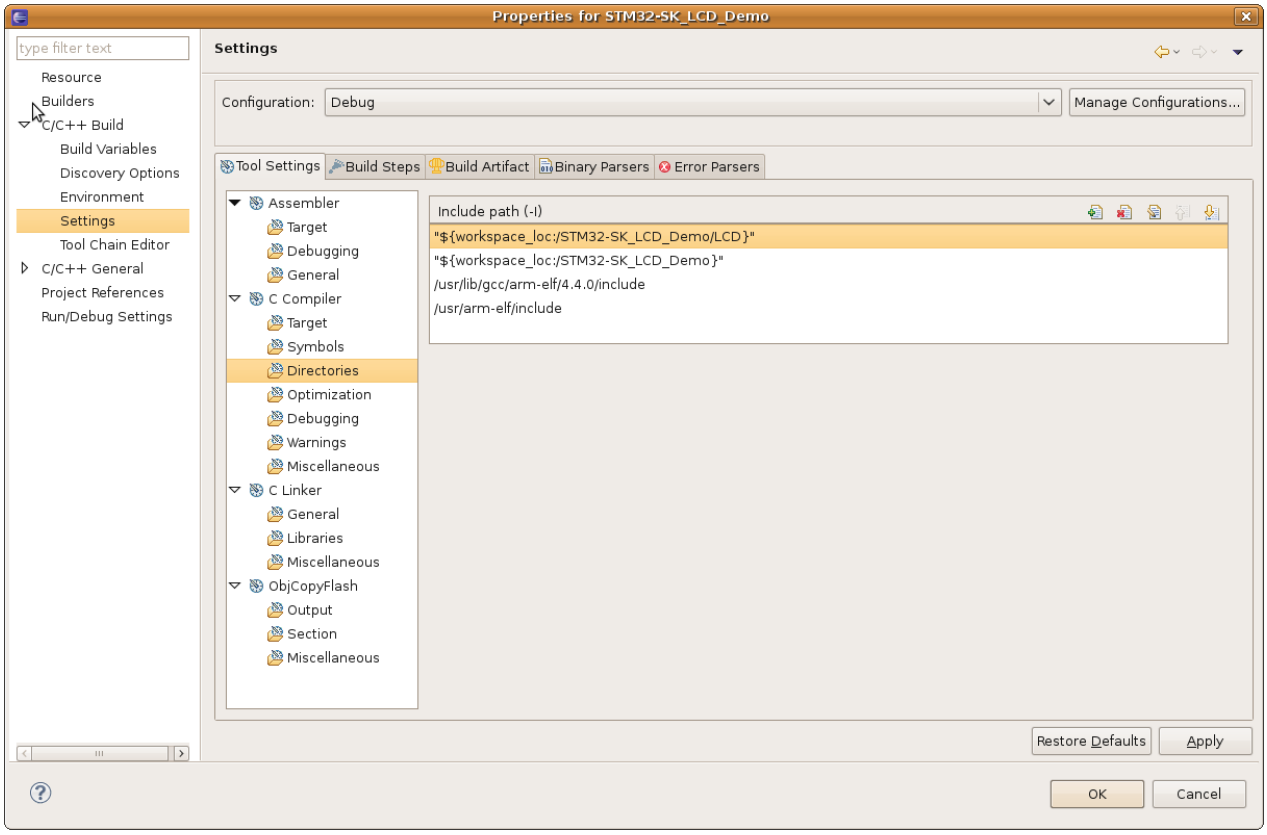
Click on add in the Library search path section. Enter the path for the libgcc library for thumb2 and click on OK.



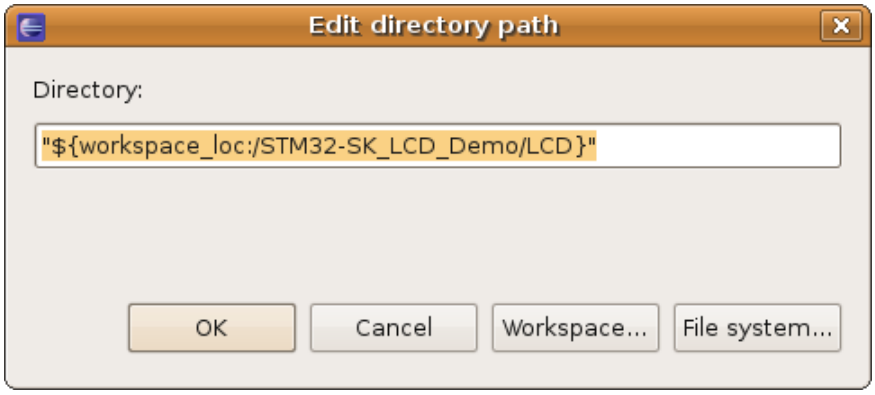
Click again on add and enter the patch for the library for libc for thumb2 and click on OK.



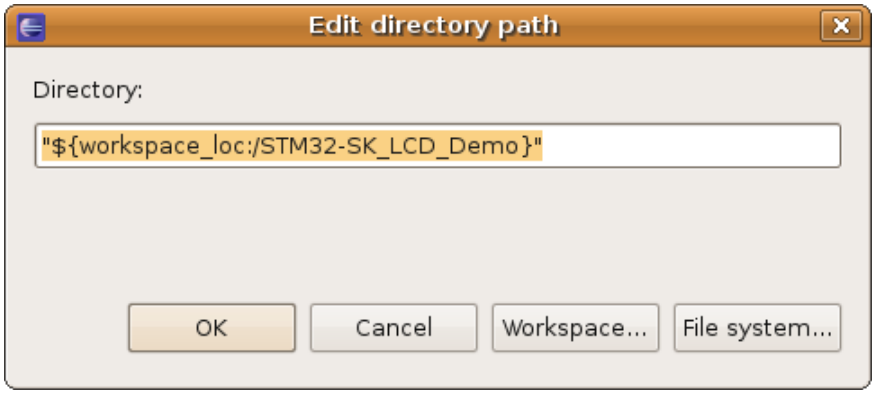
Next we need to setup the include paths



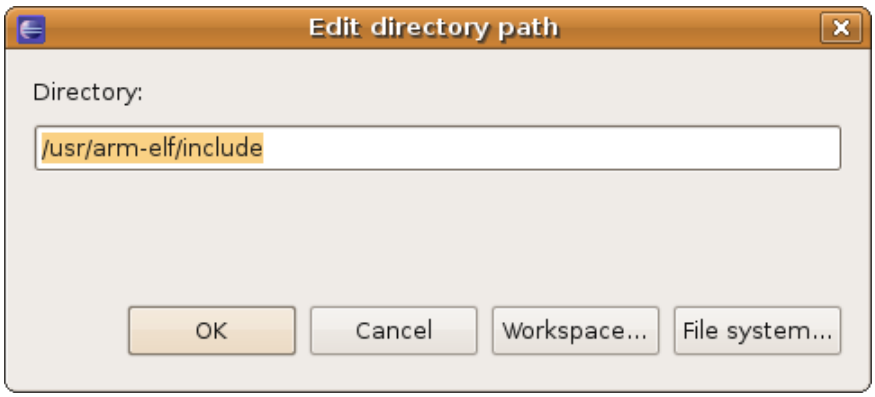
Click on the add button and enter the local path the LCD directory.



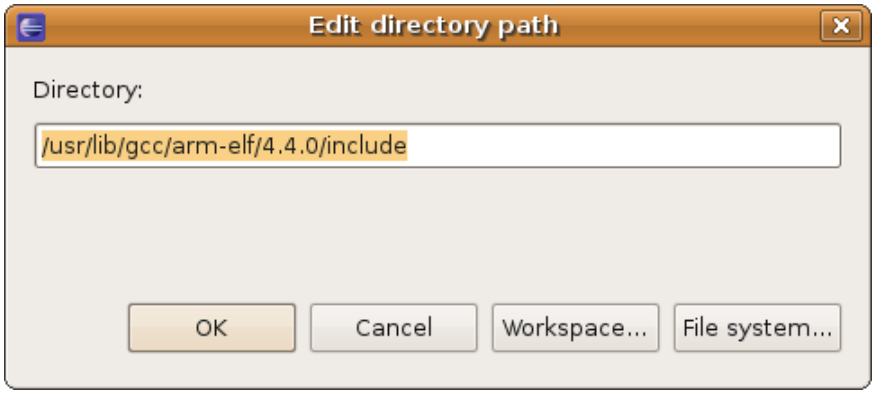
Click on the add button and enter the local path the project directory.



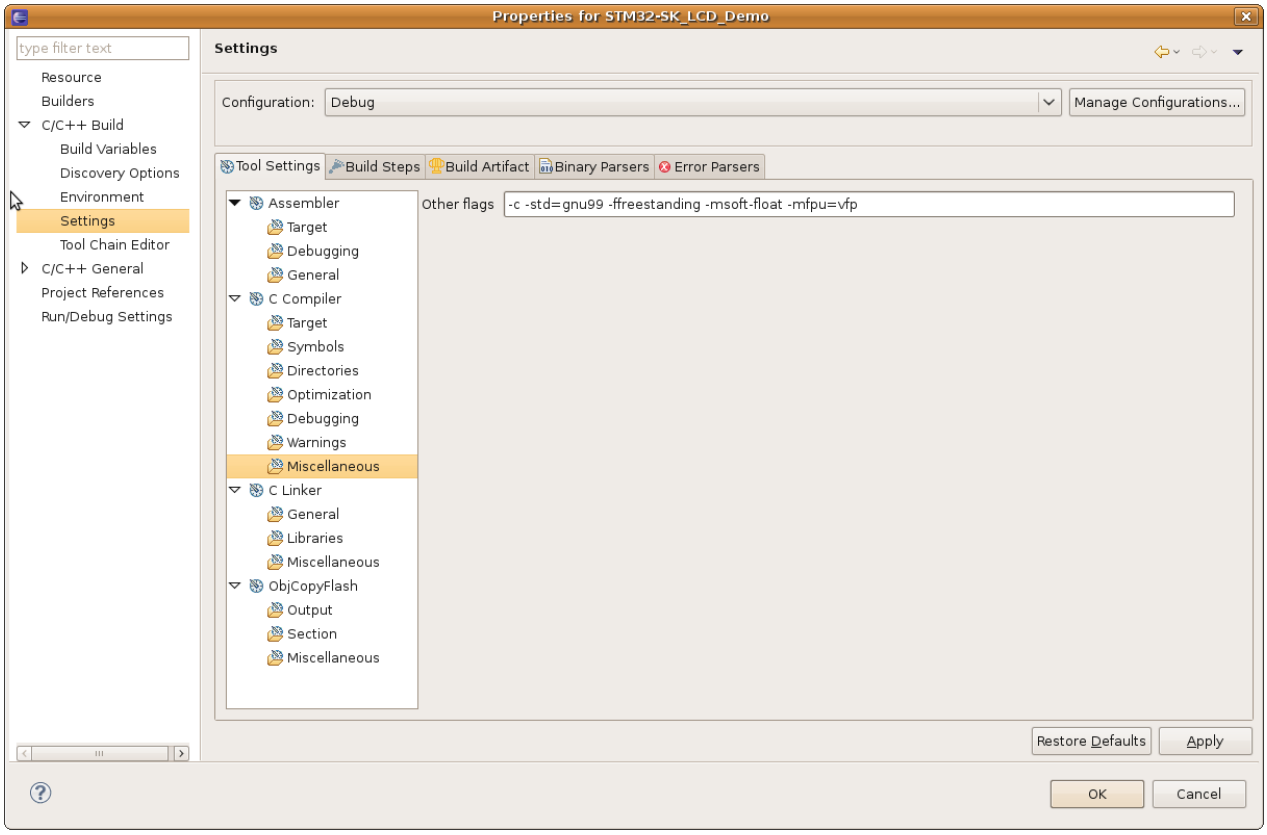
Click on the add button and enter the include path for libc.



Click on the add button and enter the include path for libgcc.



We need to add a few options. The flags -std=gnu99 is added to support some new features. The flag –-freestanding tells gcc that it is an embedded application where main doesn't require a return value. The flags -msoft-float and -mfpu=vfp are needed to make sure that the proper style of floating point is picked. It won't link if there is a mismatch between the libraries and the application code. When newlib or libgcc are not used and there are no floating point in the application the flags are not needed. If you use printf there will be a problem without those flags. When you use printf the floating point library is linked in as requested by newlib.

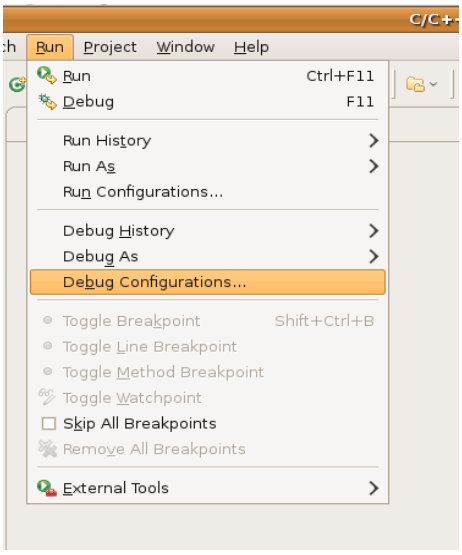


Press on apply to finish the properties settings. Now it is time to do a build. Click on the pulldown menu Project and then Rebuild Project. It should compile without error. Double check on the console at the bottom. You should see the following if everything compiled correctly.

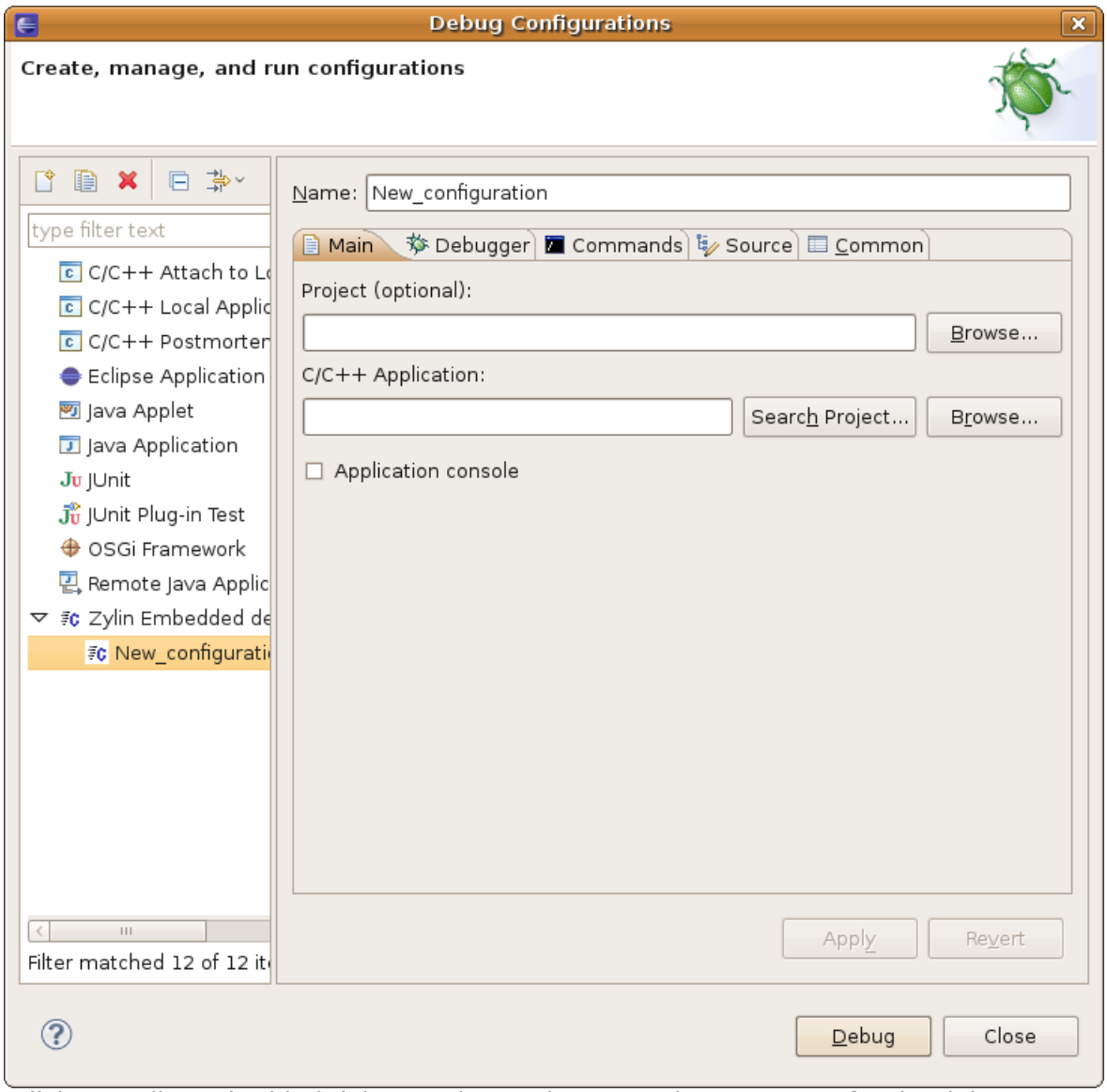
```
**** Build of configuration Debug for project STM32-SK_LCD_Demo ****
make -k all
Building file: ../main.c
Invoking: C Compiler
arm-elf-gcc -mthumb -mcpu=cortex-m3 -D__thumb2__ -I"/home/michel/workspace/STM32-SK_LCD_Demo/LCD" -I"/home/
michel/workspace/STM32-SK_LCD_Demo" -I/usr/lib/gcc/arm-elf/4.4.0/include -I/usr/arm-elf/include -O0 -g2 -Wall -c
-std=gnu99 -ffreestanding -msoft-float -mfpu=vfp -o"main.o" "../main.c" && \
        echo -n 'main.d' ./ > 'main.d' && \
        arm-elf-gcc -MM -MG -P -w -mthumb -mcpu=cortex-m3 -D__thumb2__ -I"/home/michel/workspace/STM32-
SK_LCD_Demo/LCD" -I"/home/michel/workspace/STM32-SK_LCD_Demo" -I/usr/lib/gcc/arm-elf/4.4.0/include -I/usr/arm-
elf/include -O0 -g2 -Wall -c -std=gnu99 -ffreestanding -msoft-float -mfpu=vfp  "../main.c" >> 'main.d'
Finished building: ../main.c

Building file: ../startup.c
Invoking: C Compiler
arm-elf-gcc -mthumb -mcpu=cortex-m3 -D__thumb2__ -I"/home/michel/workspace/STM32-SK_LCD_Demo/LCD" -I"/home/
michel/workspace/STM32-SK_LCD_Demo" -I/usr/lib/gcc/arm-elf/4.4.0/include -I/usr/arm-elf/include -O0 -g2 -Wall -c
-std=gnu99 -ffreestanding -msoft-float -mfpu=vfp -o"startup.o" "../startup.c" && \
        echo -n 'startup.d' ./ > 'startup.d' && \
        arm-elf-gcc -MM -MG -P -w -mthumb -mcpu=cortex-m3 -D__thumb2__ -I"/home/michel/workspace/STM32-
SK_LCD_Demo/LCD" -I"/home/michel/workspace/STM32-SK_LCD_Demo" -I/usr/lib/gcc/arm-elf/4.4.0/include -I/usr/arm-
elf/include -O0 -g2 -Wall -c -std=gnu99 -ffreestanding -msoft-float -mfpu=vfp  "../startup.c" >> 'startup.d'
Finished building: ../startup.c

Building file: ../stm32f10x_adc.c
Invoking: C Compiler
arm-elf-gcc -mthumb -mcpu=cortex-m3 -D__thumb2__ -I"/home/michel/workspace/STM32-SK_LCD_Demo/LCD" -I"/home/
michel/workspace/STM32-SK_LCD_Demo" -I/usr/lib/gcc/arm-elf/4.4.0/include -I/usr/arm-elf/include -O0 -g2 -Wall -c
-std=gnu99 -ffreestanding -msoft-float -mfpu=vfp -o"stm32f10x_adc.o" "../stm32f10x_adc.c" && \
        echo -n 'stm32f10x_adc.d' ./ > 'stm32f10x_adc.d' && \
        arm-elf-gcc -MM -MG -P -w -mthumb -mcpu=cortex-m3 -D__thumb2__ -I"/home/michel/workspace/STM32-
SK_LCD_Demo/LCD" -I"/home/michel/workspace/STM32-SK_LCD_Demo" -I/usr/lib/gcc/arm-elf/4.4.0/include -I/usr/arm-
elf/include -O0 -g2 -Wall -c -std=gnu99 -ffreestanding -msoft-float -mfpu=vfp  "../stm32f10x_adc.c" >> 'stm32f10x_adc.d'
Finished building: ../stm32f10x_adc.c

Building file: ../stm32f10x_flash.c
Invoking: C Compiler
arm-elf-gcc -mthumb -mcpu=cortex-m3 -D__thumb2__ -I"/home/michel/workspace/STM32-SK_LCD_Demo/LCD" -I"/home/
michel/workspace/STM32-SK_LCD_Demo" -I/usr/lib/gcc/arm-elf/4.4.0/include -I/usr/arm-elf/include -O0 -g2 -Wall -c
-std=gnu99 -ffreestanding -msoft-float -mfpu=vfp -o"stm32f10x_flash.o" "../stm32f10x_flash.c" && \
        echo -n 'stm32f10x_flash.d' ./ > 'stm32f10x_flash.d' && \
        arm-elf-gcc -MM -MG -P -w -mthumb -mcpu=cortex-m3 -D__thumb2__ -I"/home/michel/workspace/STM32-
SK_LCD_Demo/LCD" -I"/home/michel/workspace/STM32-SK_LCD_Demo" -I/usr/lib/gcc/arm-elf/4.4.0/include -I/usr/arm-
elf/include -O0 -g2 -Wall -c -std=gnu99 -ffreestanding -msoft-float -mfpu=vfp  "../stm32f10x_flash.c" >>
'stm32f10x_flash.d'
Finished building: ../stm32f10x_flash.c

Building file: ../stm32f10x_gpio.c
Invoking: C Compiler
arm-elf-gcc -mthumb -mcpu=cortex-m3 -D__thumb2__ -I"/home/michel/workspace/STM32-SK_LCD_Demo/LCD" -I"/home/
michel/workspace/STM32-SK_LCD_Demo" -I/usr/lib/gcc/arm-elf/4.4.0/include -I/usr/arm-elf/include -O0 -g2 -Wall -c
-std=gnu99 -ffreestanding -msoft-float -mfpu=vfp -o"stm32f10x_gpio.o" "../stm32f10x_gpio.c" && \
        echo -n 'stm32f10x_gpio.d' ./ > 'stm32f10x_gpio.d' && \
        arm-elf-gcc -MM -MG -P -w -mthumb -mcpu=cortex-m3 -D__thumb2__ -I"/home/michel/workspace/STM32-
SK_LCD_Demo/LCD" -I"/home/michel/workspace/STM32-SK_LCD_Demo" -I/usr/lib/gcc/arm-elf/4.4.0/include -I/usr/arm-
elf/include -O0 -g2 -Wall -c -std=gnu99 -ffreestanding -msoft-float -mfpu=vfp  "../stm32f10x_gpio.c" >>
'stm32f10x_gpio.d'
Finished building: ../stm32f10x_gpio.c
Building file: ../stm32f10x_it.c
Invoking: C Compiler
```

```
arm-elf-gcc -mthumb -mcpu=cortex-m3 -D__thumb2__ -I"/home/michel/workspace/STM32-SK_LCD_Demo/LCD" -I"/home/
michel/workspace/STM32-SK_LCD_Demo" -I/usr/lib/gcc/arm-elf/4.4.0/include -I/usr/arm-elf/include -O0 -g2 -Wall -c
-std=gnu99 -ffreestanding -msoft-float -mfpu=vfp -o"stm32f10x_it.o" "../stm32f10x_it.c" && \
        echo -n 'stm32f10x_it.d' ./ > 'stm32f10x_it.d' && \
        arm-elf-gcc -MM -MG -P -w -mthumb -mcpu=cortex-m3 -D__thumb2__ -I"/home/michel/workspace/STM32-
SK_LCD_Demo/LCD" -I"/home/michel/workspace/STM32-SK_LCD_Demo" -I/usr/lib/gcc/arm-elf/4.4.0/include -I/usr/arm-
elf/include -O0 -g2 -Wall -c -std=gnu99 -ffreestanding -msoft-float -mfpu=vfp  "../stm32f10x_it.c" >> 'stm32f10x_it.d'
Finished building: ../stm32f10x_it.c

Building file: ../stm32f10x_lib.c
Invoking: C Compiler
arm-elf-gcc -mthumb -mcpu=cortex-m3 -D__thumb2__ -I"/home/michel/workspace/STM32-SK_LCD_Demo/LCD" -I"/home/
michel/workspace/STM32-SK_LCD_Demo" -I/usr/lib/gcc/arm-elf/4.4.0/include -I/usr/arm-elf/include -O0 -g2 -Wall -c
-std=gnu99 -ffreestanding -msoft-float -mfpu=vfp -o"stm32f10x_lib.o" "../stm32f10x_lib.c" && \
        echo -n 'stm32f10x_lib.d' ./ > 'stm32f10x_lib.d' && \
        arm-elf-gcc -MM -MG -P -w -mthumb -mcpu=cortex-m3 -D__thumb2__ -I"/home/michel/workspace/STM32-
SK_LCD_Demo/LCD" -I"/home/michel/workspace/STM32-SK_LCD_Demo" -I/usr/lib/gcc/arm-elf/4.4.0/include -I/usr/arm-
elf/include -O0 -g2 -Wall -c -std=gnu99 -ffreestanding -msoft-float -mfpu=vfp  "../stm32f10x_lib.c" >> 'stm32f10x_lib.d'
Finished building: ../stm32f10x_lib.c

Building file: ../stm32f10x_nvic.c
Invoking: C Compiler
arm-elf-gcc -mthumb -mcpu=cortex-m3 -D__thumb2__ -I"/home/michel/workspace/STM32-SK_LCD_Demo/LCD" -I"/home/
michel/workspace/STM32-SK_LCD_Demo" -I/usr/lib/gcc/arm-elf/4.4.0/include -I/usr/arm-elf/include -O0 -g2 -Wall -c
-std=gnu99 -ffreestanding -msoft-float -mfpu=vfp -o"stm32f10x_nvic.o" "../stm32f10x_nvic.c" && \
        echo -n 'stm32f10x_nvic.d' ./ > 'stm32f10x_nvic.d' && \
        arm-elf-gcc -MM -MG -P -w -mthumb -mcpu=cortex-m3 -D__thumb2__ -I"/home/michel/workspace/STM32-
SK_LCD_Demo/LCD" -I"/home/michel/workspace/STM32-SK_LCD_Demo" -I/usr/lib/gcc/arm-elf/4.4.0/include -I/usr/arm-
elf/include -O0 -g2 -Wall -c -std=gnu99 -ffreestanding -msoft-float -mfpu=vfp  "../stm32f10x_nvic.c" >>
'stm32f10x_nvic.d'
Finished building: ../stm32f10x_nvic.c

Building file: ../stm32f10x_rcc.c
Invoking: C Compiler
arm-elf-gcc -mthumb -mcpu=cortex-m3 -D__thumb2__ -I"/home/michel/workspace/STM32-SK_LCD_Demo/LCD" -I"/home/
michel/workspace/STM32-SK_LCD_Demo" -I/usr/lib/gcc/arm-elf/4.4.0/include -I/usr/arm-elf/include -O0 -g2 -Wall -c
-std=gnu99 -ffreestanding -msoft-float -mfpu=vfp -o"stm32f10x_rcc.o" "../stm32f10x_rcc.c" && \
        echo -n 'stm32f10x_rcc.d' ./ > 'stm32f10x_rcc.d' && \
        arm-elf-gcc -MM -MG -P -w -mthumb -mcpu=cortex-m3 -D__thumb2__ -I"/home/michel/workspace/STM32-
SK_LCD_Demo/LCD" -I"/home/michel/workspace/STM32-SK_LCD_Demo" -I/usr/lib/gcc/arm-elf/4.4.0/include -I/usr/arm-
elf/include -O0 -g2 -Wall -c -std=gnu99 -ffreestanding -msoft-float -mfpu=vfp  "../stm32f10x_rcc.c" >> 'stm32f10x_rcc.d'
Finished building: ../stm32f10x_rcc.c

Building file: ../stm32f10x_tim1.c
Invoking: C Compiler
arm-elf-gcc -mthumb -mcpu=cortex-m3 -D__thumb2__ -I"/home/michel/workspace/STM32-SK_LCD_Demo/LCD" -I"/home/
michel/workspace/STM32-SK_LCD_Demo" -I/usr/lib/gcc/arm-elf/4.4.0/include -I/usr/arm-elf/include -O0 -g2 -Wall -c
-std=gnu99 -ffreestanding -msoft-float -mfpu=vfp -o"stm32f10x_tim1.o" "../stm32f10x_tim1.c" && \
        echo -n 'stm32f10x_tim1.d' ./ > 'stm32f10x_tim1.d' && \
        arm-elf-gcc -MM -MG -P -w -mthumb -mcpu=cortex-m3 -D__thumb2__ -I"/home/michel/workspace/STM32-
SK_LCD_Demo/LCD" -I"/home/michel/workspace/STM32-SK_LCD_Demo" -I/usr/lib/gcc/arm-elf/4.4.0/include -I/usr/arm-
elf/include -O0 -g2 -Wall -c -std=gnu99 -ffreestanding -msoft-float -mfpu=vfp  "../stm32f10x_tim1.c" >>
'stm32f10x_tim1.d'
Finished building: ../stm32f10x_tim1.c

Building file: ../syscalls.c
Invoking: C Compiler
arm-elf-gcc -mthumb -mcpu=cortex-m3 -D__thumb2__ -I"/home/michel/workspace/STM32-SK_LCD_Demo/LCD" -I"/home/
michel/workspace/STM32-SK_LCD_Demo" -I/usr/lib/gcc/arm-elf/4.4.0/include -I/usr/arm-elf/include -O0 -g2 -Wall -c
-std=gnu99 -ffreestanding -msoft-float -mfpu=vfp -o"syscalls.o" "../syscalls.c" && \
        echo -n 'syscalls.d' ./ > 'syscalls.d' && \
        arm-elf-gcc -MM -MG -P -w -mthumb -mcpu=cortex-m3 -D__thumb2__ -I"/home/michel/workspace/STM32-
SK_LCD_Demo/LCD" -I"/home/michel/workspace/STM32-SK_LCD_Demo" -I/usr/lib/gcc/arm-elf/4.4.0/include -I/usr/arm-
elf/include -O0 -g2 -Wall -c -std=gnu99 -ffreestanding -msoft-float -mfpu=vfp  "../syscalls.c" >> 'syscalls.d'
Finished building: ../syscalls.c
```

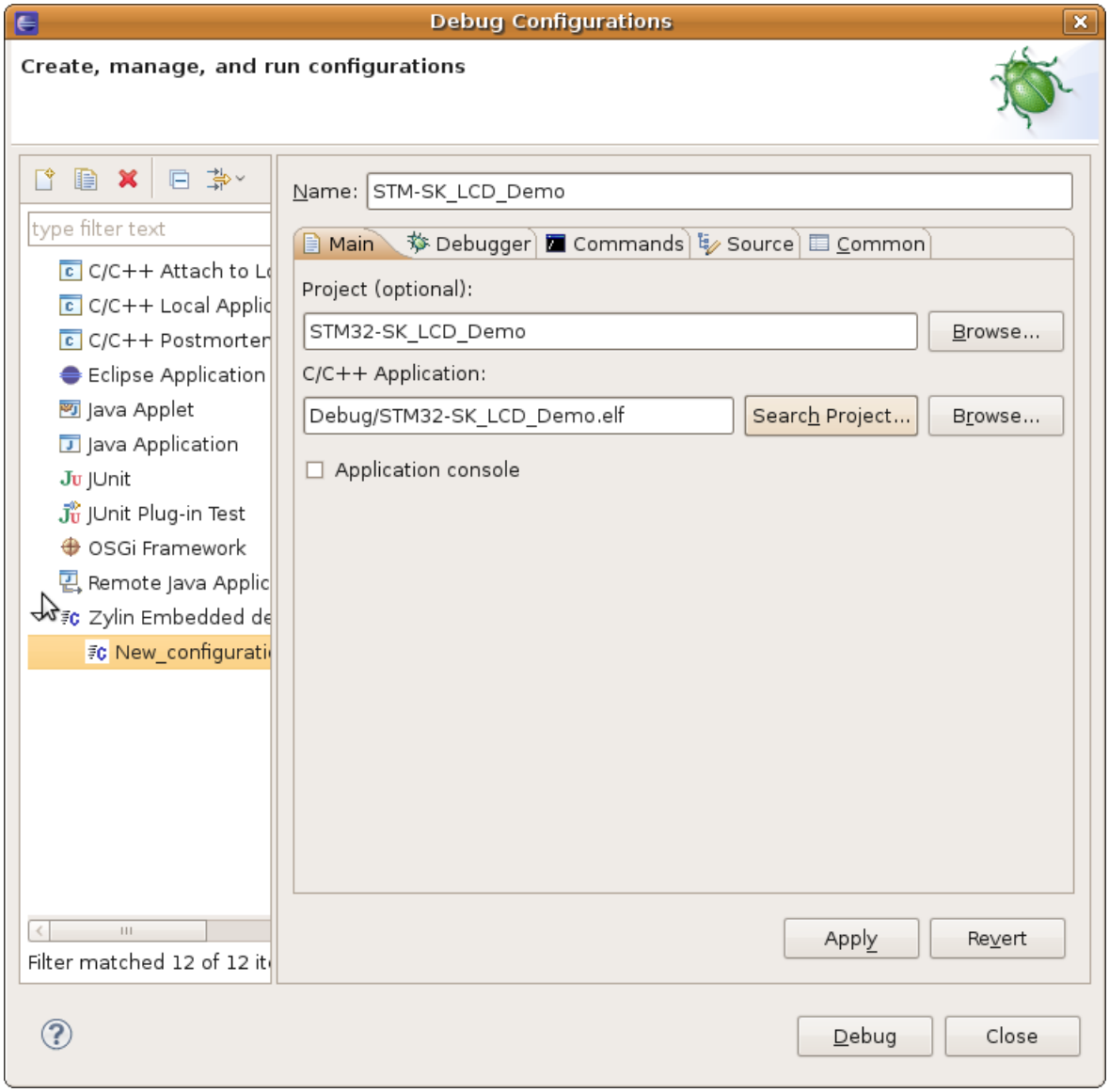
```
Building file: ../LCD/drv_hd44780.c
Invoking: C Compiler
arm-elf-gcc -mthumb -mcpu=cortex-m3 -D__thumb2__ -I"/home/michel/workspace/STM32-SK_LCD_Demo/LCD" -I"/home/
michel/workspace/STM32-SK_LCD_Demo" -I/usr/lib/gcc/arm-elf/4.4.0/include -I/usr/arm-elf/include -O0 -g2 -Wall -c
-std=gnu99 -ffreestanding -msoft-float -mfpu=vfp -o"LCD/drv_hd44780.o" "../LCD/drv_hd44780.c" && \
        echo -n 'LCD/drv_hd44780.d' LCD/ > 'LCD/drv_hd44780.d' && \
        arm-elf-gcc -MM -MG -P -w -mthumb -mcpu=cortex-m3 -D__thumb2__ -I"/home/michel/workspace/STM32-
SK_LCD_Demo/LCD" -I"/home/michel/workspace/STM32-SK_LCD_Demo" -I/usr/lib/gcc/arm-elf/4.4.0/include -I/usr/arm-
elf/include -O0 -g2 -Wall -c -std=gnu99 -ffreestanding -msoft-float -mfpu=vfp  "../LCD/drv_hd44780.c" >>
'LCD/drv_hd44780.d'
Finished building: ../LCD/drv_hd44780.c

Building file: ../LCD/drv_hd44780_l.c
Invoking: C Compiler
arm-elf-gcc -mthumb -mcpu=cortex-m3 -D__thumb2__ -I"/home/michel/workspace/STM32-SK_LCD_Demo/LCD" -I"/home/
michel/workspace/STM32-SK_LCD_Demo" -I/usr/lib/gcc/arm-elf/4.4.0/include -I/usr/arm-elf/include -O0 -g2 -Wall -c
-std=gnu99 -ffreestanding -msoft-float -mfpu=vfp -o"LCD/drv_hd44780_l.o" "../LCD/drv_hd44780_l.c" && \
        echo -n 'LCD/drv_hd44780_l.d' LCD/ > 'LCD/drv_hd44780_l.d' && \
        arm-elf-gcc -MM -MG -P -w -mthumb -mcpu=cortex-m3 -D__thumb2__ -I"/home/michel/workspace/STM32-
SK_LCD_Demo/LCD" -I"/home/michel/workspace/STM32-SK_LCD_Demo" -I/usr/lib/gcc/arm-elf/4.4.0/include -I/usr/arm-
elf/include -O0 -g2 -Wall -c -std=gnu99 -ffreestanding -msoft-float -mfpu=vfp  "../LCD/drv_hd44780_l.c" >>
'LCD/drv_hd44780_l.d'
Finished building: ../LCD/drv_hd44780_l.c

Building target: STM32-SK_LCD_Demo.elf
Invoking: C Linker
arm-elf-ld  ./main.o ./startup.o ./stm32f10x_adc.o ./stm32f10x_flash.o ./stm32f10x_gpio.o ./stm32f10x_it.o
./stm32f10x_lib.o ./stm32f10x_nvic.o ./stm32f10x_rcc.o ./stm32f10x_tim1.o ./syscalls.o  ./LCD/drv_hd44780.o
./LCD/drv_hd44780_l.o    -T"/home/michel/workspace/STM32-SK_LCD_Demo/STM32-SK-FLASH.ld" -nostartfiles -static -lc
-lgcc -L/usr/arm-elf/lib/thumb/thumb2 -L/usr/lib/gcc/arm-elf/4.4.0/thumb/thumb2 -Map=STM32-SK_LCD_Demo.map
-o"STM32-SK_LCD_Demo.elf"
Finished building target: STM32-SK_LCD_Demo.elf

Invoking: ObjCopyFlash
arm-elf-objcopy -O ihex STM32-SK_LCD_Demo.elf "STM32-SK_LCD_Demo.hex"
Finished building: STM32-SK_LCD_Demo.hex
```



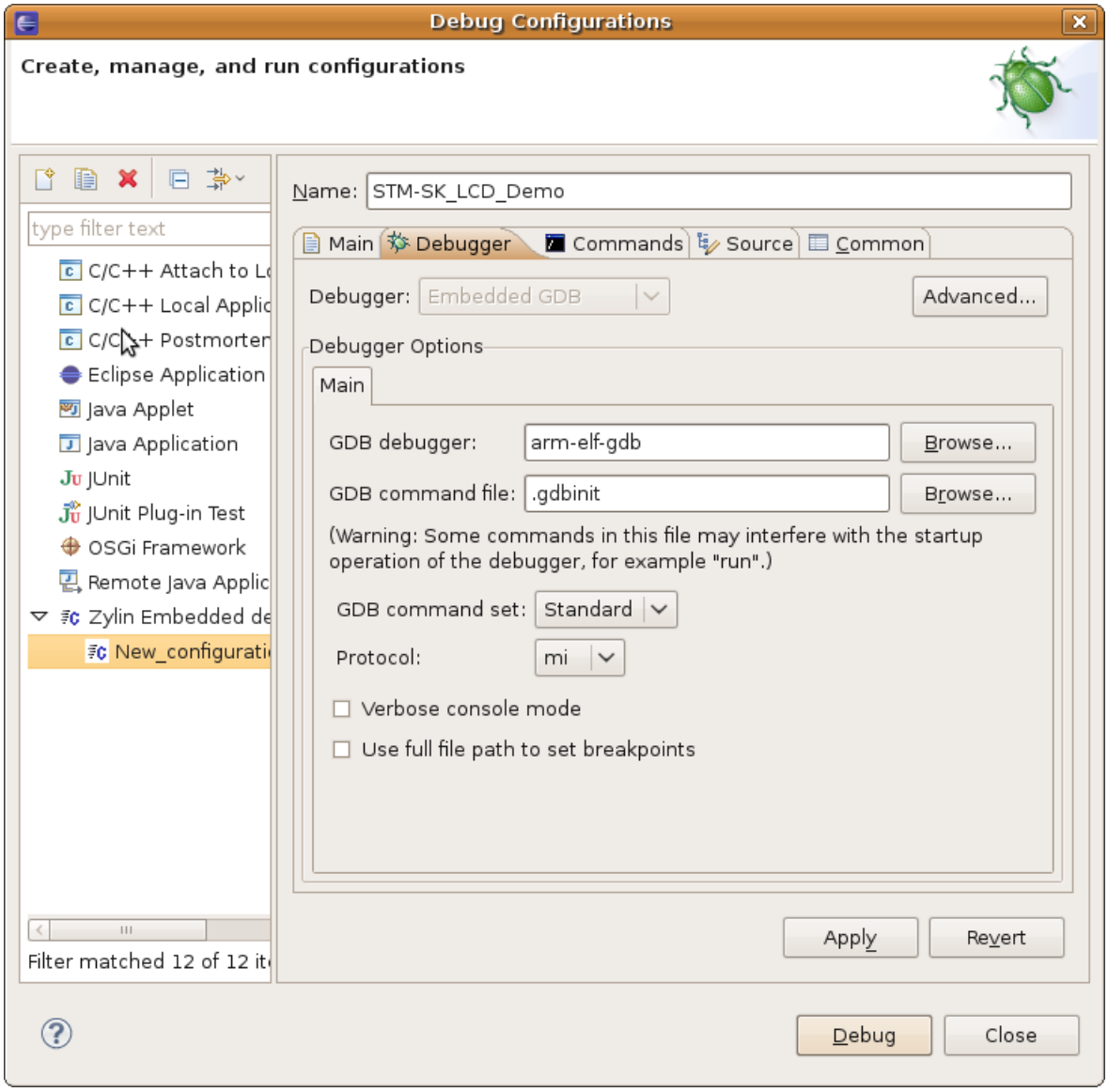
Now you need to setup some way to debug.
Click on Run and Debug Configurations.

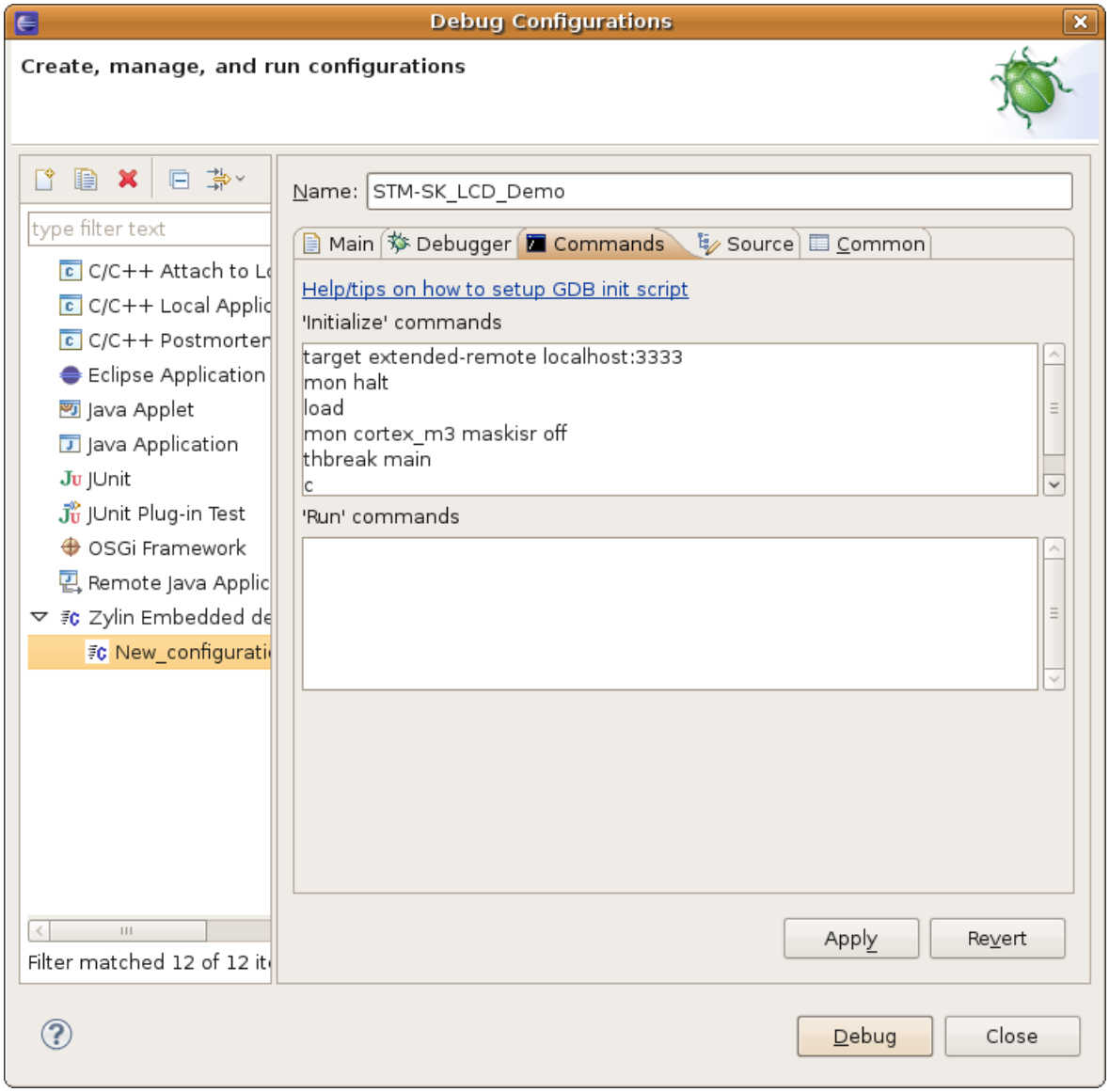


Double click on Zylin Embedded debug and type the name that you want for the debugger menu.


Debug Configurations
Create, manage, and run configurations
type filter text
C/C++ Attach to L
C/C++ Local Applic
C/C++ Postmorter
Eclipse Application
Java Applet
Java Application
JUnit
JUnit Plug-in Test
OSGi Framework
Remote Java Applic
Zylin Embedded de
New_configurati
Filter matched 12 of 12 it
Name: STM-SK_LCD_Demo
Main
Debugger
Commands
Source
Common
Project (optional):
STM32-SK_LCD_Demo
Browse...
C/C++ Application:
Debug/STM32-SK_LCD_Demo.elf
Search Project...
Browse...
Application console
Apply
Revert
Debug
Close

Switch to the Debugger section. Remove the .gdbinit in the command line entry since we will not use it. Make sure that arm-elf-gdb is the name of the debugger.





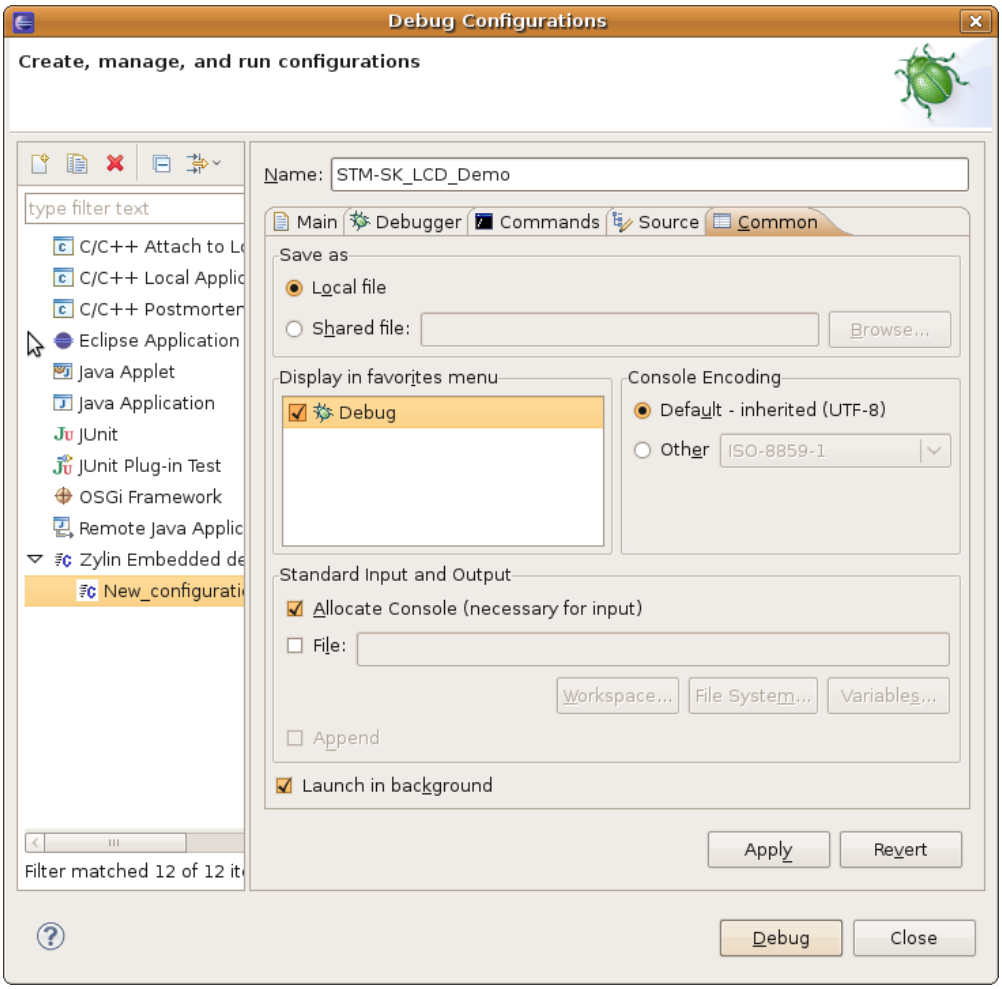
Instead of having a .gdbinit file we enter the startup commands in the "'Initialize' commands" section. The reason for that is so we can follow what is going on in the Console section and not wondering whether or not something has crashed.

The commands to enter are :
target extended-remote localhost:3333
mon halt
load
mon cortex_m3 maskisr off
thbreak main
c

mon is short for monitor. It is needed when the command is for OpenOCD and not a gdb command. The load command will program the flash with our program.
Because of a bug in OpenOCD where the maskisr is not restored. This cripples the interrupt structure, making it impossible to have any interrupt when debugging.

An alternative may be this. This would turn the interrupts off when breaking and reenable them when restarting. Not fully tested, I got that from a newsgroup.

target extended-remote localhost:3333
#turn off timers and peripherals while we are stopped
mon halt
#mask interrupts when we stop, re-enable when we turn back on.
define hook-stop
mon cortex_m3 maskisr on
end
define hook-continue
mon cortex_m3 maskisr off
end
load
mon cortex_m3 maskisr off
thbreak main
c



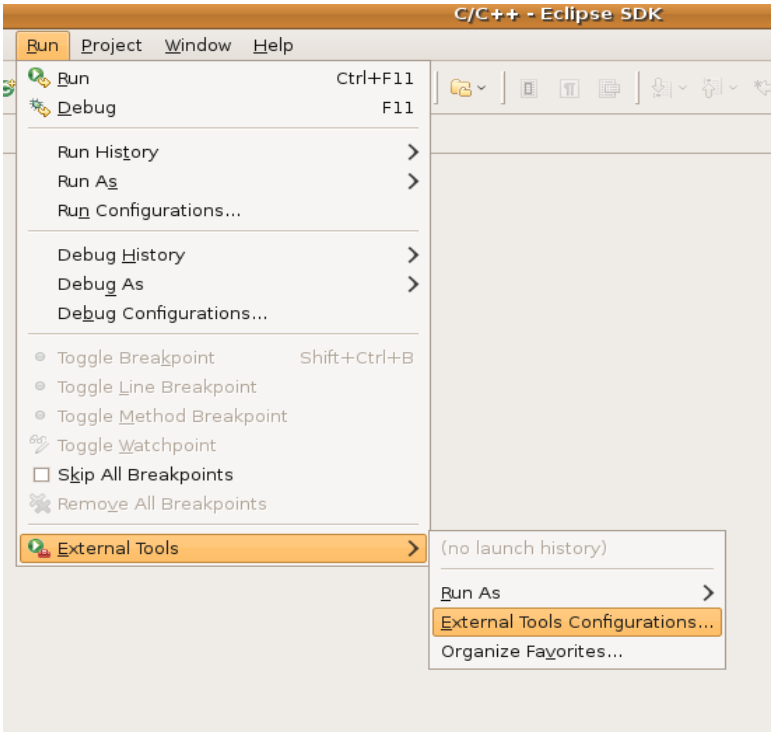
We unmark “Launch in background” so we know when loading is done and don't assume that it is crashed, it also gives us the option to cancel if we conclude that it did crash.
Mark the box near Debug so it add it to the top menu. You need to click on Apply when done and then close so you don't have to redo the whole thing.

You would probably want to create another launcher, on that doesn't include the load command for faster load when you don't need to reflash the device. The setup is almost identical to the one used to flash the device and go into debug.

The commands to enter for that launcher

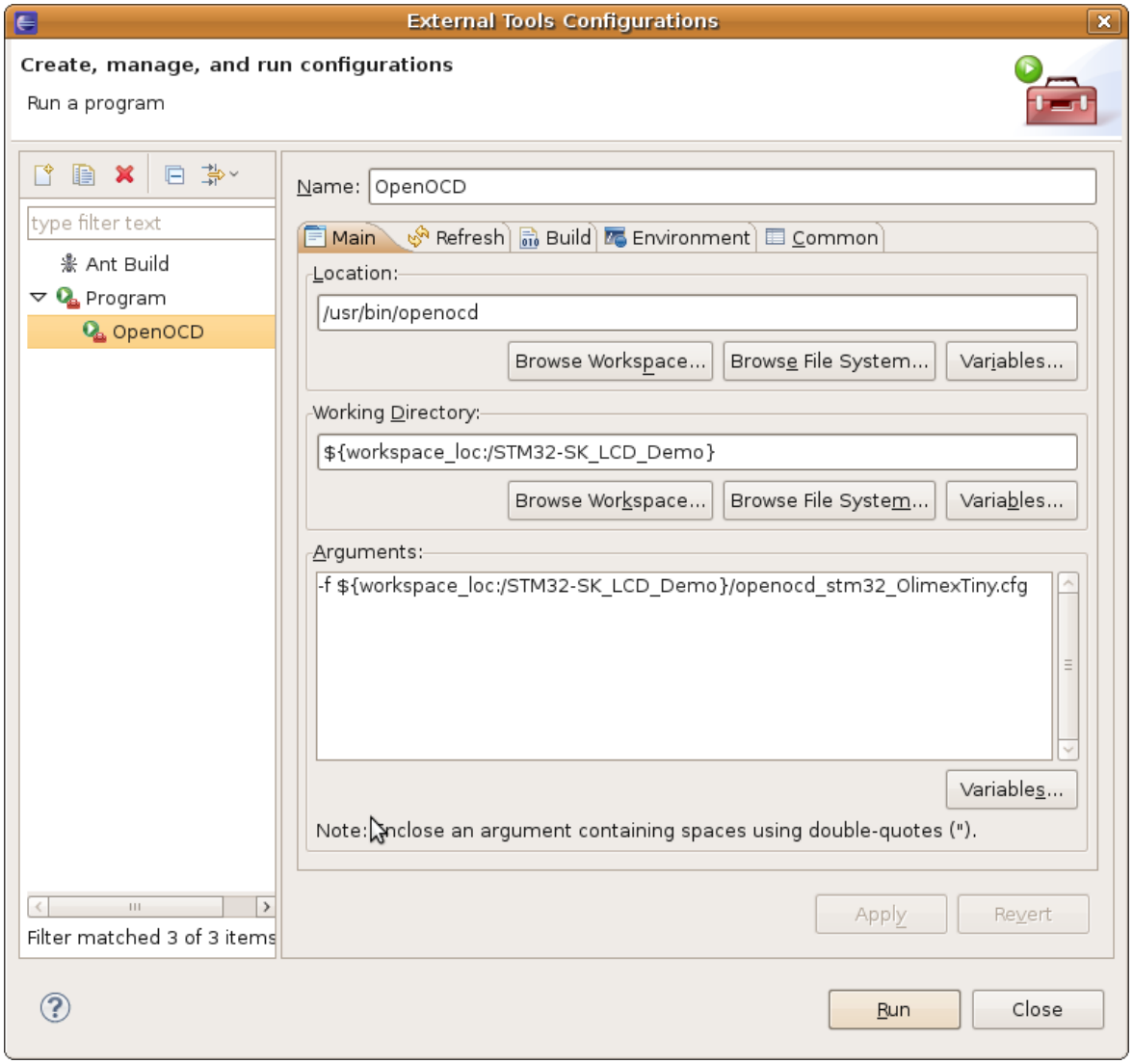
target extended-remote localhost:3333
mon halt
thbreak _startup
R
thbreak main
c

In this case the application is allready programmed and running. We have to stop it, reset the program, set a temporary breakpoint at main and restart the program. It will stop at the breakpoint. Keep in mind that only 2 breakpoints are allowed so using a fixed breakpoint for this would leave you with only one breakpoint. Any code that runs out of RAM is not bound with this limitation. When you need several breakpoints, you need to compile some of the code to run out of RAM. In that case you will have to make sure that your startup code copies the actual code in RAM otherwise this will hang. The startup file needs to be modified for that to occur.



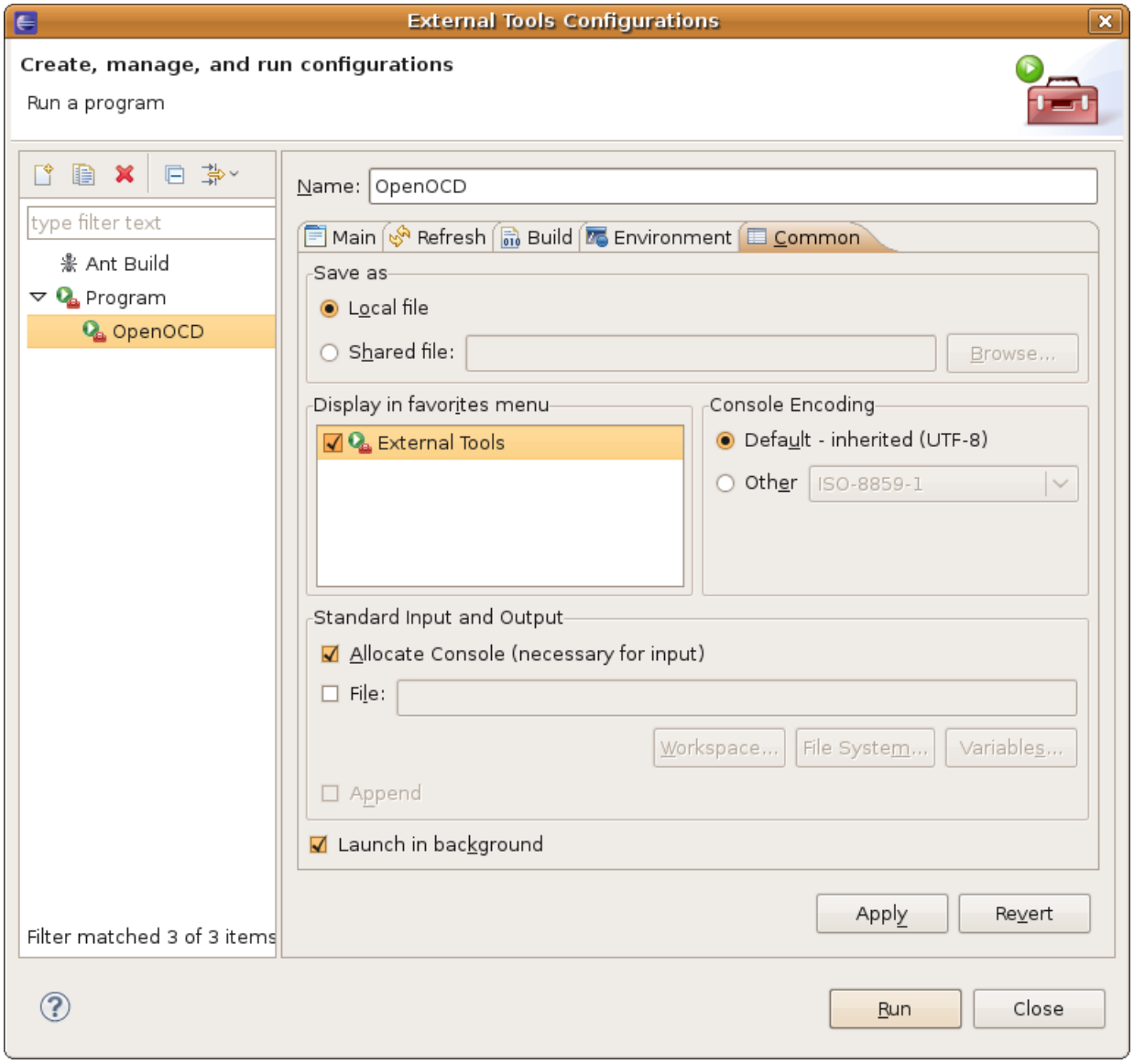
arm-elf-gdb communicates with OpenOCD with target extended-remote localhost:3333. We need to create a launcher to start OpenOCD before we attempt to load the debugger. Click on “External Tools – External Tools Configurations”

Highlight Program and click on new

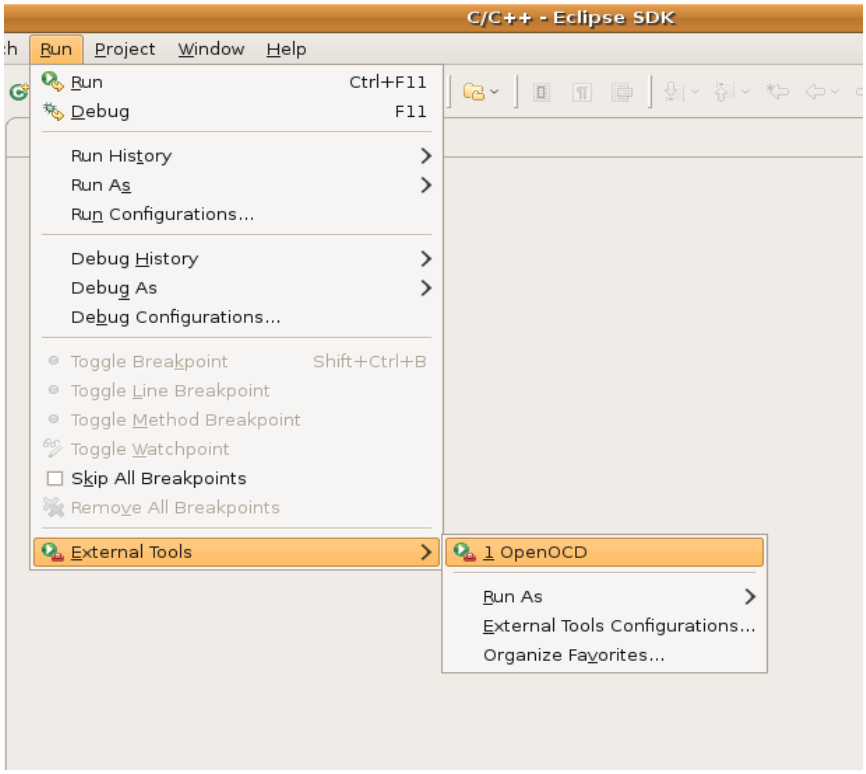


Type /usr/bin/openocd in the location. For the Working Directory click on Browse Workspace and accept the workspace where the project is. For the argument enter the configuration used for your JTAG programmer. If you use something else than jlink or Olimex Tiny you need to create your own configuration file.

Click on Apply and change to the Common page.



Mark the box for External Tools so it will appear as a menu.



Now we need to test our setup. Click on Run – External Tools -OpenOCD.

This will start OpenOCD.

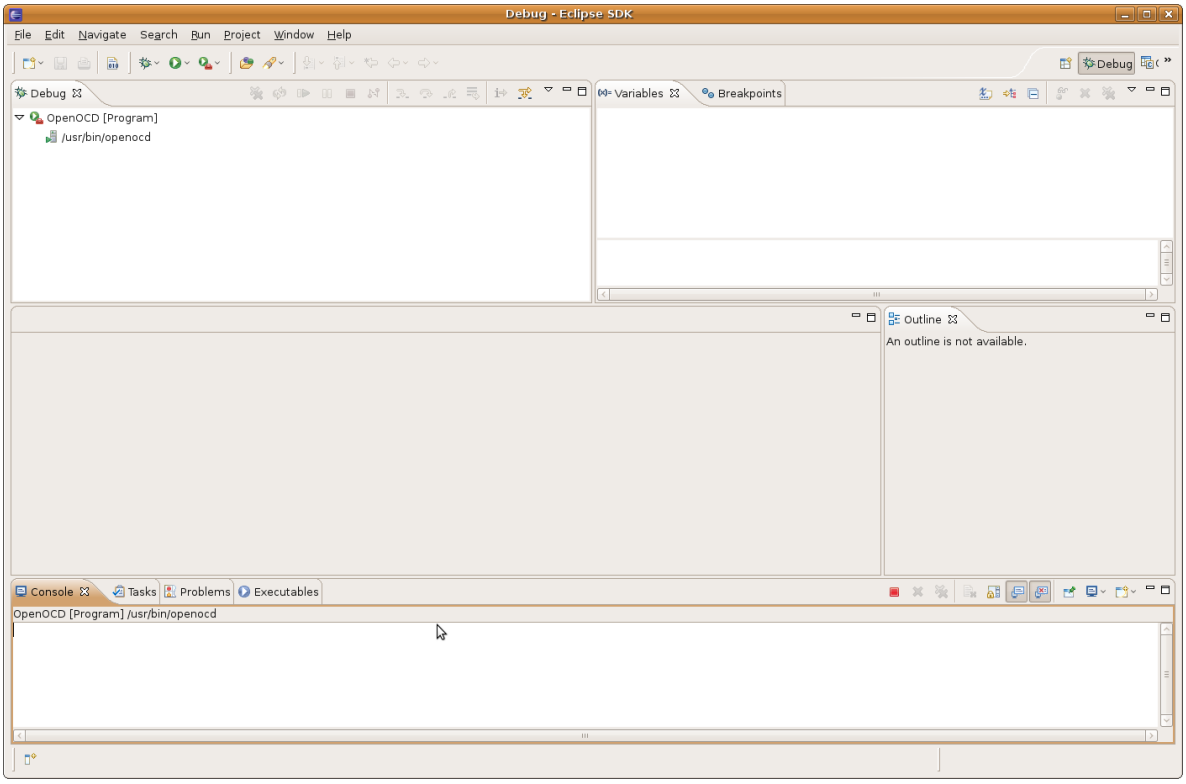
This will appear on the bottom console

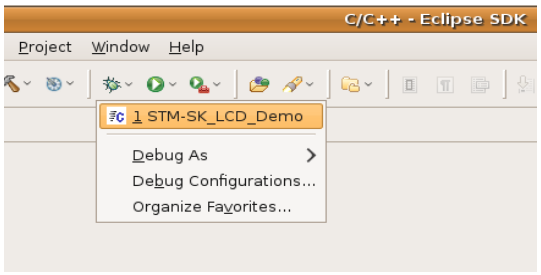
Open On-Chip Debugger 0.1.0 (2009-02-07-00:57) Release

BUGS? Read http://svn.berlios.de/svnroot/repos/openocd/trunk/BUGS

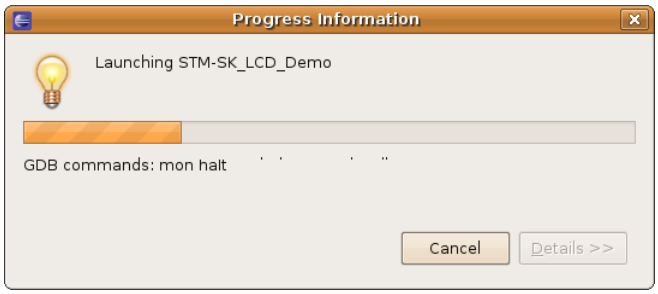
$URL: https://kc8apf@svn.berlios.de/svnroot/repos/openocd/tags/openocd-0.1.0/src/openocd.c $
jtag_speed: 20
500 kHz
Info : JTAG tap: stm32.cpu tap/device found: 0x3ba00477 (Manufacturer: 0x23b, Part: 0xba00, Version: 0x3)
Info : JTAG Tap/device matched
Info : JTAG tap: stm32.bs tap/device found: 0x16410041 (Manufacturer: 0x020, Part: 0x6410, Version: 0x1)
Info : JTAG Tap/device matched
Warn : no tcl port specified, using default port 6666

Now switch the perspective to debug. You should see the following window





Once OpenOCD is running we can start the debugger

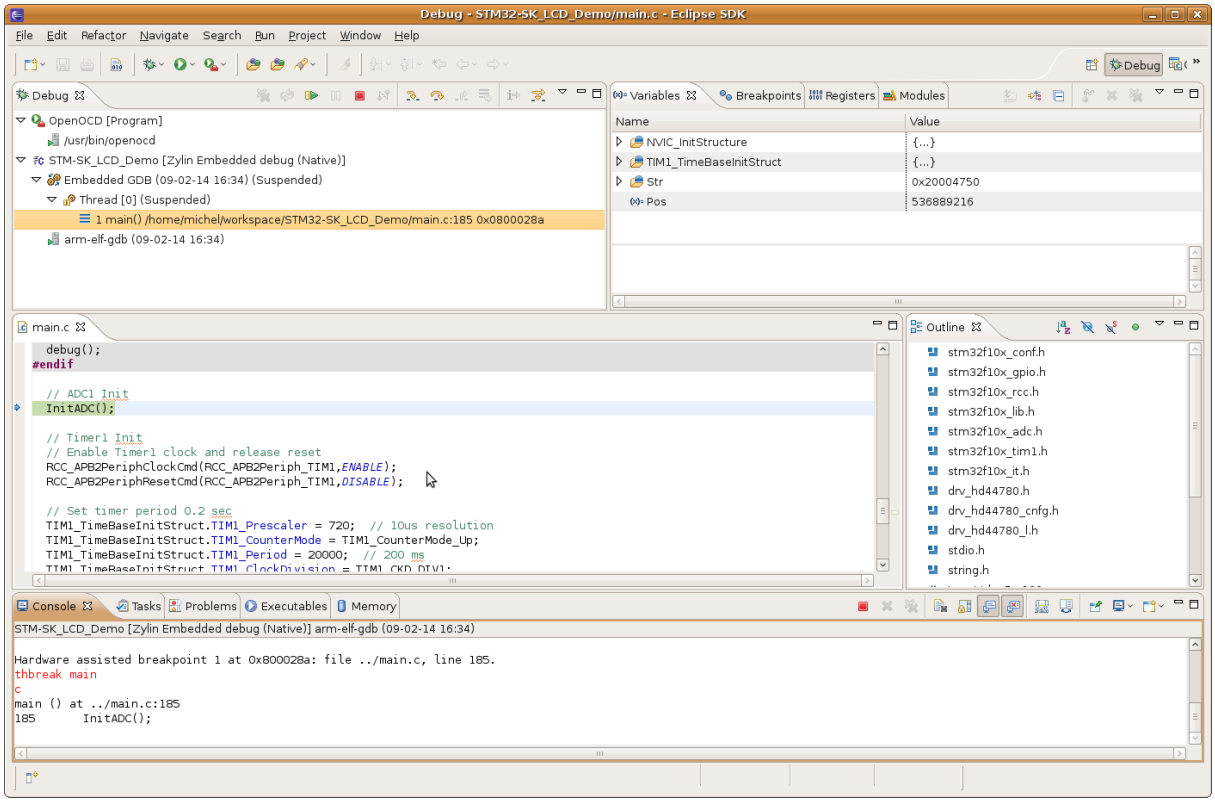


This window appears as the program is being flashed into the STM32 device. If it takes way too long press on cancel and retry again.
I have found that with that STM32-SK board reflashing always fails the first time right after power up but usually succeeds on the second try. To restart a failed load right click on the debug line for the project and click on Terminate and Remove. Then click on the debug menu again. It should work this time.

If you don't have something like this in the console section on the bottom there is a problem and you will need to restart until you get it to work right. I have not seen a case yet where it doesn't work the second time after power up.

```
target extended-remote localhost:3333
0xfffffffe in ?? ()
mon halt

Loading section .data, size 0x50c lma 0x20000000
load
Loading section .text, size 0x7458 lma 0x8000000
Loading section .eh_frame, size 0x2648 lma 0x8007458
Loading section .rodata.str1.4, size 0x34 lma 0x8009aa0
Start address 0x8000285, load size 40928
Transfer rate: 2 KB/sec, 8185 bytes/write.
mon cortex_m3 maskisr off
cortex_m3 interrupt mask off

Hardware assisted breakpoint 1 at 0x800028a: file ../main.c, line 185.
thbreak main
c
main () at ../main.c:185
185    InitADC();
```

You are now ready for debugging.



If you feel that this has been helpful to you feel free to make a donation with paypal at mcatudal@comcast.net

Revision History

Release 1 : First draft