Open source Coldfire IDE

Set up and configuration guide

Draft 1.2

Date 18 April 2008

Written by:

Paul McConkey

paul.mcconkey@cambridgeimaging.co.uk

Open source Coldfire IDE

Introduction

Here is a description of how you can set up a professional quality integrated development environment for the FreeScale ColdFire CPU family. You may be able to adapt some of the details for other target CPUs.

If you are used to working with commercial IDEs for PC platforms it can be a bit difficult to know where to start when you are trying to set up an IDE for an embedded environment. We had this problem when we tried to find a suitable development environment for a ColdFire based product that we were developing. Our main area of expertise is development of server software for Microsoft platforms, using C++ and .Net with Visual Studio. We also use PHP and have some experience of Linux.

We used the demo versions of CodeWarrior with the FreeScale development boards for the mcf5282 and mcf52335 CPUs and managed reasonably well, but we found the CodeWarrior interface awkward to use. It also seemed very expensive.

We decided to search for an alternative. We required an IDE with a visual debugger, good editing tools, preferably with modern functionality such as code completion and code browsing tools. It should integrate with SubVersion for source control and it needed to be usable day in, day out by professional programmers. In addition, it should be able to run natively on Windows platforms as the learning curve for embedded development was steep enough without adding a Linux learning curve as well.

Components

After several weeks we had managed to collect the right tools together and we now have an IDE that exceeds all of our requirements. The components are as follows:

Eclipse – an open-source, Java IDE that was originally developed by IBM and then donated to the open-source community. This is on of the most professional quality open-source applications that you will find and it has been adopted by many developers, manufacturers and tool chain vendors for hundreds of development and target platforms.

Eclipse CDT 4.0 and GDB Hardware debugging plug-in – Eclipse is no use to us unless we can use its visual debugger with our embedded target. If you are able to use a Windows natively hosted tool chain, then these components provide a C\C++ IDE along with the ability to debug a remote target using BDM or JTAG tools.

Zylin Eclipse CDT plug-in – This is an alternative to the above, which works much better with Cygwin hosted tool chains. You will need to use this if your project uses a *nix-style filing system.

Subclipse – a SubVersion client that integrates with Eclipse to provide a full-featured source control system (assuming you have a SubVersion server running somewhere). There is an alternative to Subclipse called Subversive.

CodeSourcery Sourcery G++ - a version of the GNU tool chain ported for ColdFire and ARM processors. This runs natively on Windows platforms.

Support

None of this software comes with support. If you have an interesting problem and you both research it well and describe it well on an appropriate forum, you may get an answer which helps you out. Or you may not.

Don't worry if you think that your question is very naïve or too simple, as long you show that you have researched the problem appropriately (Google, FreeScale and GNU documentation, etc.) then the easier questions usually get good answers. Just don't expect others to do your job for you!

Unless you are prepared to go it alone and expect to have to learn a lot by reading documentation you should consider buying a commercial development package. Although the solution described here doesn't need you to get out your credit card, you must be prepared to invest your time. Your time may be money in which case there will be a cost!

One of the best value packages around has to be that offered by CodeSourcery – as well as distributing their open-source GNU tool chains for free, they will also license you a full-featured Eclipse-based package which comes with full support for a very reasonable price. Their version of Eclipse also supports managed make files so you will not have to climb the almost vertical learning curve for GNU Make!

What you need before you start

You need a PC running Windows. We use XP but other versions may also work. As with any professional IDE you want as many big screens as you can get hold of!

You need a Coldfire development board and a P&E BDM cable. We have used the FreeScale M52233DEMO and M5282LITE boards along with the P&E parallel and USB cables. Get the USB cable if you can as it is quicker. The M52233DEMO board has an integrated P&E BDM interface so it only needs a USB cable, but the mcf52235 processor has no external bus so it may not suit you.

P&E are at http://www.pemicro.com/

You do not need the P&E drivers as the CodeSourcery tool chain includes them.

How to get the software

Download locations for all the packages are given below. Please note that the locations given are correct at the time of writing (January 2007) but they will undoubtedly change as new versions are released. If the links are broken, search the websites for the right links.

Note: Subclipse will be downloaded and installed after Eclipse is installed.

Download the latest Java runtime if necessary:

You need to make a choice between the Eclipse version of the CDT and the Zylin version. As mentioned earlier, the Zylin version works better with Cygwin hosted toolchains.

If you are going to go the Eclipse route, go to:

> http://www.eclipse.org/downloads
>
> *Download the 'Eclipse IDE for C\C++ developers'.*

Download the CodeSourcery tool chain:

> http://www.codesourcery.com/gnu_toolchains/coldfire/index.html
>
> *Click on the 'Downloads' tab to find them.*

Download the hello_5282 example from http://www.cambridgeimaging.co.uk/Hello_5282.zip.

Installing the software

Run the Java runtime installer.

Run the CodeSourcery installer. Allow the installer to add the Sourcery G++ Lite path to your PATH environment variable.

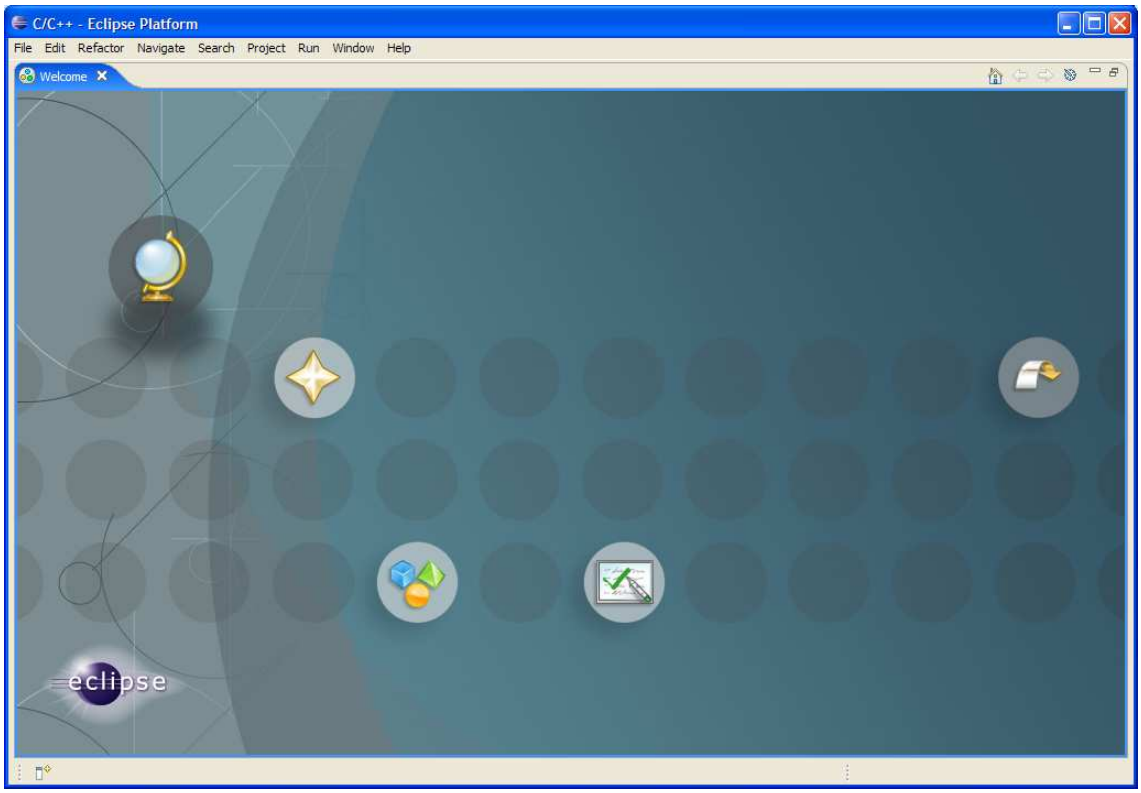
Unzip the Eclipse Platform zip file to c:\ - it will create a folder called c:\eclipse.

Unzip the two Zylin zip files into c:\eclipse if you are using the Zylin CDT.

Copy the startup.bat and shortcut files from the hello_5282 zip file into the c:\eclipse folder

Configuring the software

1. Start Eclipse by double clicking on the shortcut.
2. You have to create a workspace, so make a new one in a folder called c:\eclipse\workspace.
3. The welcome page has several icons on it - select the workbench icon (the arrow on the right hand side), your welcome screen may not be exactly the same.



4. Open the 'Window->Open Perspective' menu and select 'C\C++'.
5. You now need to add a plugin that adds a hardware debugging extension. This integrates Eclipse with GDB and a JTAG or BDM connector so that you can debug code on your development board.

   Either add the Zylin plugin (which seems to be better for Cygwin):

   a. Select Help->Software Updates->Find and Install
   b. Select 'Search for new features to install'
   c. Select 'New Remote Site'

d. Enter 'Zylin CDT plugin' for the name and http://www.zylin.com/zylincdt for the URL

e. Select Finish

Or add the CDT plugin;

a. Select Help->Software Updates->Find and Install

b. Select 'Search for new features to install'

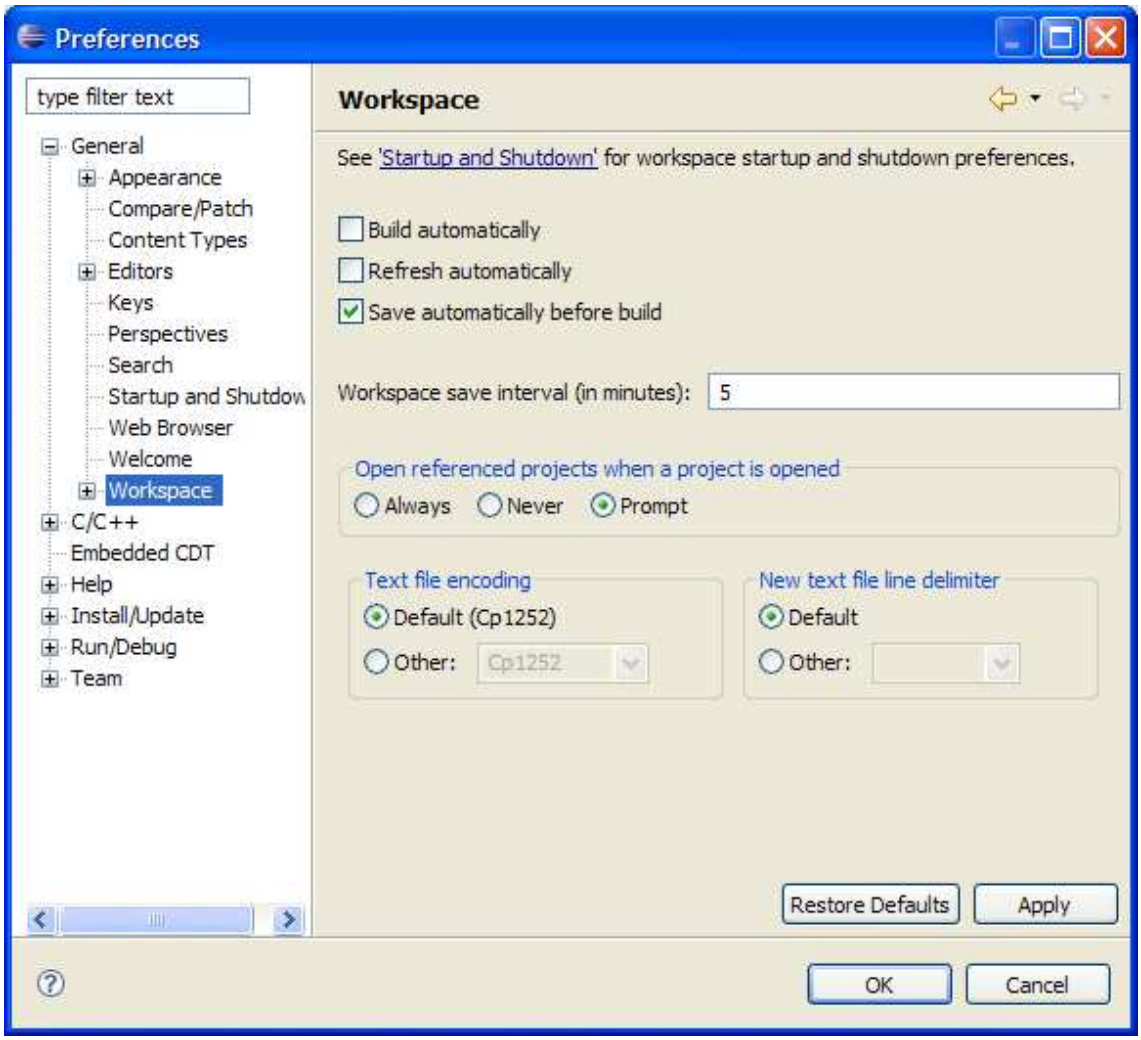
c. Select 'CDT Updates'

d. Expand the 'CDT Updates' and 'CDT Optional Features' branches in the search results tree and select 'Eclipse C/C++ GDB Hardware Debugging'

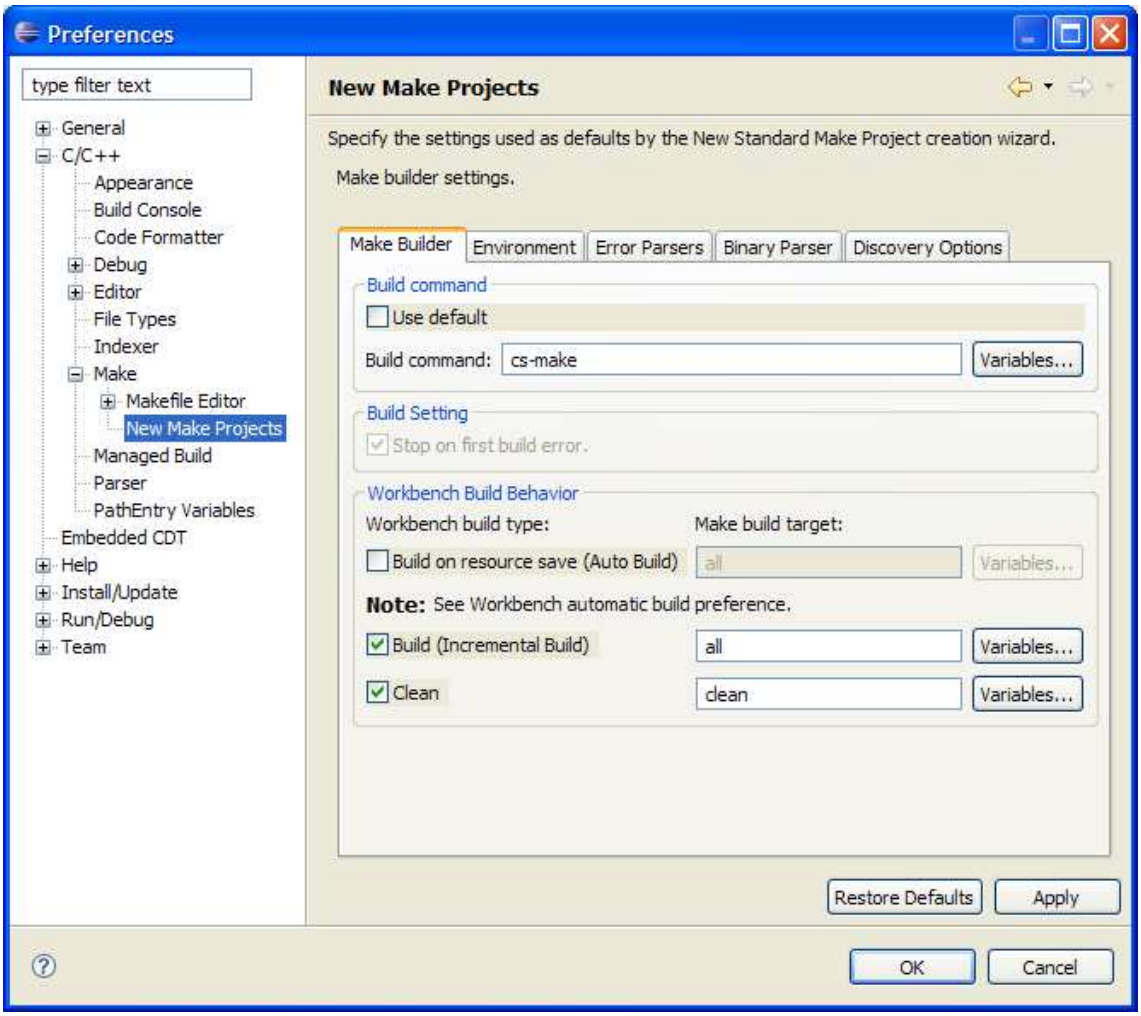
e. Select Finish

The following instructions assume that the CDT plugin has been installed, but the differences for the Zylin plugin are trivial and the alternative instructions should be obvious.

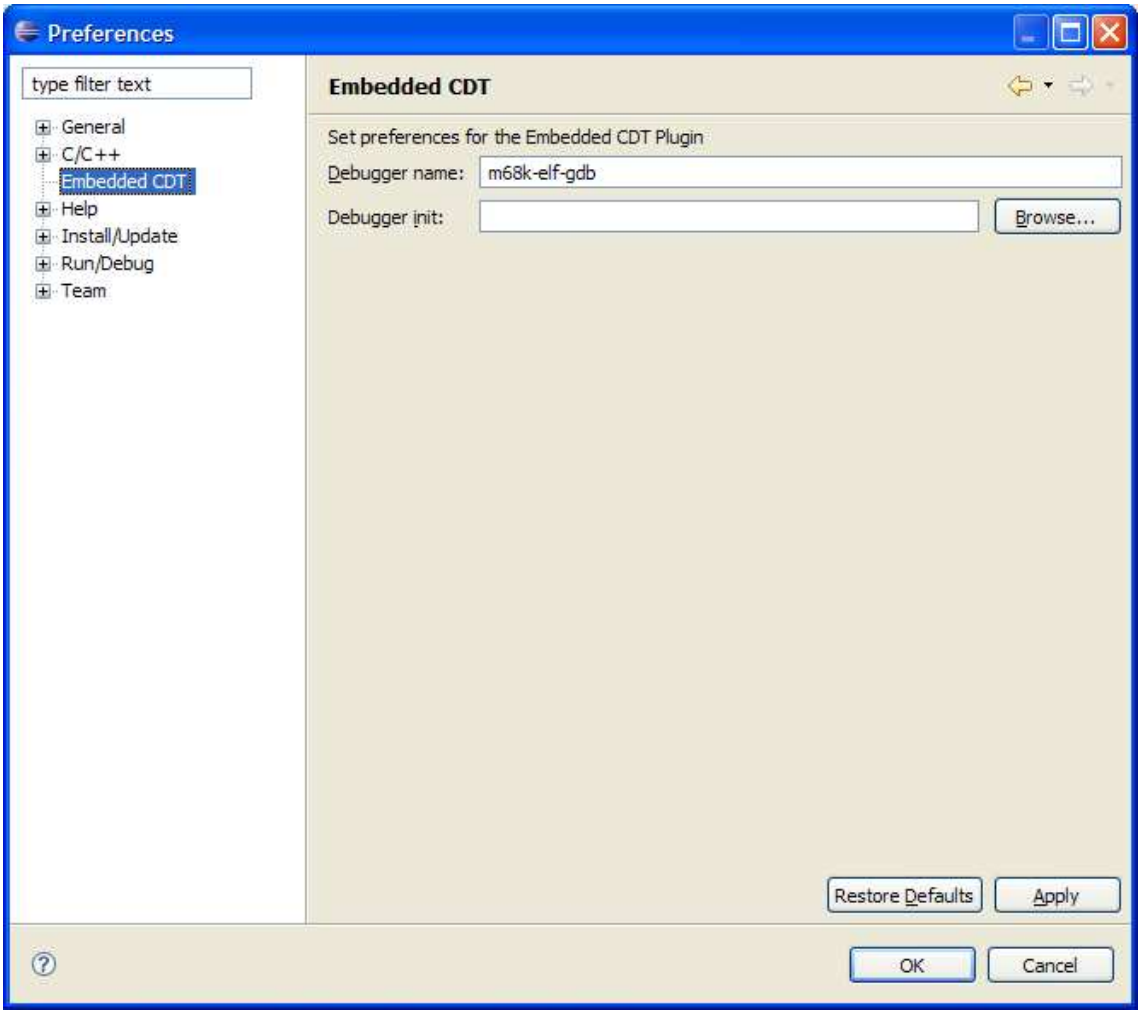
6. You can skip this step, but if you are used to the IDE saving everything before it does a build you will save problems later by selecting 'Preferences...' from the Window menu. Expand the 'General' branch, select 'Workspace' and check 'Save automatically before build'.



7. Expand the ‘C/C++’ branch and then expand the ‘Make’ branch and select ‘New Make Projects’. Change the ‘Build command’ to cs-make.



8. Select the 'Embedded CDT' tab and change the debugger name to m68k-elf-gdb. This tab will have a different name depending on which hardware debugging extension that you use.



9. If you want to add source control to Eclipse, this would be a good time to install Subclipse. You can do this using the built in update functionality in Eclipse. Instructions are at http://subclipse.tigris.org/install.html.

How to create a project

We have provided a sample application so that you can try out the IDE for yourself. The example requires no operating system or boot loader on the target board and it doesn't attempt to demonstrate the best way to create a project.

For real life use you may need an operating system or at least a network stack. If you are creating a bare-metal project (no boot loader or OS) you will need startup code to configure your board's memory map and initialise the system. You will also need drivers for any peripherals such as UARTs and the FEC that you may want to use. A good source for this is example code provided by FreeScale from their website. Much of FreeScale's code is written with CodeWarrior and so it needs to be tweaked to compile with a GNU tool chain.

The example provided here uses the standard NewLib library. It uses a hardware initialisation hook in the CodeSourcery startup code to initialise the UART and PLL. It also provides a very simple replacement for NewLib's write() function to transmit data to the UART.

If you want to know what is going on, you will have to read the mcf5282 Coldfire Microcontroller User's Manual, which you can download from the FreeScale website. This book will tell you everything you need to know about the ColdFire but sometimes the information you need is very hard to find. We have been stuck several times, trying to figure out why something won't work and the answer is found as a single line of text in the middle of 600+ pages of information!

The make file provided in the example works for us. It is not ideal as it doesn't make the object files depend on the header files properly. However, it will keep all your object files away from your source files and it works with Windows paths. It relies on touch.exe being available on your path.

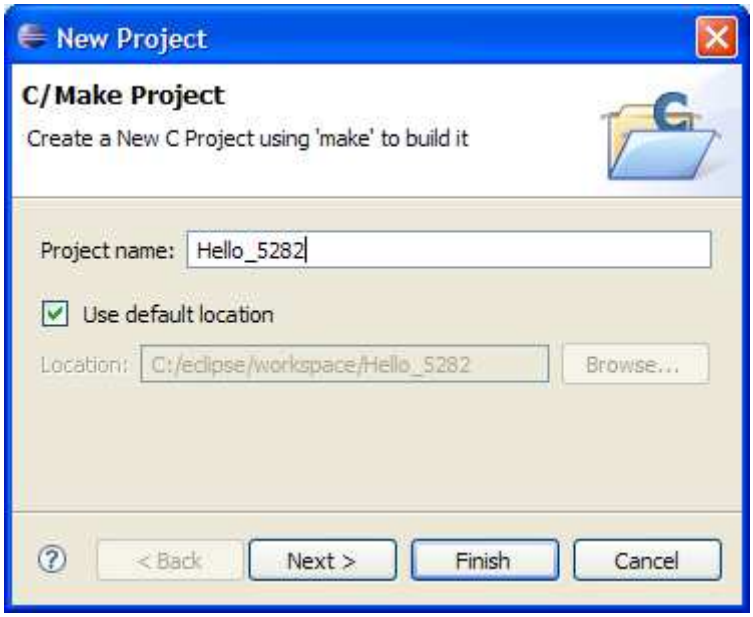
You can get touch.exe from http://gnuwin32.sourceforge.net/packages/coreutils.htm.

If you can't get the example to work on your platform then, sorry, but we can't help. It works for us. If you figure out how to make it work and think that others will be interested then by all means let us know and we'll add the information here.

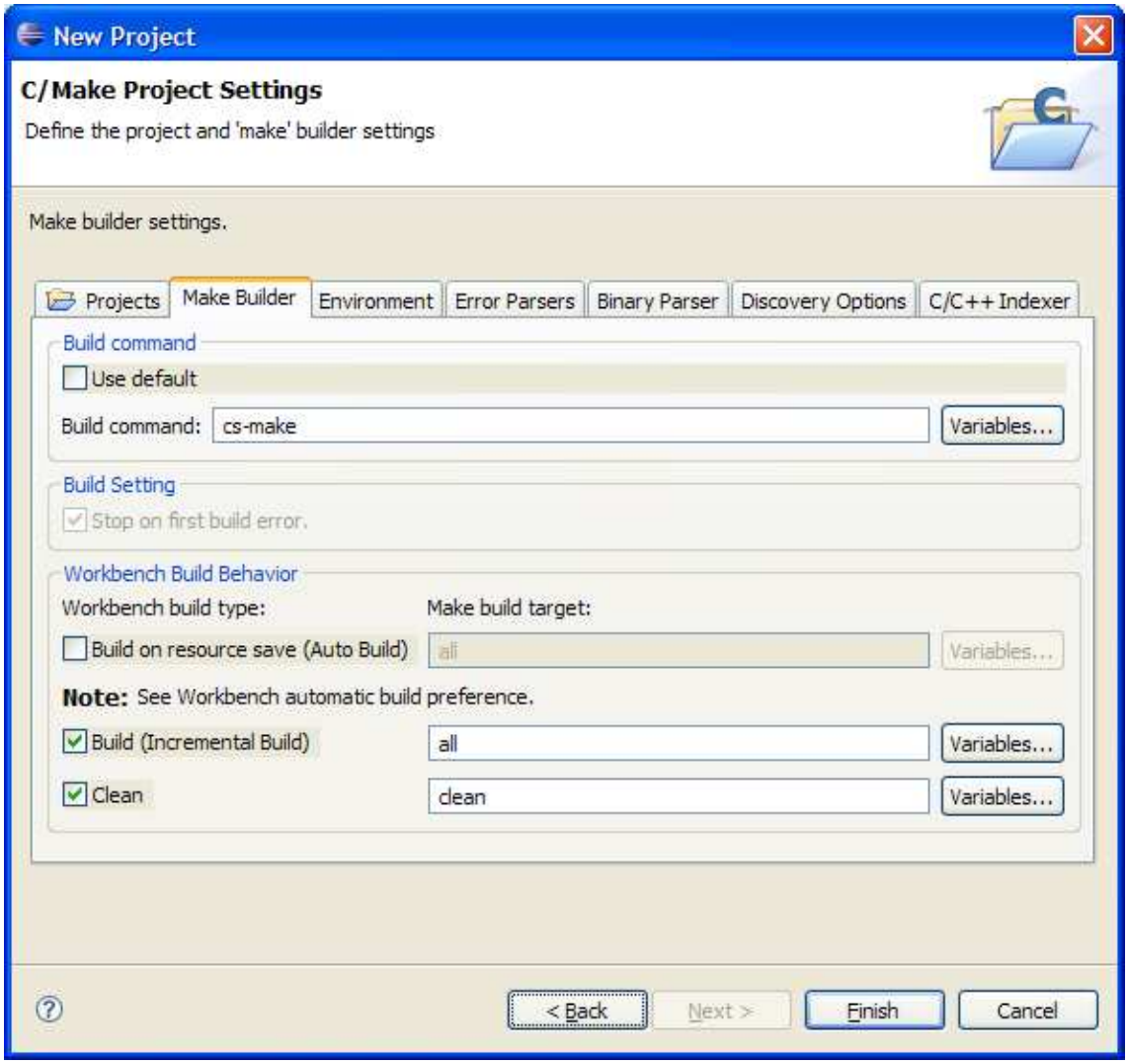
The hello_5282 example

Unzip the hello_5282.zip file somewhere temporary. You can create new projects and files from scratch, but that is quite easy and well described in the Eclipse help. We will import the file from the hello_5282 example into a new project to save a bit of time

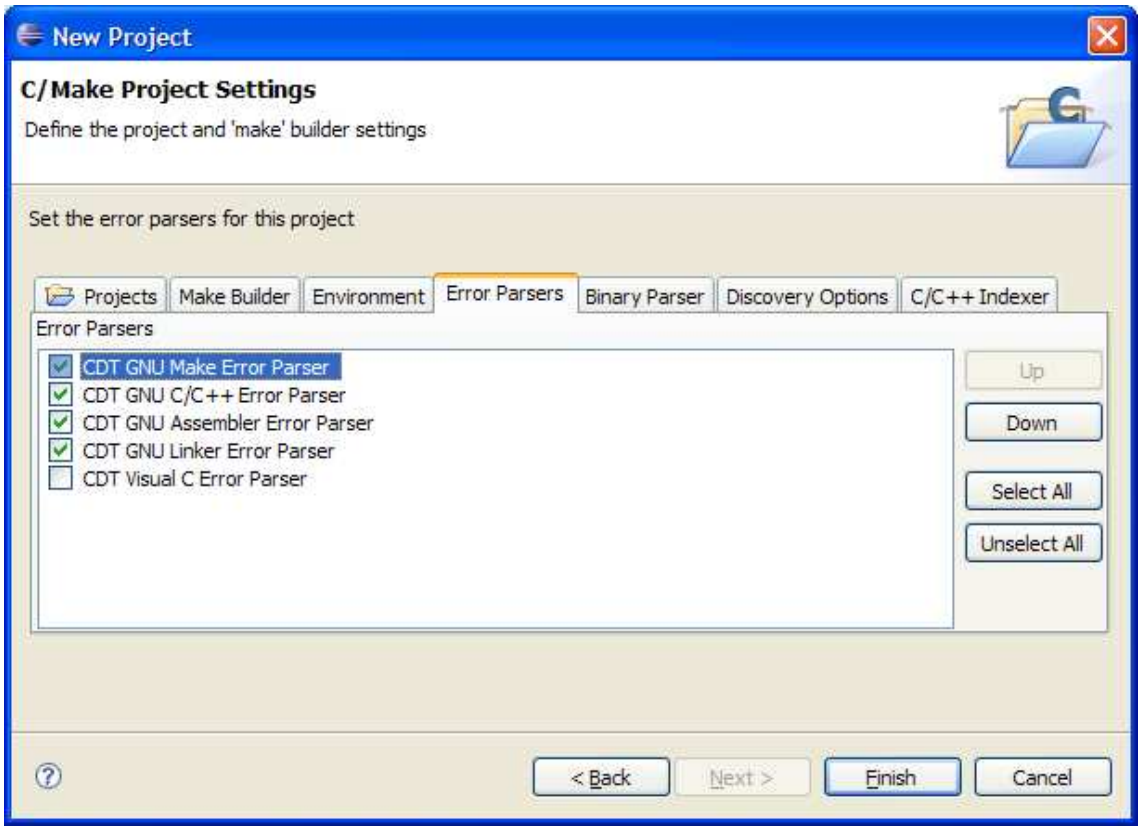
From the File->New menu select 'Standard Make C Project'. Enter the project name in the dialog and click the 'Next' button.



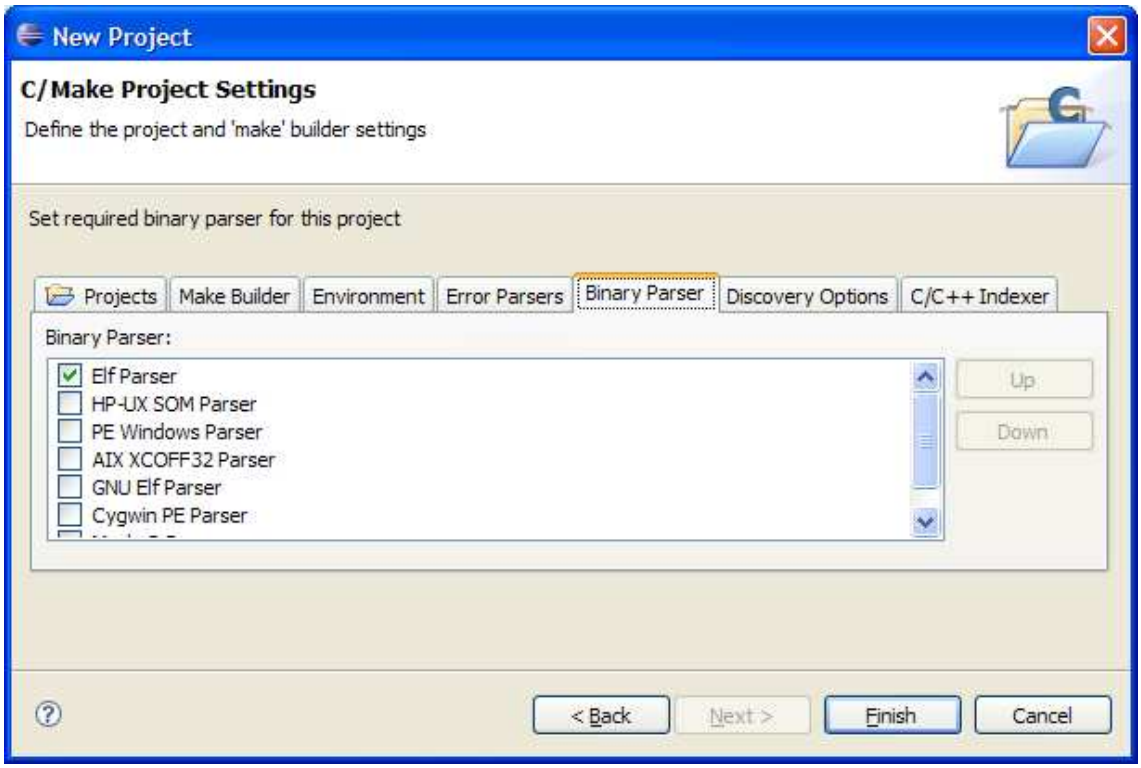
You need to change some of the default settings in the next dialog:



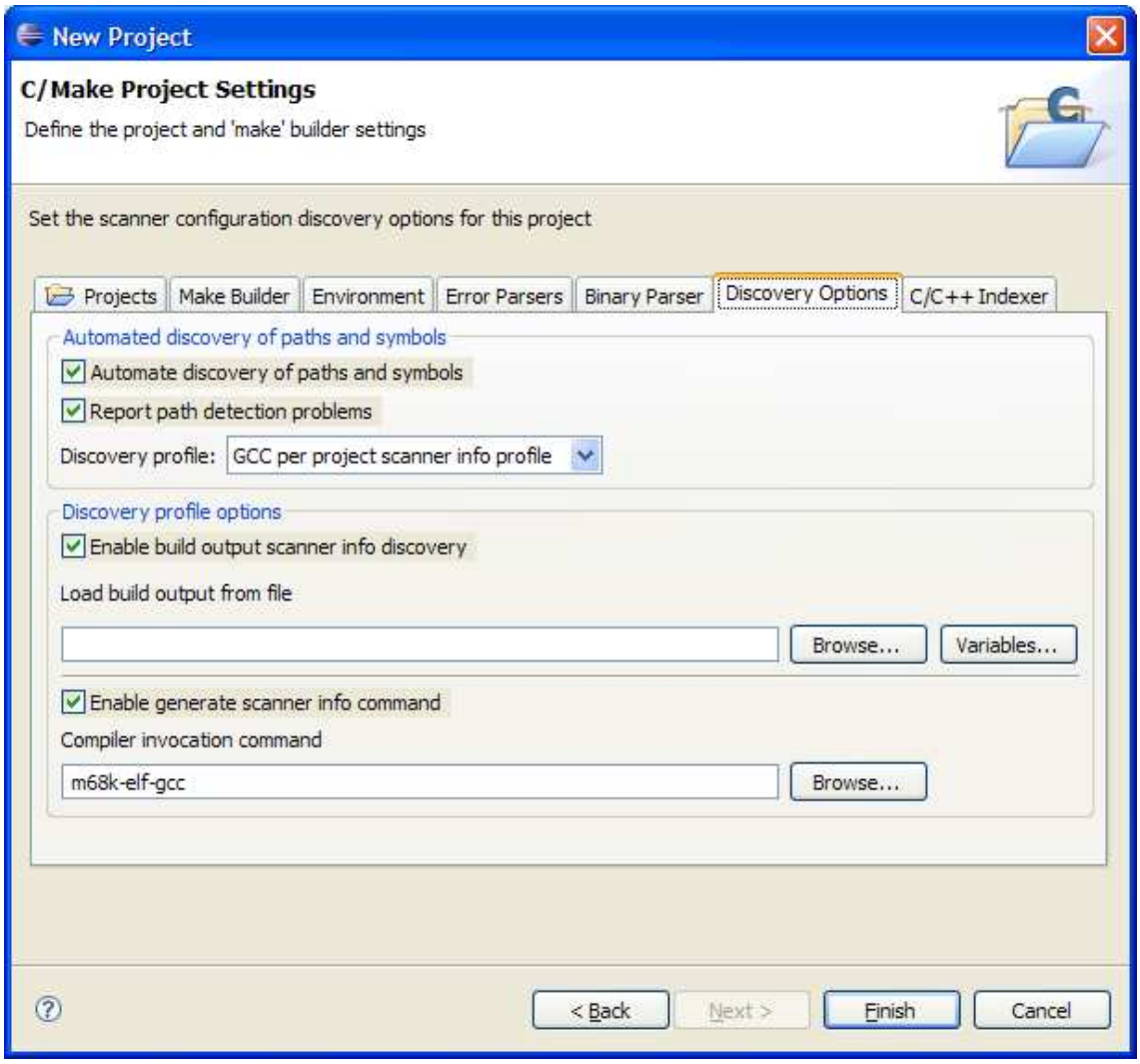
Check the 'Make Builder' tab – the 'Build command' should be 'cs-make'.



You may as well remove Visual C error parser.



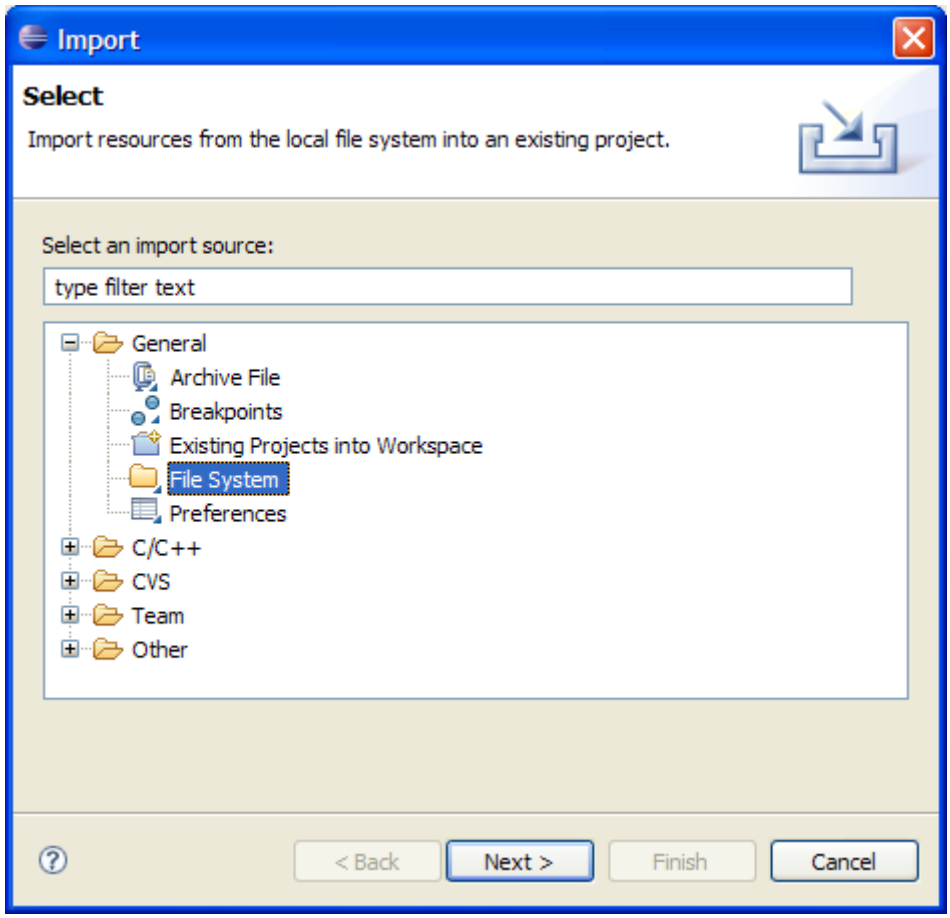
The binary parser should be 'Elf Parser'.



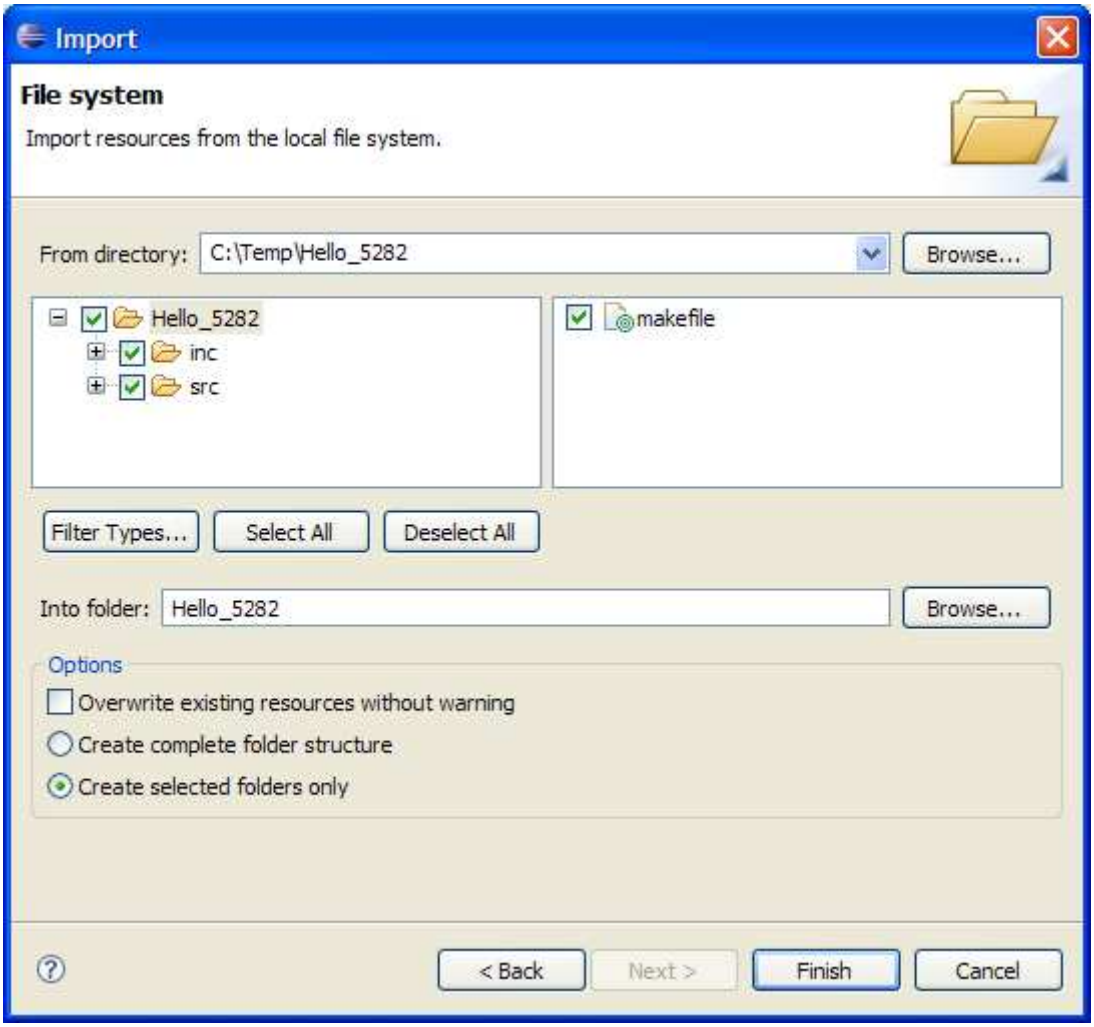
On the 'Discovery Options' tab, 'Compiler invocation command' should be 'm68k-elf-gcc'

Click on the 'Finish' button and the project will be created. You will see a new project in the Navigator view called Hello_5282.

Right click on the new project name and select 'Import...', this dialog is displayed:



Expand the 'General' branch and select 'File system'. Click on the 'Next' button.



Browse to the folder containing the Hello_5282 example project. The project folder will be displayed with an empty check box beside it. Click the checkbox to select the entire sub-tree and click on the 'Finish' button.

The directory tree and files will be copied to your workspace folder and the will be added to your new project.

Build the project

We will now add a couple of make targets to the perspective.

On the right-hand side of the C/C++ perspective select the 'Make Targets' tab.

Right click on the project name and select 'Add Make Target'

Target name and make target should both be 'all';

'Build command' should be 'cs-make' and 'Run all project builders' should be checked.

Click the 'Create' button.

Now for the 'clean' target:

Right click on the project name and select 'Add Make Target'

Target name and make target should both be 'clean';

'Build command' should be 'cs-make' and 'Run all project builders' should be checked.

Click the 'Create' button.

Double click on the 'all' make target and the compiler will be launched.

The Console window will show output and any error messages.

If the build fails then you may need to check that the tool chain is correctly installed and on the current path (enter 'm68k-elf-gcc' at a command prompt and you should get an error along the lines of 'm68k-elf-gcc: no input files', if you get a 'File not found' error you need to fix your path to point to the Sourcery G++ bin folder).

If the compiler is found and still you have errors, then you will have to read the error messages and try to work out what is going wrong. Most likely, you have a syntax error in the code or a problem with the way you have installed the tool chain and Eclipse.

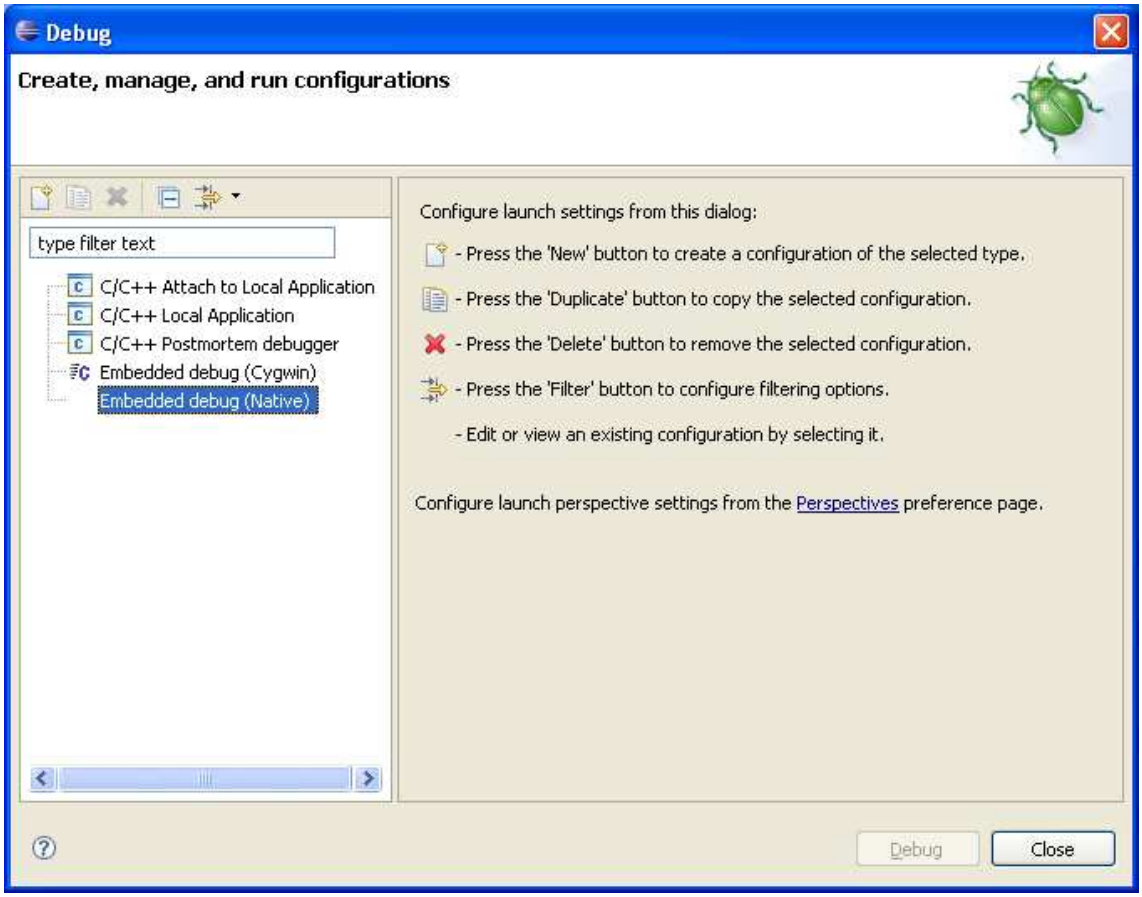
You can always test the tool chain independently of Eclipse from a command prompt. Change to the directory containing the make file and enter 'cs-make'.

Running the project

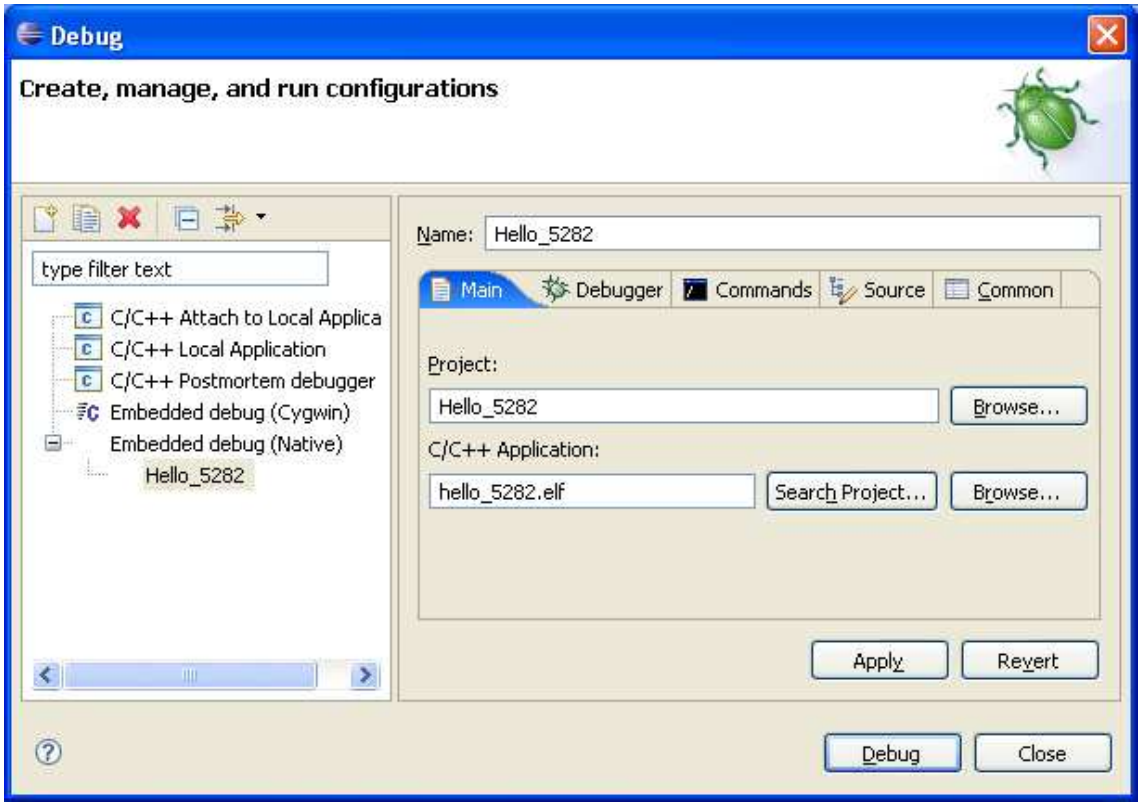
The final steps that you need to take before running the example are to power up and connect your board. If you have a USB BDM cable then Windows will prompt for drivers when you connect the cable. The drivers have already been installed by the CodeSourcery package so Windows should find them.

The make file and linker script used by the example will build an executable that will run in RAM. We now need to configure Eclipse so that the executable will be loaded onto the board via the BDM cable.

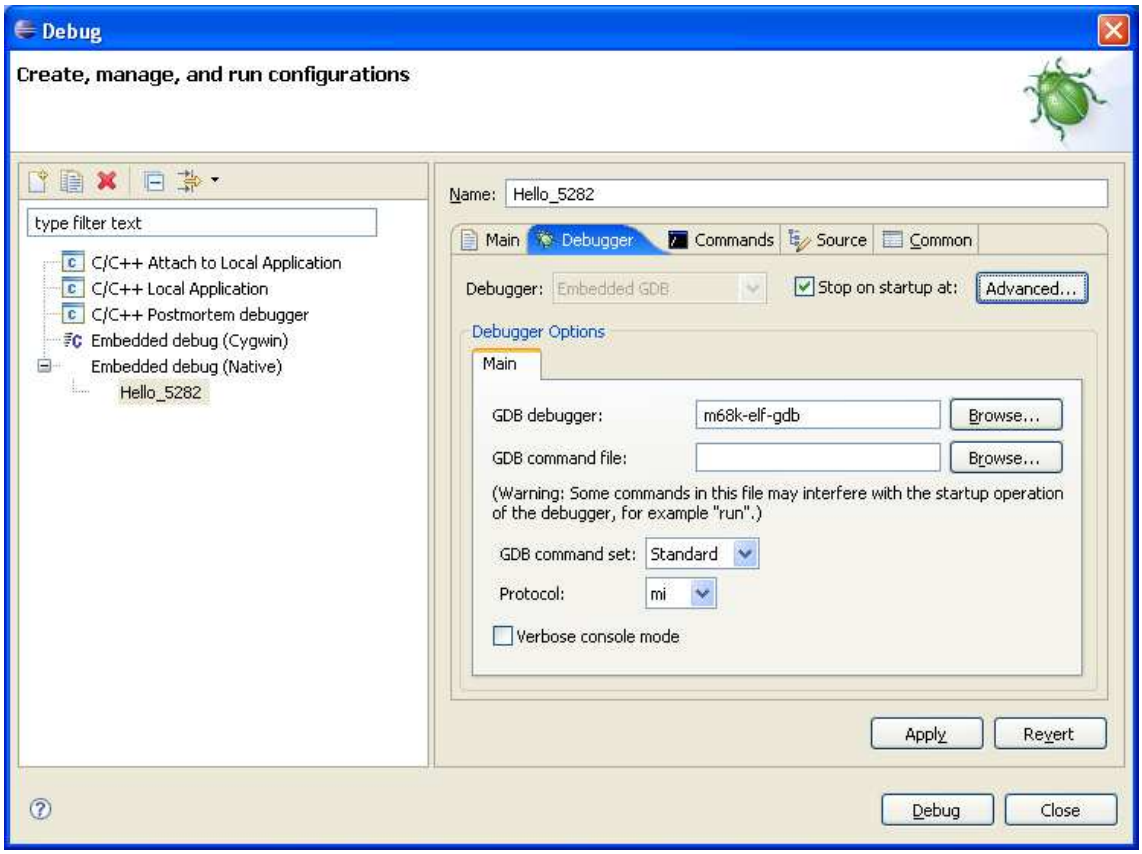
Select 'Debug...' from the 'Run' menu.



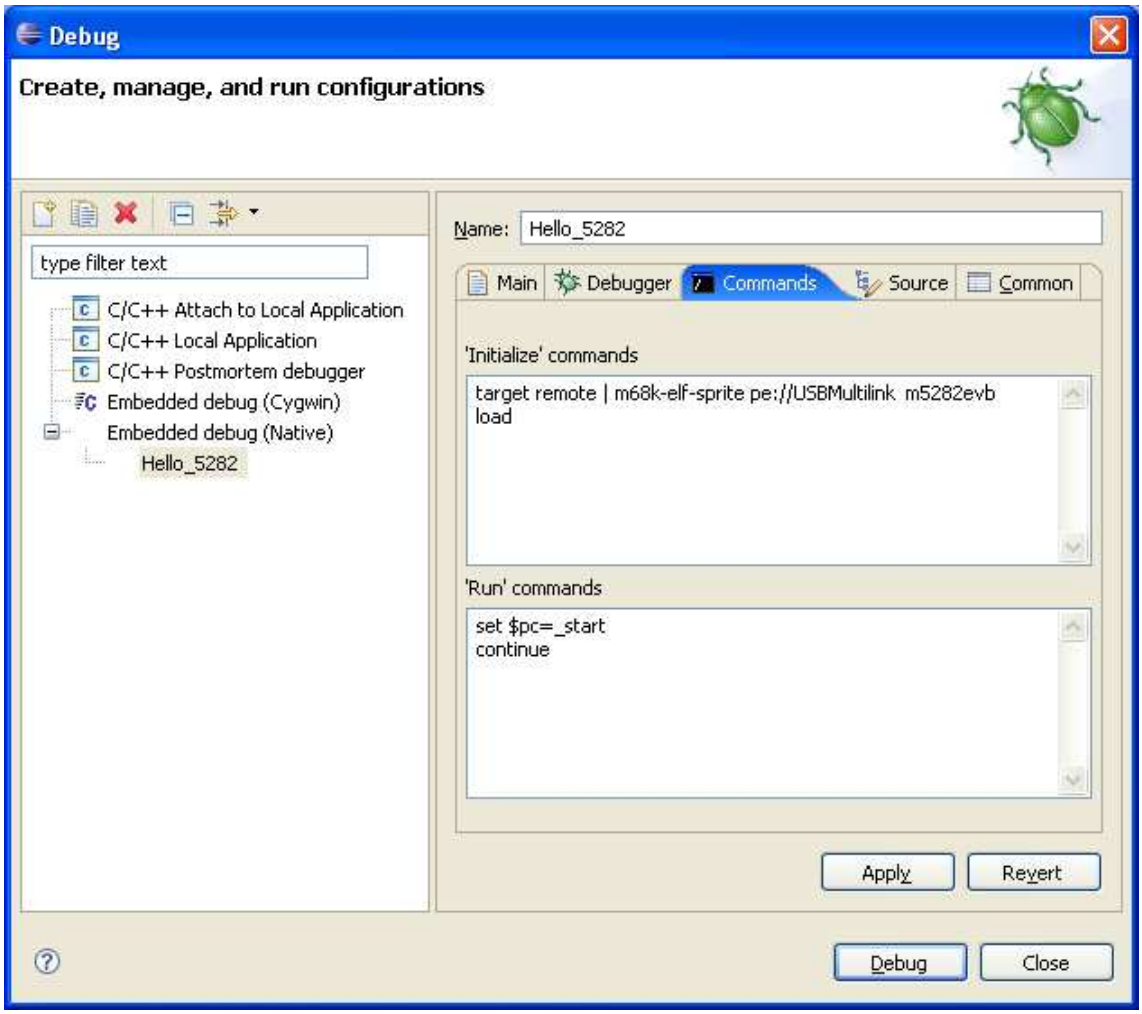
Select 'Embedded debug (Native)' and click on the 'New' button to create a new configuration.



Enter the name of the 'C/C++ Application' as 'hello_5282.elf'.



Select the 'Debugger' tab and check that the name of the 'GDB debugger' is 'm68k-elf-gdb'.



You now need to enter the commands that will initialise GDB, load your executable and then run it. To make it easier to cut and paste, the 'Initialize' commands are:

```
target remote | m68k-elf-sprite pe://USBMultilink m5282evb
load
```

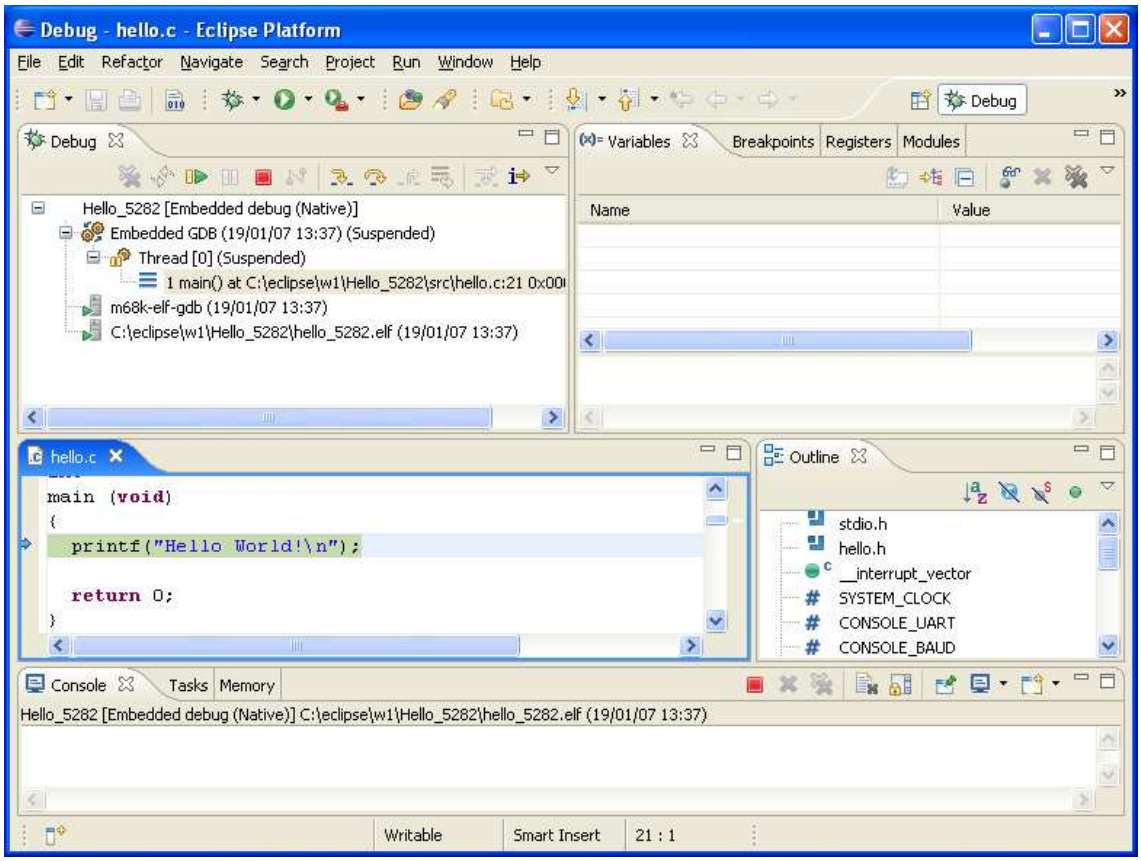
The 'Run' commands are:

```
set $pc=_start
continue
```

If you have a different target board (different CPU) then you may need to use a different configuration script. If your BDM cable is not a USB cable then the interface description will be different. See the CodeSourcery 'Getting Started' manual for details.

You can add all sorts of additional commands if you like. By default, the script will load the executable and stop execution when it gets to main(). You could add the command 'break hardware_init_hook' before the 'continue' command and the debugger will stop when the hardware_init_hook() function is called by the startup code.

If you click on the 'Debug' button, the configuration will be saved and the debugger will be launched. You may be asked if you want to switch to the 'Debug perspective', if so say yes.



The executable will be loaded and execution will halt at the first line in the main() function.

If you click on the green and yellow 'Resume' button on the 'Debug' tab, the program will execute and 'Hello World!' will be printed from the console port. The example as written uses UART0 and a baud rate of 19200.

The icing on the cake

Most embedded designers use a serial console to run and debug their software – how great would it be if the serial console was part of the IDE?

With Eclipse there are hundreds of tools you can download and buried in the Eclipse Device Software Development Platform project you will find, joy of joys, a serial terminal component! In fact it also does telnet and ssh.

To get this, go to the Target Management downloads page:

http://download.eclipse.org/dsdp/tm/downloads/index.php

You need a version of the Target Management RSE integration build that is later than 2.0M4. At the time of writing this means using an 'Integration Build' rather than a 'Stable Build', version I20070118-0400. Click on the relevant build's page link and download the TM-Terminal component from the standalone offerings section.

You will also need the RXTX extension to your JVM so that Java can access your serial ports, which can be found at:

ftp://ftp.qbang.org/pub/rxtx/rxtx-2.1-7-bins-r2.zip

To install:

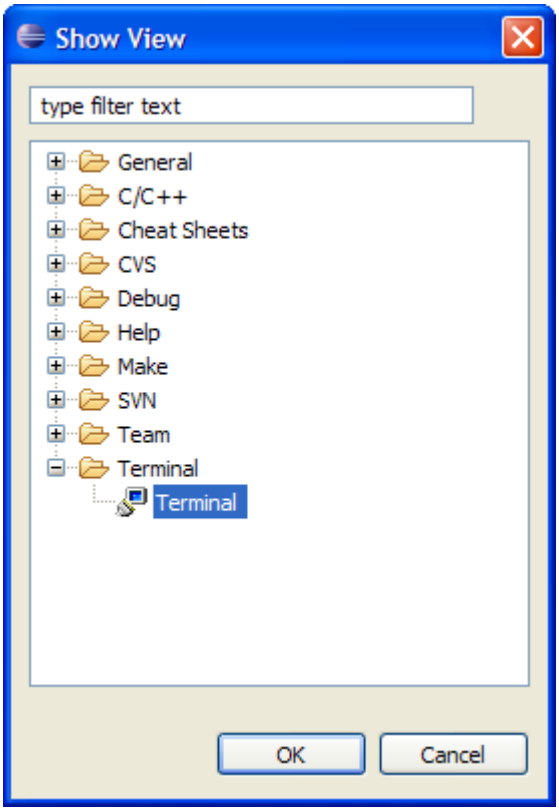
1. Copy RXTXcomm.jar from the rxtx zip file into $JRE/lib/ext.
2. Copy the rxtxSerial.dll file to $JRE/bin.

3. Extract the TM-Terminal zip package to your Eclipse installation folder.

$JRE is the folder that your Java Runtime Environment is installed in. Use the Java applet on the control panel to find this out. It is usually something like:

C:\Program Files\Java\jre1.5.0_10

You will need to restart Eclipse and then select Show View->Other... from the Window menu and you will see this dialog:



Select 'Terminal' and click OK and you will see a Terminal window appear at the bottom of the IDE. I'll leave it to you to figure out how to set the communication parameters and how to adjust the position terminal window.