PEEDI

Powerful Embedded Ethernet Debug Interface

Using Eclipse

Version 1.0

RONETIX
embedded development tools

September, 2012
Ronetix has made every attempt to ensure that the information in this document is accurate and complete. However, Ronetix assumes no responsibility for any errors, omissions, or for any consequences resulting from the use of the information included herein or the equipment it accompanies. Ronetix reserves the right to make changes in its products and specifications at any time without notice.
Any software described in this document is furnished under a license or non-disclosure agreement. It is against the law to copy this software on magnetic tape, disk, or other medium for any purpose other than the licensee's personal use.

Ronetix Development Tools GmbH
Waidhausenstrasse 13/5
1140 Vienna
Austria
Tel: +43-720-500315
+43-1 956 3138
Fax: +43-1- 8174 955 3464
Internet: www.ronetix.at
E-Mail info@ronetix.at

Acknowledgments:

ARM, ARM7, ARM9, ARM11, Cortex-M3, Cortex-A8 and Thumb are trademarks of ARM Ltd.
PowerPC and ColdFire are trademarks of Freescale Ltd.
Blackfin is trademark of Analog Devices Ltd.
Windows, Win32, Windows CE are trademarks of Microsoft Corporation.
Ethernet is a trademark of XEROX.
MIPS32 is a trademark of MIPS Technologies
AVR32 is a trademark of Atmel
All other trademarks are trademarks of their respective companies.

© 2005-2012 RONETIX GmbH
All rights reserved.

Change log

September 2012	- First release

1 Introduction

This document will show you how easy is to install and use eclipse to debug embedded applications using PEEDI as a JTAG interface.

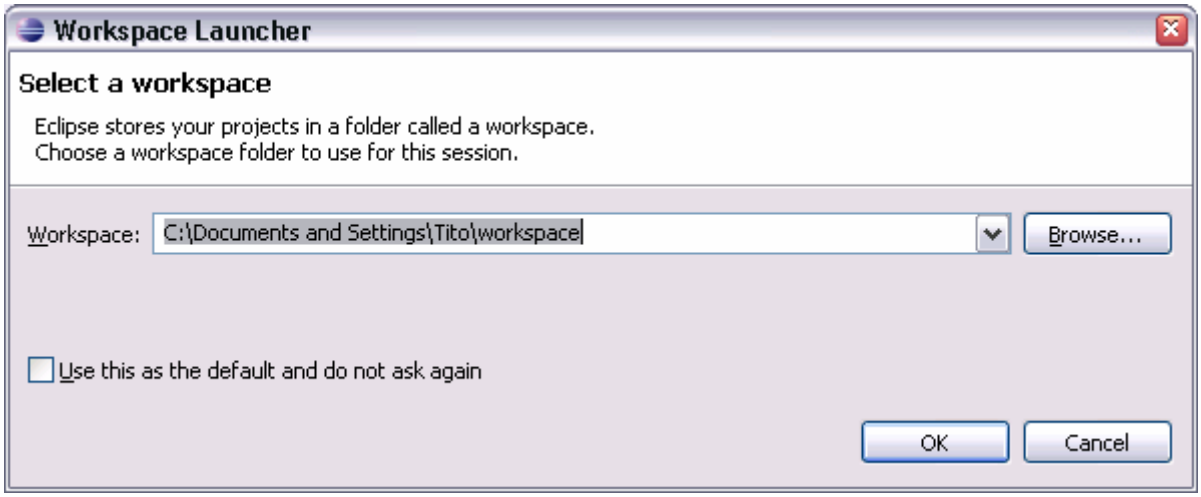
We will assume this:

- We already have an existing makefile based project we want to debug, so opening a console and typing **make** command should build our project producing and ELF file.
- We have gdb(arm-elf-gdb, powerpc-eabi-gdb or any suitable) installed, usually it is included in the GNU GCC toolchain used to build the application.
- We have a PEEDI that is configured and successfully connects to the test board.

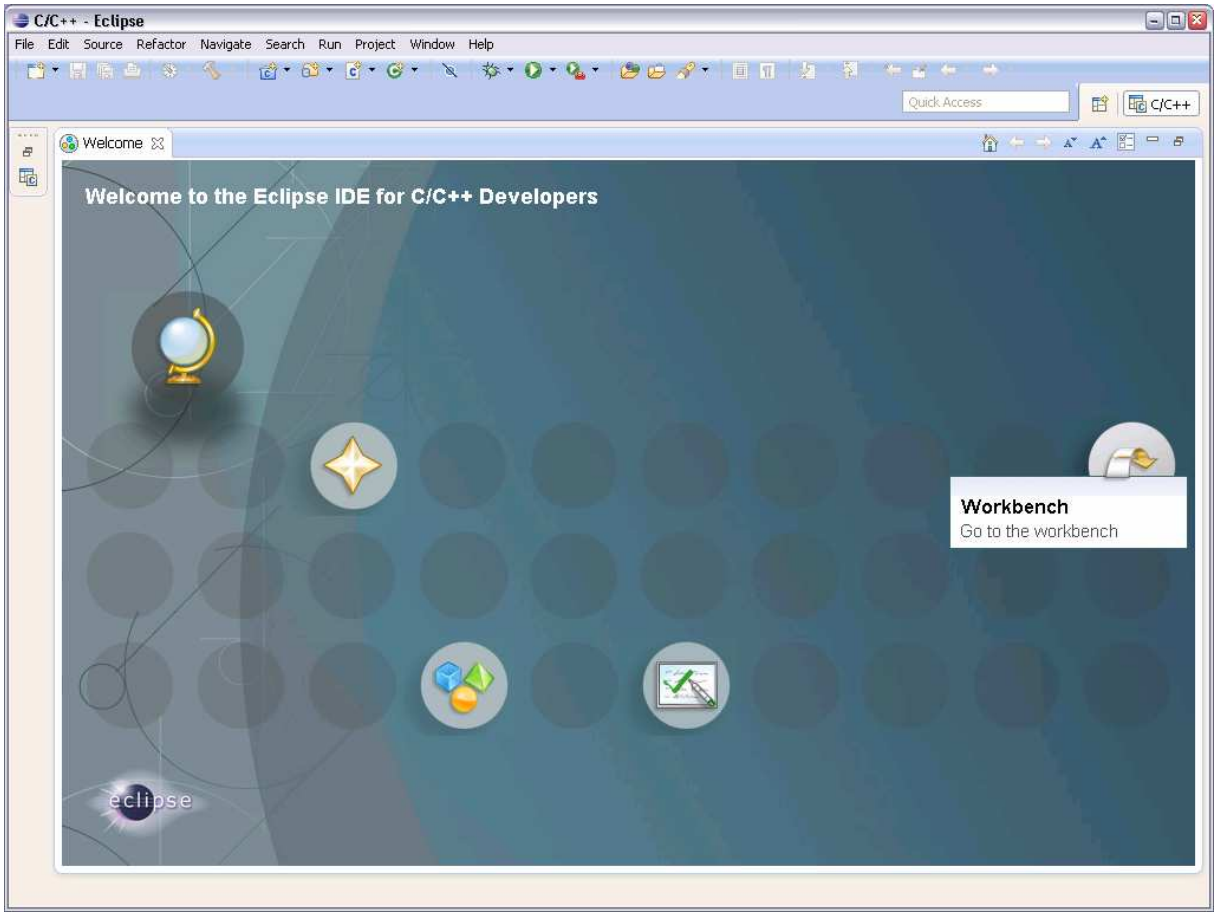
2 Installing Eclipse

2.1 Installing Eclipse

First go to the Eclipse website http://www.eclipse.org/downloads/ and download **Eclipse IDE for C/C++ Developers**. Extract the contents of the archive somewhere in your PC and start eclipse. When eclipse opens it will ask you for a workspace directory, choose one or leave the default:



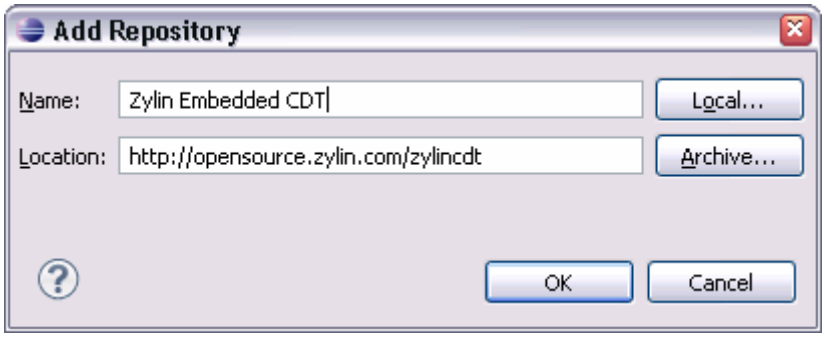
Next the eclipse welcome screen is shown:



Click the **Go to the workbench** button to close it.

2.2 Installing the Embedded debug plugin

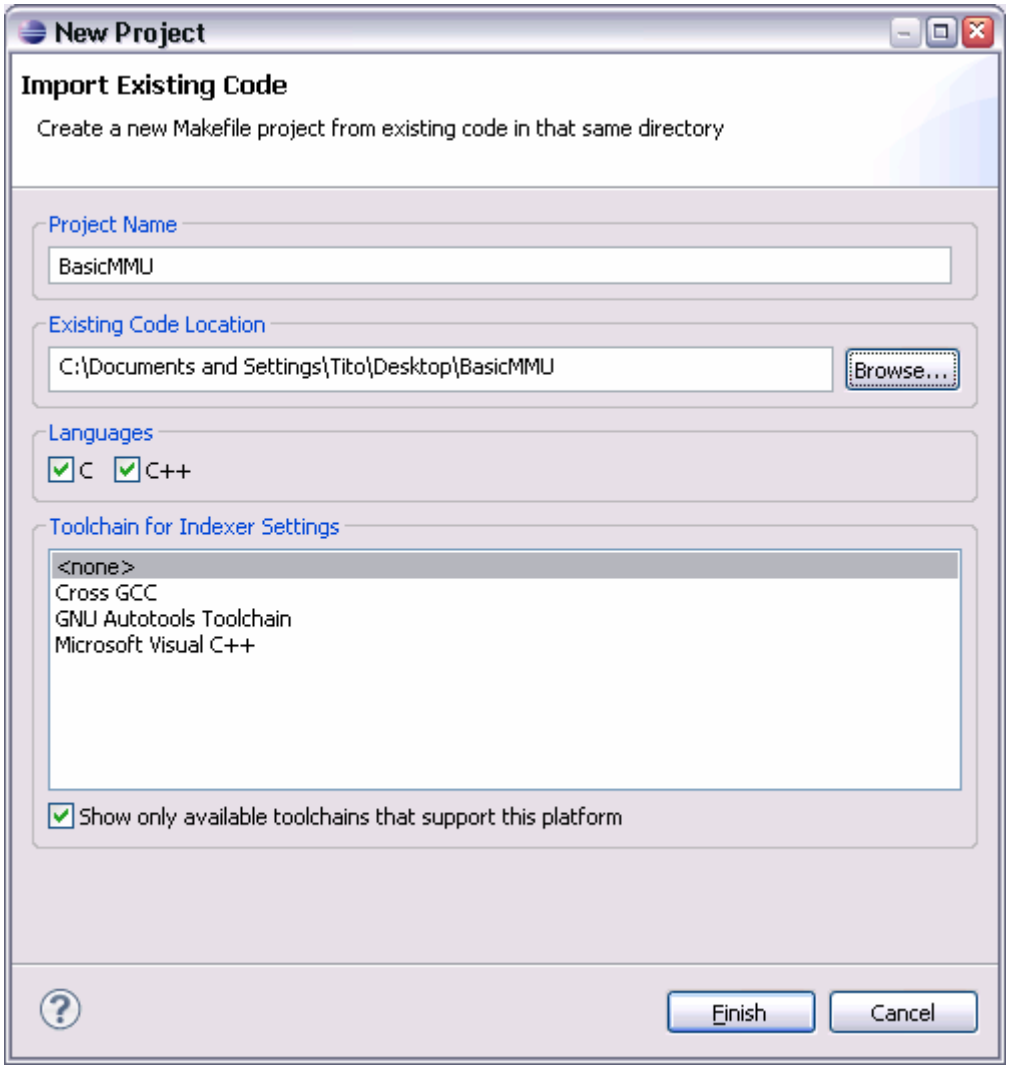
Now go to **Help->Install New Software**, then click **Add** and fill the dialog as shown bellow:



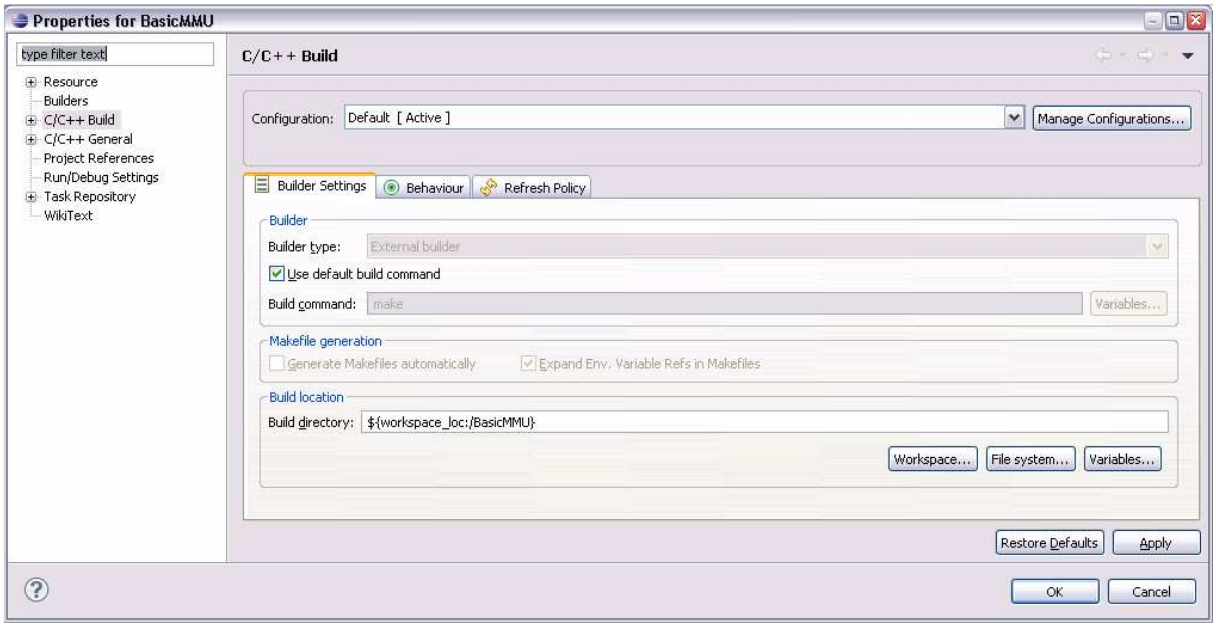
Then check the **Zylin Embedded CDT** box, click **Next** and follow the instructions. Eventually you will be asked to agree restarting eclipse – accept this.

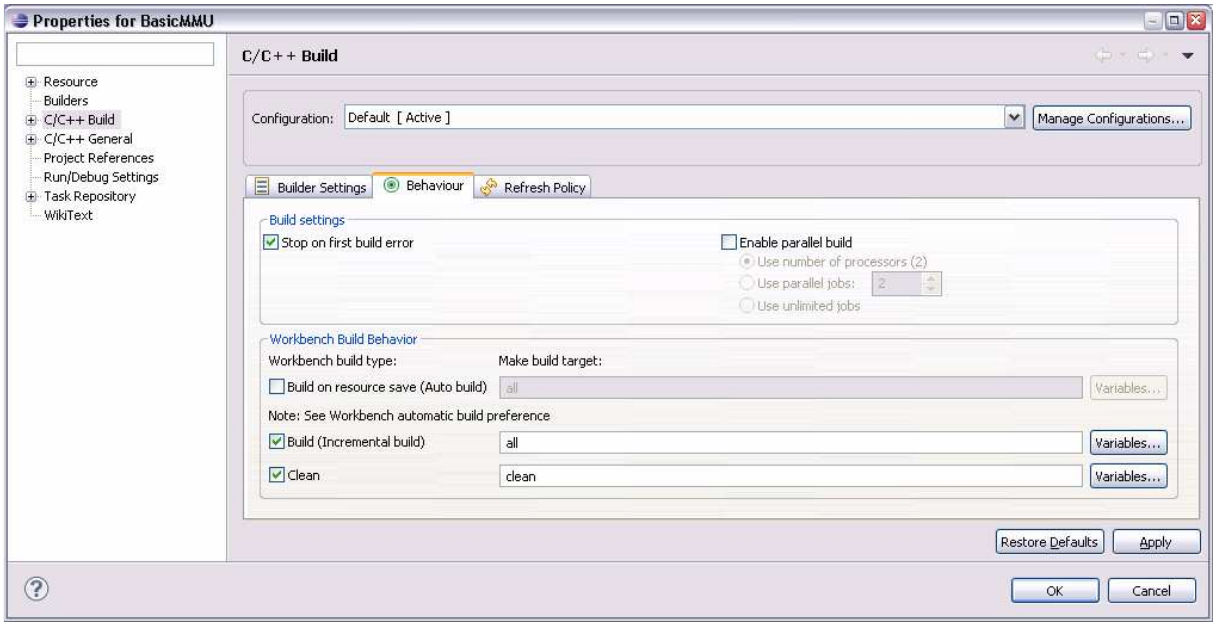
3 Importing your project

Now click **File->New->Makefile Project with Existing Code** and the **New Project** dialog will open, where you have to navigate to your project, clicking the **Browse** button. This will fill the project name automatically – change it or leave, it if you like the suggested name and click **Finish**:

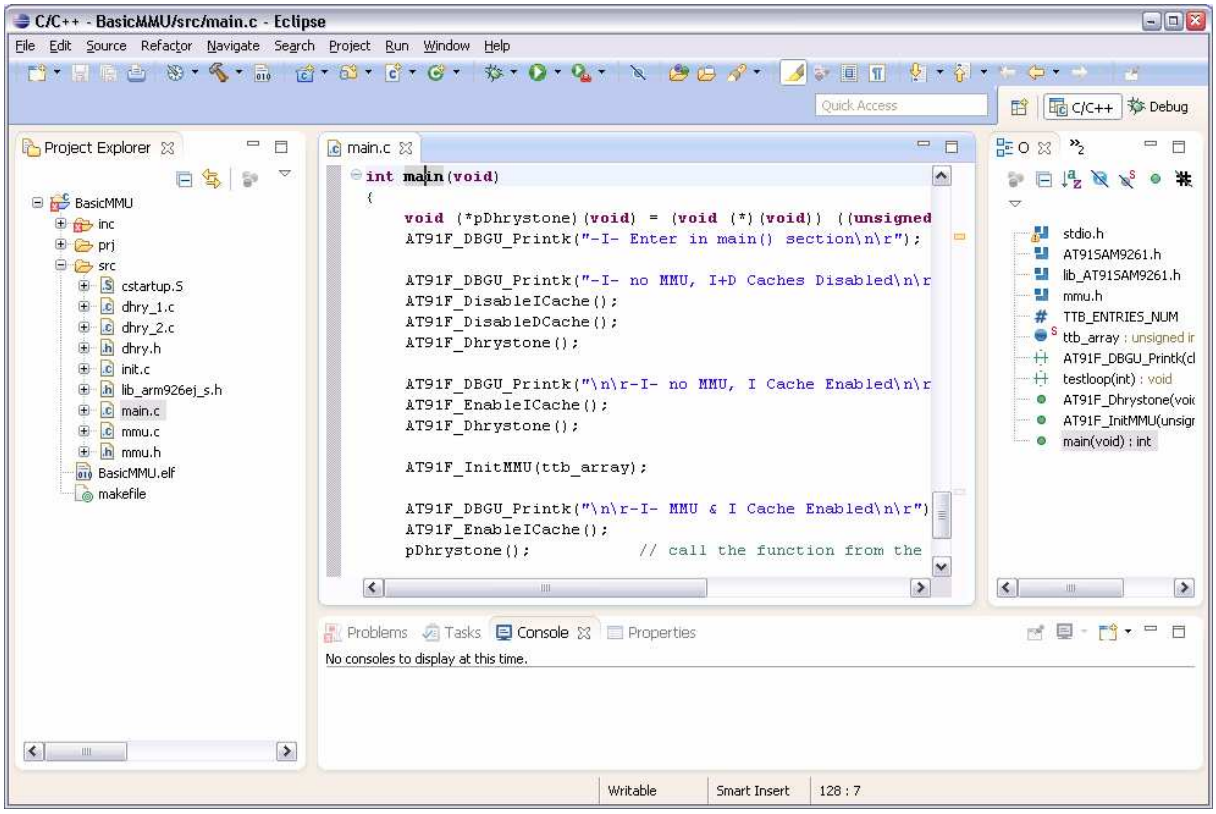


Now your project is imported and if you click the “Hammer” toolbar button it should build successfully. By default eclipse will execute the **make all** command to build your project. If it requires another command, for example **make debug**, you can set this right clicking on your project and select **Properties** and then **C/C++ Build**. Here you can override the very **make** command, or in the **Behavior** tab, you can change only the make sub-command for example from **all** to **debug:**


Properties for BasicMMU
type filter text
Resource
Builders
C/C++ Build
C/C++ General
Project References
Run/Debug Settings
Task Repository
WikiText
C/C++ Build
Configuration: Default [Active]
Manage Configurations...
Builder Settings
Behaviour
Refresh Policy
Builder
Builder type: External builder
Use default build command
Build command: make
Variables...
Makefile generation
Generate Makefiles automatically
Expand Env. Variable Refs in Makefiles
Build location
Build directory: ${workspace_loc:/BasicMMU}
Workspace...
File system...
Variables...
Restore Defaults
Apply
OK
Cancel

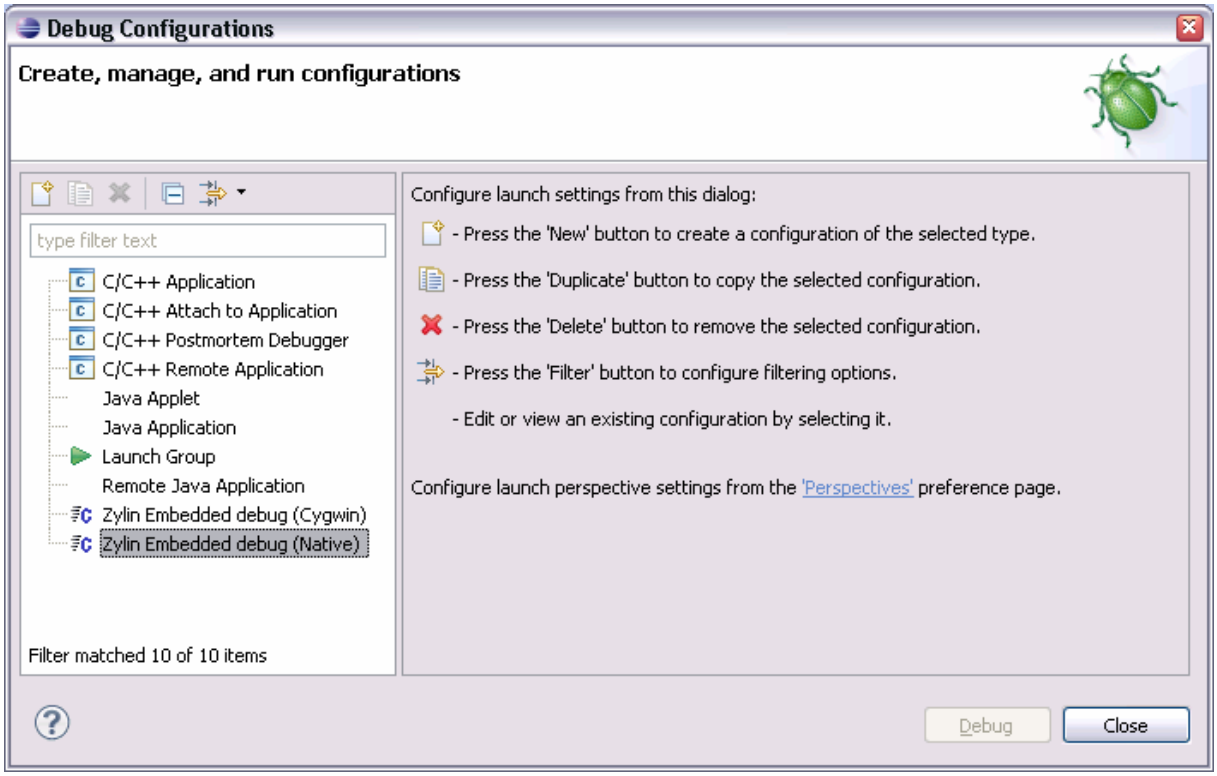

Properties for BasicMMU
Resource
Builders
C/C++ Build
C/C++ General
Project References
Run/Debug Settings
Task Repository
WikiText
C/C++ Build
Configuration: Default [Active]
Manage Configurations...
Builder Settings
Behaviour
Refresh Policy
Build settings
Stop on first build error
Enable parallel build
Use number of processors (2)
Use parallel jobs: 2
Use unlimited jobs
Workbench Build Behavior
Workbench build type:
Make build target:
Build on resource save (Auto build)
all
Variables...
Note: See Workbench automatic build preference
Build (Incremental build)
all
Variables...
Clean
clean
Variables...
Restore Defaults
Apply
OK
Cancel

Finally the C/C++ perspective should look like this:

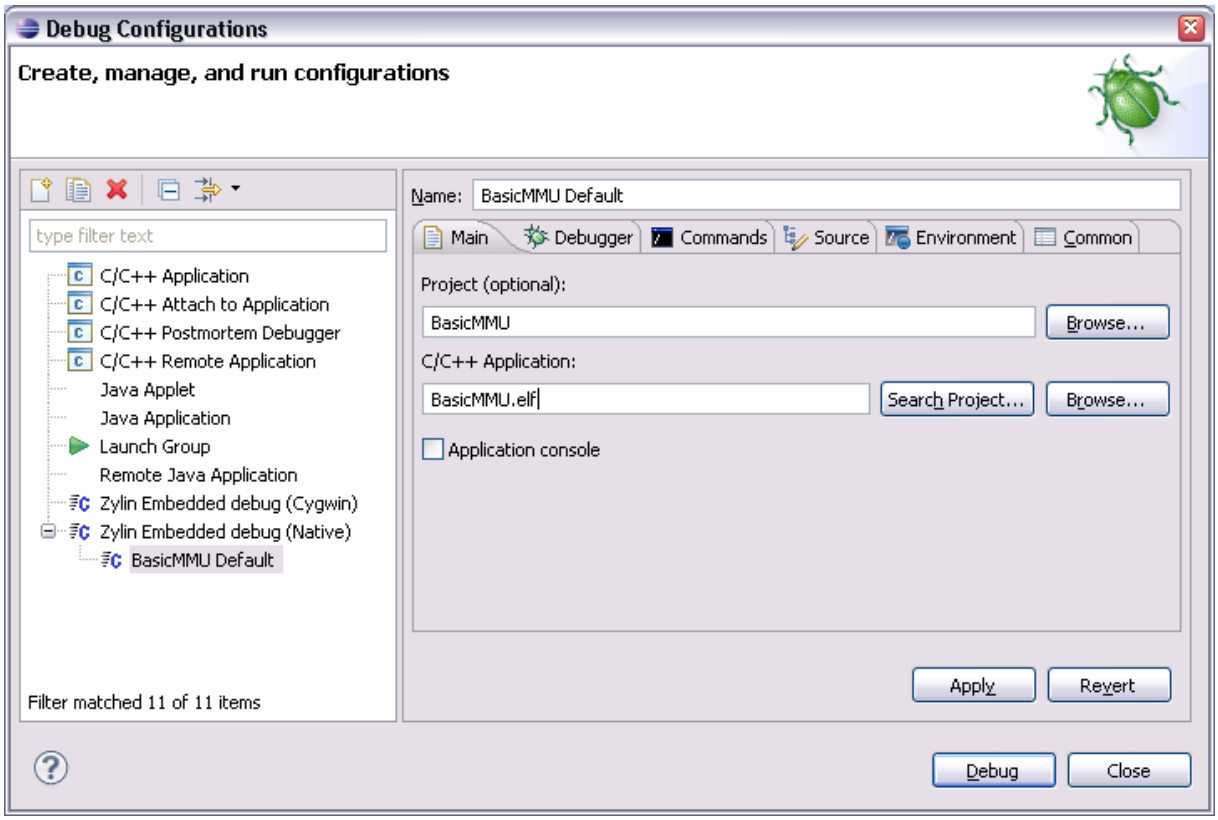


4 Setting the debug configuration

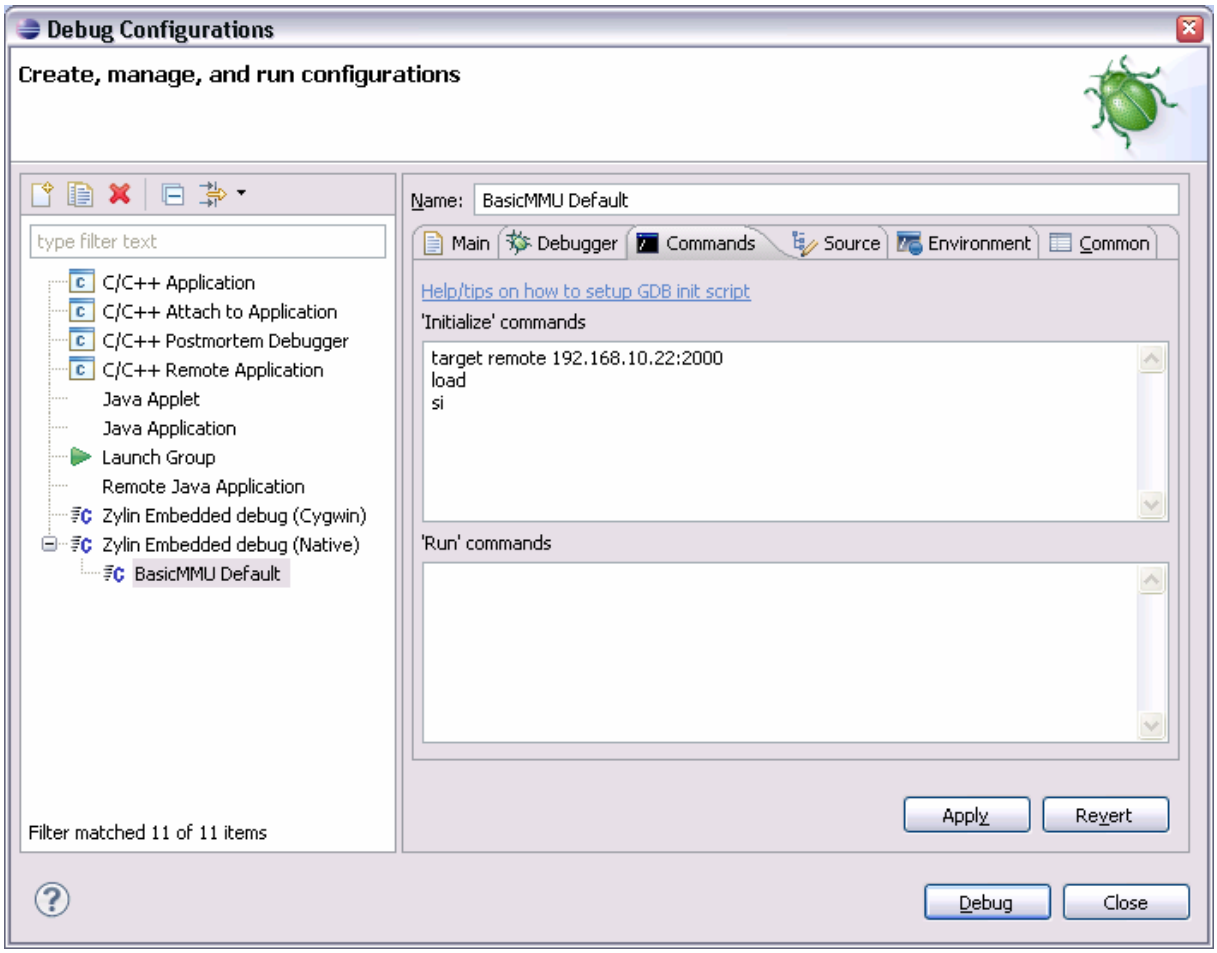
Now click on your project on the left pane just to select it and then click the down arrow on the right of the “Bug” toolbar button and select **Debug configurations**, this opens the following dialog:



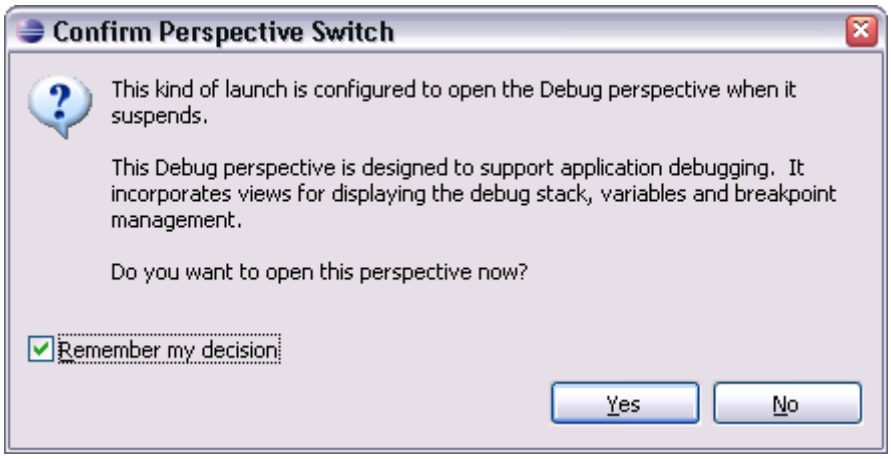
Double clicking on the **Zylin Embedded debug (Native)** will create a new debug configuration, named on your project, here you will need to enter or select the project ELF file:



Next open the debugger tab and make sure the right gdb is selected as **GDB debugger**, for ARM targets it should be arm-xxx-gdb, for example arm-elf-gdb, arm-none-gnueabi-gdb or just the gdb that comes with your GCC toolchain. If the selected debugger is not found because its path is not set correctly, you can use the **Browse** button to exactly point to it and thus exclude any path problems. Next open the **Commands** tab and fill it like this considering your PEEDI IP:



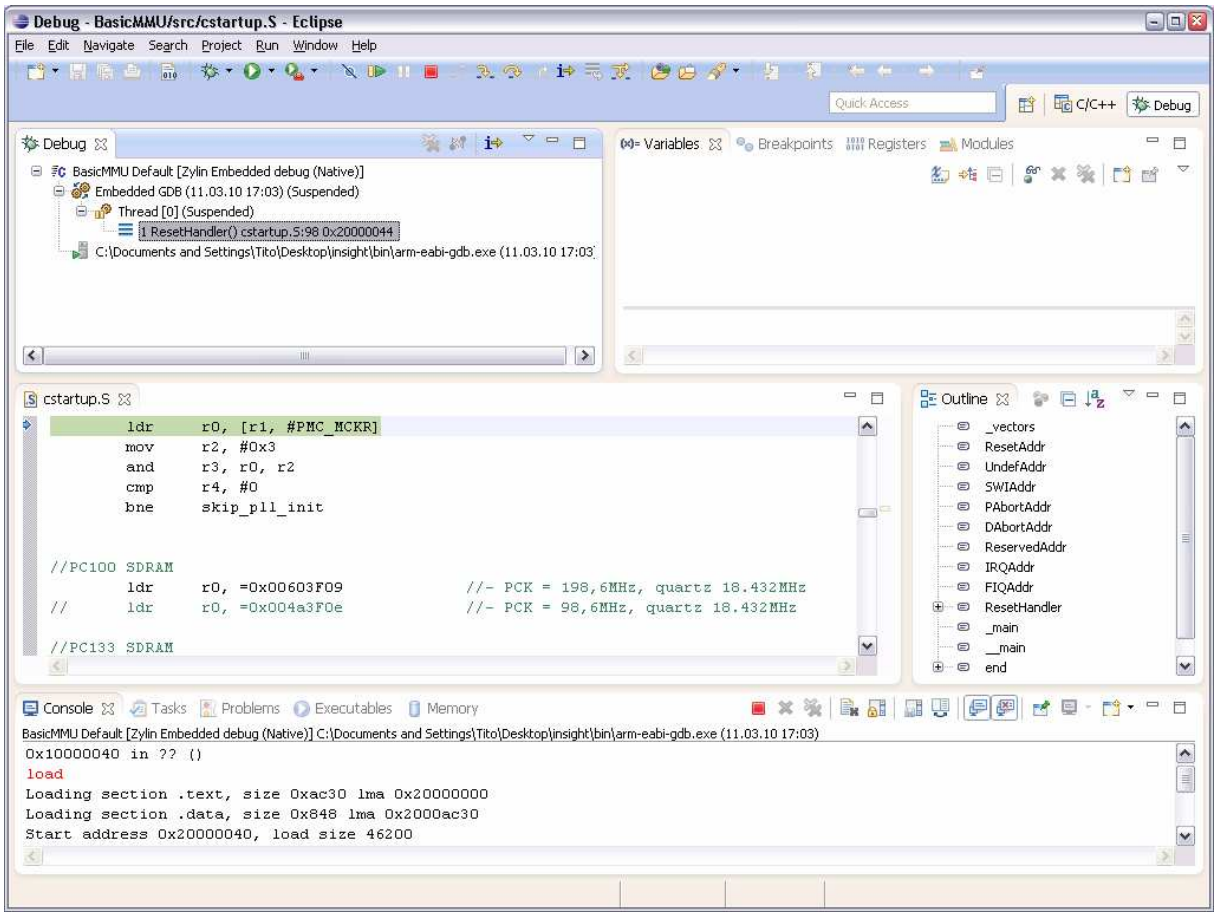
The **target remote** command tells to gdb to connect to PEEDI using the provided IP address. The **load** command loads the code to be debugged to the target (in this case it loads it to the RAM). The **si** command tells the gdb to make a single step just to refresh the eclipse debug view. After finished click **Debug**, this will start a debug session and eclipse will ask you to witch to debug perspective, click **Remember my decision** and then Yes:



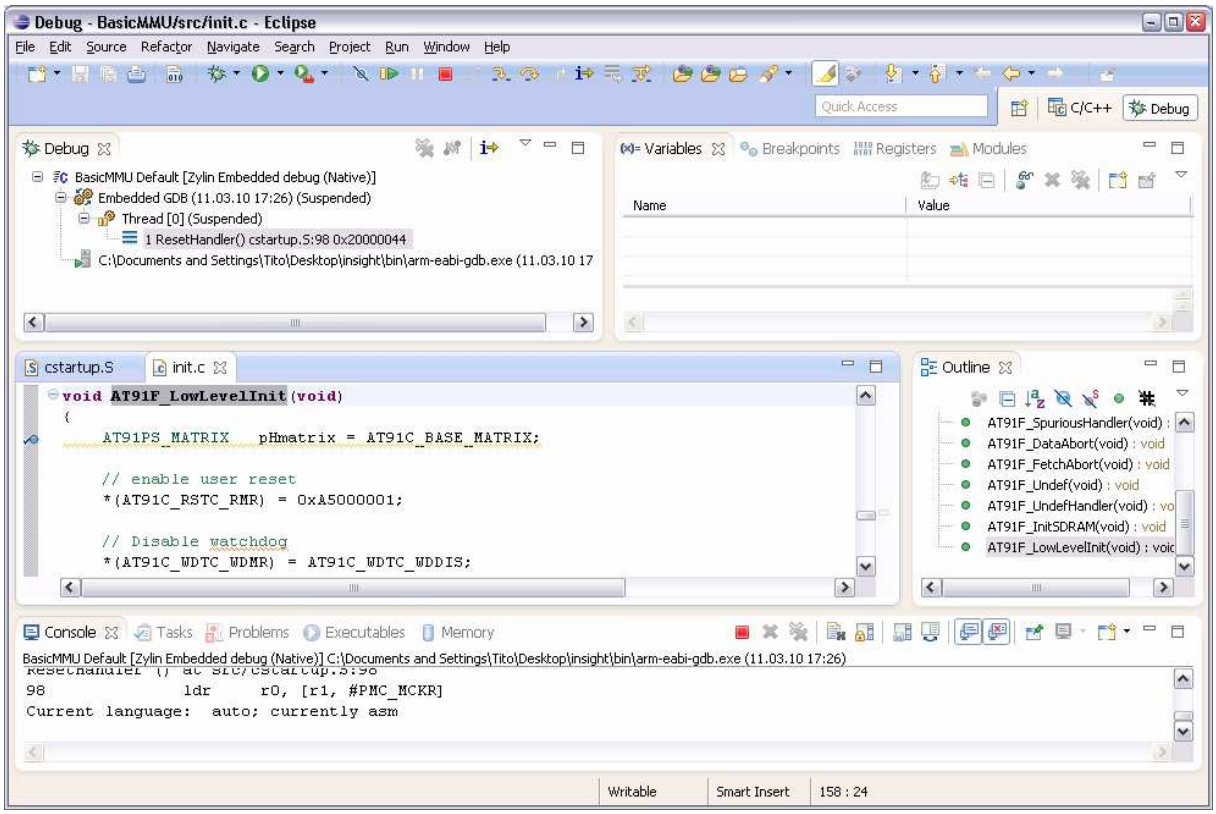
This will open the debug perspective.

5 Debugging

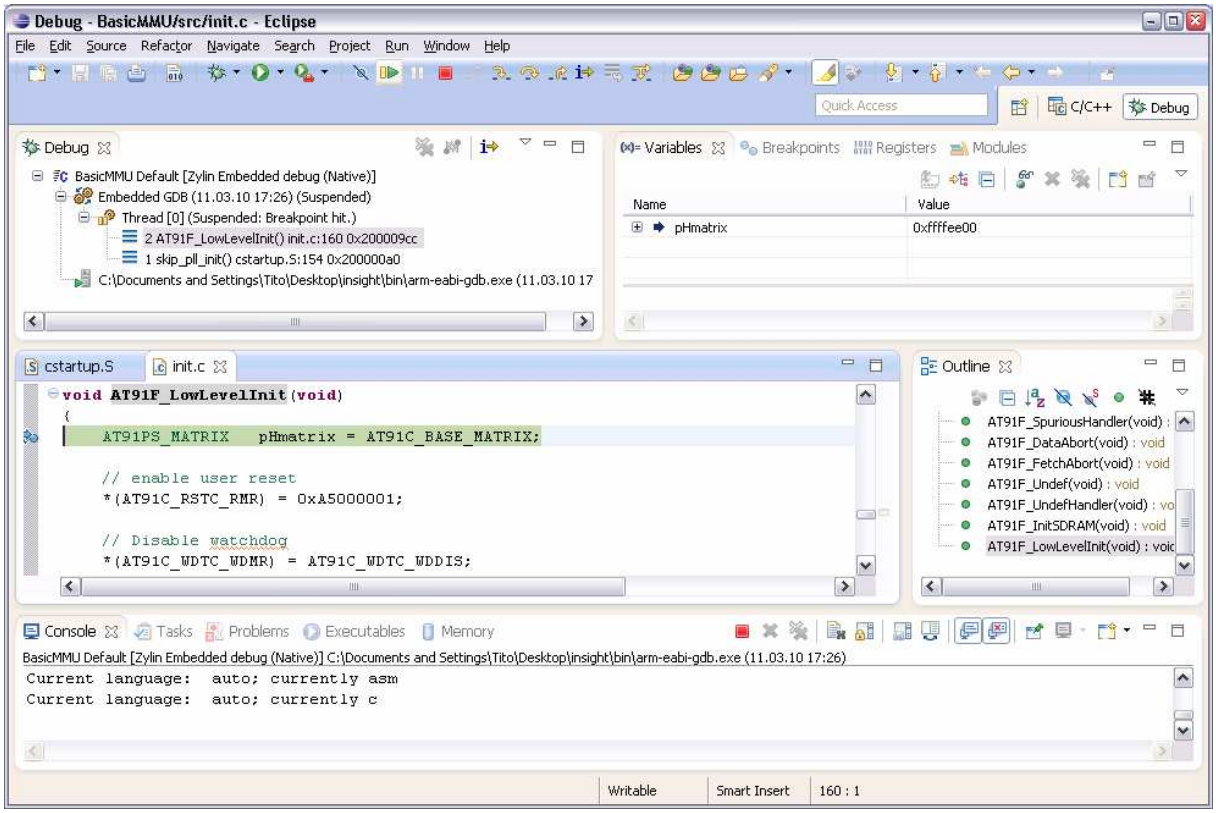
Here is the debug perspective opened.



Now you can continue debugging normally, i.e. do some single steps, put breaks or hit the resume button to start the executions. You can insert a break clocking on the beginning of the desired line in the source file, for example I will click on the C/C++ perspective just to see the source files, so I can open init.c, then I will switch back to the debug perspective and put a break on the first line of AT91F_LowLevelInit() function:



Hitting the **Resume** toolbar button or F8 will start the target and a moment later hit the break I set:



If you want to restart the debug just right click on the debug session **BasicMMU Defgault** and select **Relaunch** or whatever action you need to take. To again start the debug session from the C/C++ perspective just clock the down arrow on the right of “Bug” toolbar button and select your project debug configuration.

6 Conclusion

As you can see Eclipse is a rich feature IDE and more and more companies use it as a development environment. It has all of the known IDE functionalities and much more, installing the right plugin from **Help->Marketplace**. For example the subclipse plugin allows you to synchronize your project with a SVN repository using the Team Synchronization perspective, which is much easier then using the command line svn client.