Development environment with GNU Tool Chain

32-BIT MICROCONTROLLER
FM3 Family Application note

APPLICATION NOTE

Publication Number MB9BFD18T-AN706-00061 Revision 1.1 Issue Date January 31, 2014

Table of Contents

Target products

This application note is described about below products;

(TYPE0)

Series	Product Number (not included Package suffix)
MB9A100A	MB9AF102NA,MB9AF104NA,MB9AF105NA MB9AF102RA,MB9AF104RA,MB9AF105RA
MB9B100A	MB9BF102NA,MB9BF104NA,MB9BF105NA,MB9BF106NA MB9BF102RA,MB9BF104RA,MB9BF105RA,MB9BF106RA
MB9B300B	MB9BF304NB,MB9BF305NB,MB9BF306NB MB9BF304RB,MB9BF305RB,MB9BF306RB
MB9B400A	MB9BF404NA,MB9BF405NA,MB9BF406NA MB9BF404RA,MB9BF405RA,MB9BF406RA
MB9B500B	MB9BF504NB,MB9BF505NB,MB9BF506NB MB9BF504RB,MB9BF505RB,MB9BF506RB

(TYPE1)

Series	Product Number (not included Package suffix)
MB9A110A	MB9AF111LA,MB9AF112LA,MB9AF114LA MB9AF111MA,MB9AF112MA,MB9AF114MA,MB9AF115MA,MB9AF116MA MB9AF111NA,MB9AF112NA,MB9AF114NA,MB9AF115NA,MB9AF116NA
MB9A310A	MB9AF311LA,MB9AF312LA,MB9AF314LA MB9AF311MA,MB9AF312MA,MB9AF314MA,MB9AF315MA,MB9AF316MA MB9AF311NA,MB9AF312NA,MB9AF314NA,MB9AF315NA,MB9AF316NA

(TYPE2)

Series	Product Number (not included Package suffix)
MB9B110T	MB9BF116S,MB9BF117S,MB9BF118S MB9BF116T,MB9BF117T,MB9BF118T
MB9B210T	MB9BF216S,MB9BF217S,MB9BF218S MB9BF216T,MB9BF217T,MB9BF218T
MB9B310T	MB9BF316S,MB9BF317S,MB9BF318S MB9BF316T,MB9BF317T,MB9BF318T
MB9B410T	MB9BF416S,MB9BF417S,MB9BF418S MB9BF416T,MB9BF417T,MB9BF418T
MB9B510T	MB9BF516S,MB9BF517S,MB9BF518S MB9BF516T,MB9BF517T,MB9BF518T
MB9B610T	MB9BF616S,MB9BF617S,MB9BF618S MB9BF616T,MB9BF617T,MB9BF618T
MB9BD10T	MB9BFD16S,MB9BFD17S,MB9BFD18S MB9BFD16T,MB9BFD17T,MB9BFD18T

(TYPE3)

Series	Product Number (not included Package suffix)
MB9A130LA	MB9AF131KA,MB9AF132KA MB9AF131LA,MB9AF132LA

(TYPE4)

Series	Product Number (not included Package suffix)
MB9B110R	MB9BF112N,MB9BF114N,MB9BF115N,MB9BF116N MB9BF112R,MB9BF114R,MB9BF115R,MB9BF116R
MB9B310R	MB9BF312N,MB9BF314N,MB9BF315N,MB9BF316N MB9BF312R,MB9BF314R,MB9BF315R,MB9BF316R
MB9B410R	MB9BF412N,MB9BF414N,MB9BF415N,MB9BF416N MB9BF412R,MB9BF414R,MB9BF415R,MB9BF416R
MB9B510R	MB9BF512N,MB9BF514N,MB9BF515N,MB9BF516N MB9BF512R,MB9BF514R,MB9BF515R,MB9BF516R

(TYPE5)

Series	Product Number (not included Package suffix)
MB9A110K	MB9AF111K,MB9AF112K
MB9A310K	MB9AF311K,MB9AF312K

(TYPE6)

Series	Product Number (not included Package suffix)
MB9A140NA	MB9AF141LA,MB9AF142LA,MB9AF144LA MB9AF141MA,MB9AF142MA,MB9AF144MA MB9AF141NA,MB9AF142NA,MB9AF144NA
MB9A340NA	MB9AF341LA,MB9AF342LA,MB9AF344LA MB9AF341MA,MB9AF342MA,MB9AF344MA MB9AF341NA,MB9AF342NA,MB9AF344NA
MB9AA40NA	MB9AFA41LA,MB9AFA42LA,MB9AFA44LA MB9AFA41MA,MB9AFA42MA,MB9AFA44MA MB9AFA41NA,MB9AFA42NA,MB9AFA44NA
MB9AB40NA	MB9AFB41LA,MB9AFB42LA,MB9AFB44LA MB9AFB41MA,MB9AFB42MA,MB9AFB44MA MB9AFB41NA,MB9AFB42NA,MB9AFB44NA

(TYPE7)

Series	Product Number (not included Package suffix)
MB9A130N	MB9AF131M,MB9AF132M MB9AF131N,MB9AF132N
MB9AA30N	MB9AFA31L,MB9AFA32L MB9AFA31M,MB9AFA32M MB9AFA31N,MB9AFA32N

(TYPE8)

Series	Product Number (not included Package suffix)
MB9A150R	MB9AF154M,MB9AF155M,MB9AF156M MB9AF154N,MB9AF155N,MB9AF156N MB9AF154R,MB9AF155R,MB9AF156R

(TYPE9)

Series	Product Number (not included Package suffix)
MB9B120M	MB9BF121K,MB9BF122K,MB9BF124K MB9BF121L,MB9BF122L,MB9BF124L MB9BF121M,MB9BF122M,MB9BF124M
MB9B320M	MB9BF321K,MB9BF322K,MB9BF324K MB9BF321L,MB9BF322L,MB9BF324L MB9BF321M,MB9BF322M,MB9BF324M
MB9B520M	MB9BF521K,MB9BF522K,MB9BF524K MB9BF521L,MB9BF522L,MB9BF524L MB9BF521M,MB9BF522M,MB9BF524M

1 Introduction

1.1 Description

This documentation describes the implementation of GNU tool chain on the Eclipse platform for the FM3 family. The hardware of a host and the target are following.
This documentation describes the method to use J-Link or ARM-USB-TINY in ICE.

Host OS	Windows7(32bit)
ICE	J-Link / ARM-USB-TINY
Target board	SK-FM3-176PMC-ETHERNET V1.1
Target MCU	MB9BFD18T

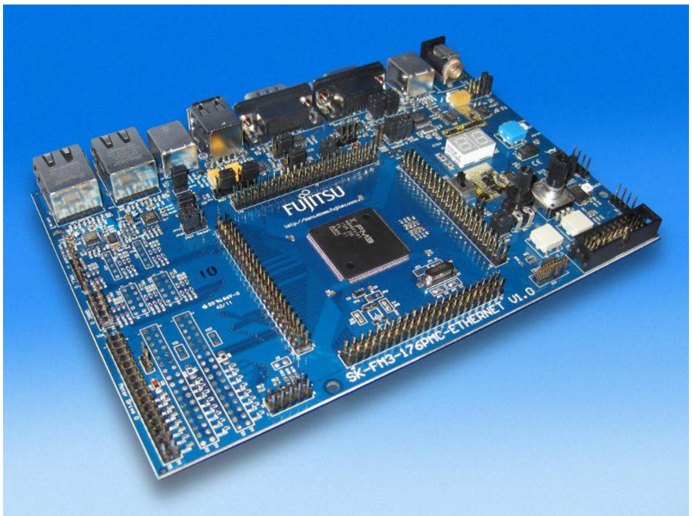


Figure 1 Spansion starterkit SK-FM3-176PMC-ETHERNET

The following programs are used to implement development environment in this documentation.

Compiler	YAGARTO
Driver	LibUSB
Debugger	OpenOCD
IDE	Eclipse + C/C++ development tooling(CDT)
Other	Java Runtime Environment

1.2 JTAG Interface

For flashing and debugging software on the MCU, the JTAG port of the board is used, and thus a JTAG interface is also needed.

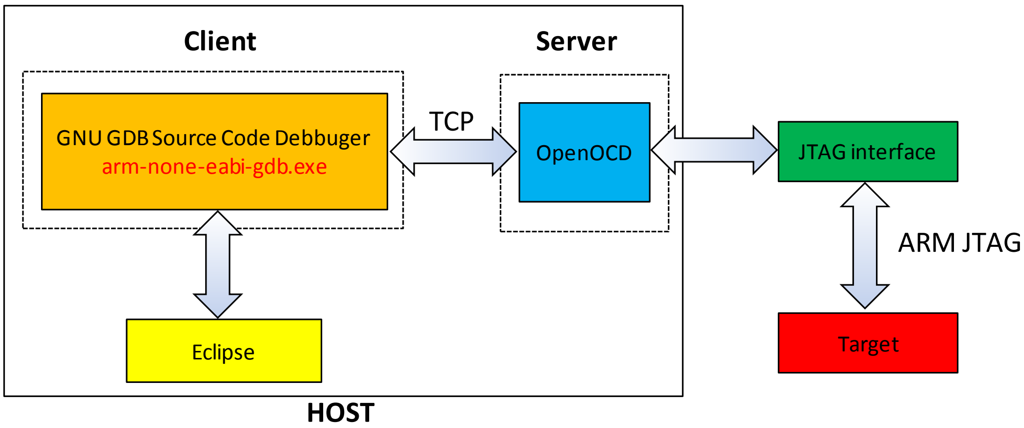


Figure 2 Relations of the host and the target with JTAG interface

1.3 J-Link

JTAG interface is the “J-Link”. This interface is product of the company IAR Systems.

Figure 3 J-Link from IAR Systems

The IAR Systems “J-Link” has the following features. For more information about the J-Link: http://www.iar.com/Global/Products/Hardware-Debug-probes/DS-J-Link-ARM-09.pdf

- USB powered JTAG emulator for Cortex-M devices
- License for J-Link GDB server
- Support download in RAM and Flash
- License for the flash breakpoints
- SWD / SWV
- Voltage range: 1.2V-5V

1.4 ARM-USB-TINY

Another JTAG interface is the “ARM-USB-TINY”. This interface is product of the company olimex.

Figure 4 ARM-USB-TINY from olimex

The olimex “ARM-USB-TINY” has the following features. For more information about the ARM-USB-TINY: https://www.olimex.com/Products/ARM/JTAG/ARM-USB-TINY/

- Debug all ARM microcontrollers supported by OpenOCD
- Fast speed USB 2.0 JTAG dongle interface
- Uses ARM’s standard 2*10 pin JTAG connector
- Voltage range: 2V-5V
- Software supported by OpenOCD

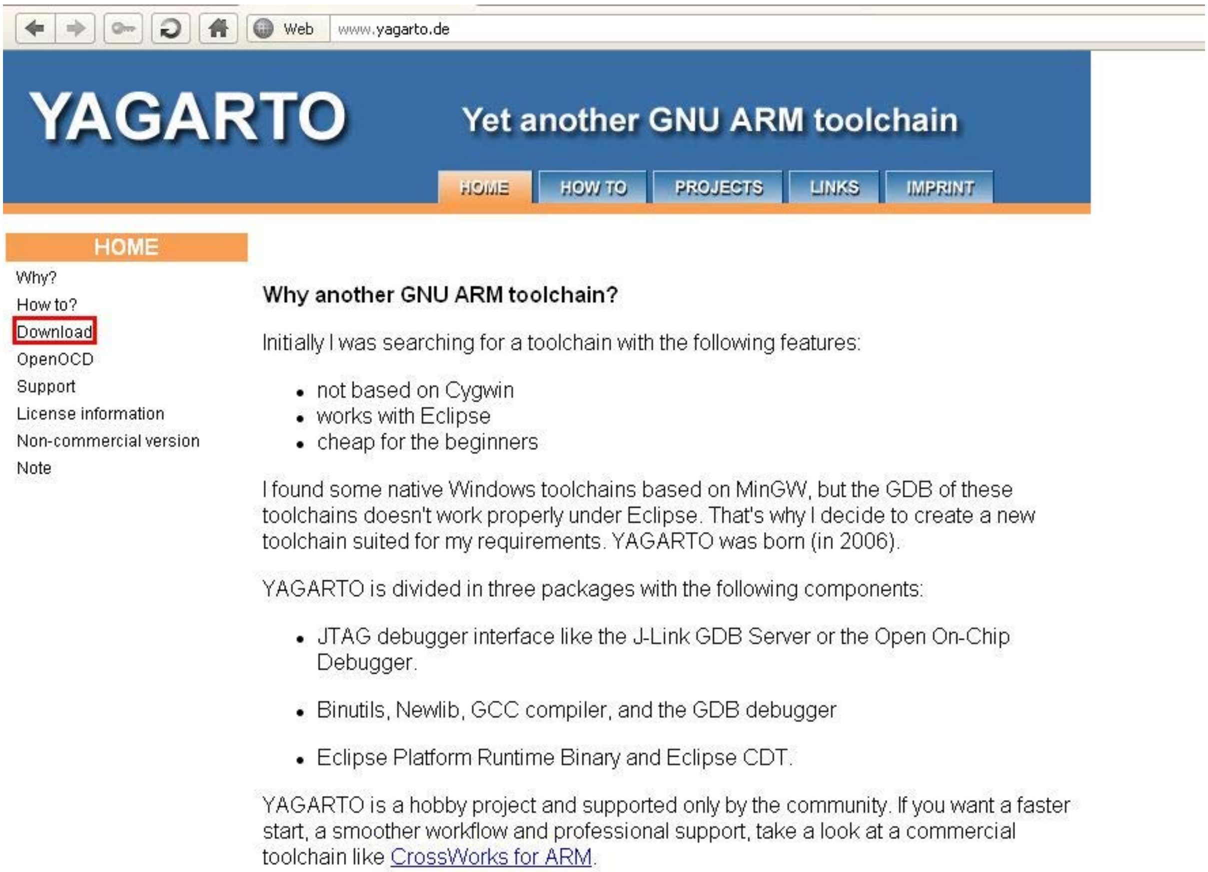
2 Compiler

2.1 Yet another GNU ARM Tool Chain (YAGARTO)

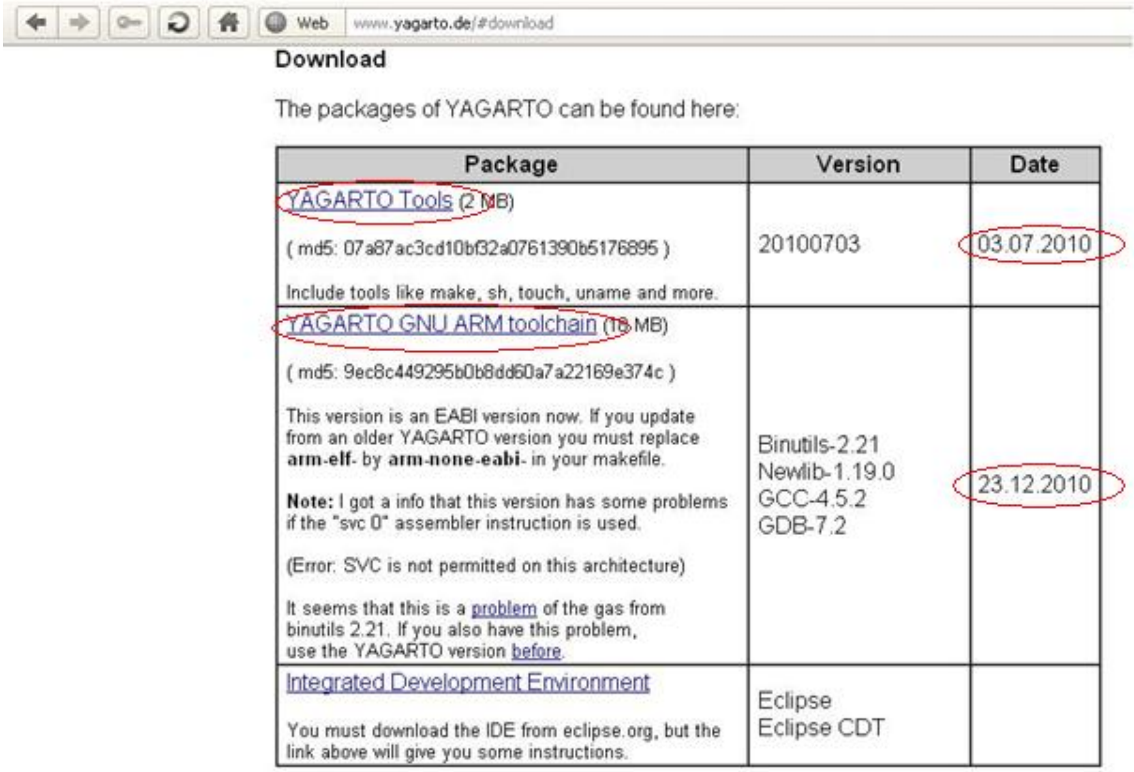
There are a number of pre-built GNU ARM compiler toolsets available on the web. This application note uses the YAGARTO pre-built ARM compiler tool suite developed by Michael Fischer. This version of the GNU compiler toolset for ARM has been natively compiled for the Intel/Windows platform.
Except the ARM compiler toolset the Yagarto project provides also other tools needed to build a make file project on Eclipse CDT e.g. make utility.

2.2 Downloading Yagarto Tools

The Yagarto components can be downloading from the Yagarto Website:
http://www.yagarto.de/



Use the “Download” link on the left menu pane.



www.yagarto.de/#download

Download

The packages of YAGARTO can be found here:

Package	Version	Date
YAGARTO Tools (2 MB) (md5: 07a87ac3cd10bf32a0761390b5176895) Include tools like make, sh, touch, uname and more.	20100703	03.07.2010
YAGARTO GNU ARM toolchain (18 MB) (md5: 9ec8c449295b0b8dd60a7a22169e374c) This version is an EABI version now. If you update from an older YAGARTO version you must replace **arm-elf-** by **arm-none-eabi-** in your makefile. **Note:** I got a info that this version has some problems if the "svc 0" assembler instruction is used. (Error: SVC is not permitted on this architecture) It seems that this is a problem of the gas from binutils 2.21. If you also have this problem, use the YAGARTO version before.	Binutils-2.21 Newlib-1.19.0 GCC-4.5.2 GDB-7.2	23.12.2010
Integrated Development Environment You must download the IDE from eclipse.org, but the link above will give you some instructions.	Eclipse Eclipse CDT	

It is recommended to use the latest versions provided on the website.

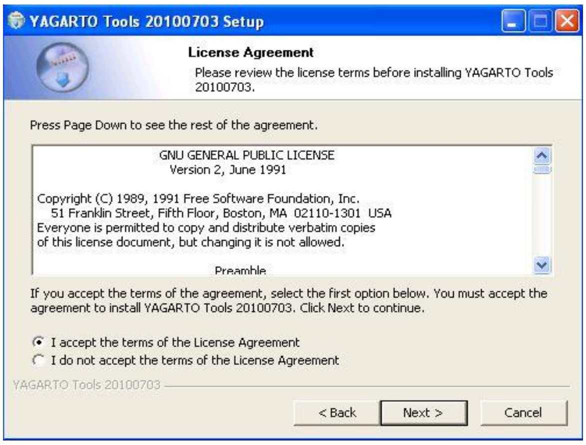
Only the first two packages are recommended at this moment, because the installation description of the third package "Eclipse IDE" and "Eclipse CDT" will be separately explained in detail in chapter 7.

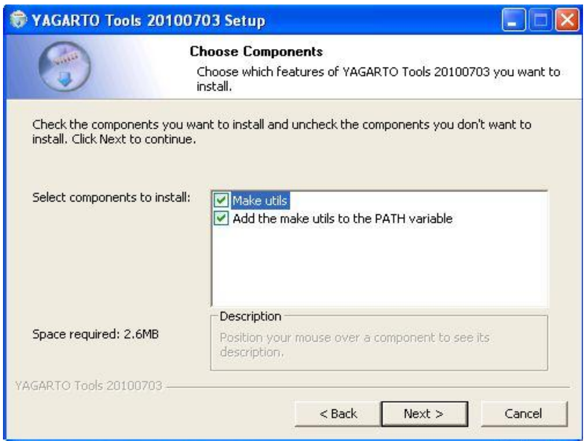
2.3 Installing YAGARTO tools

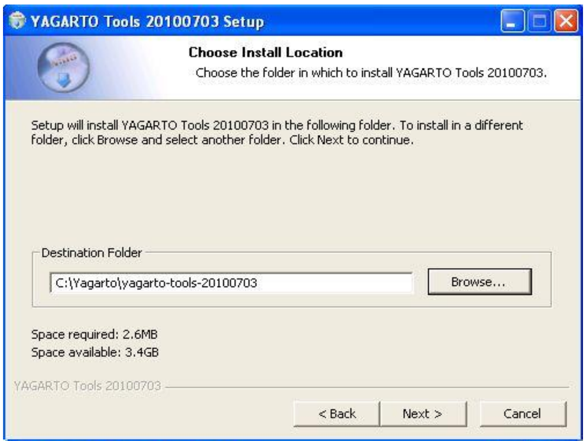
After saving the package, e.g. in the temporary folder "Yagarto-Downloads", the installation procedure of these tools can be started.

After downloading, start the installation of the make utility tools "*yagarto-tools-20100703-setup*" or newer.

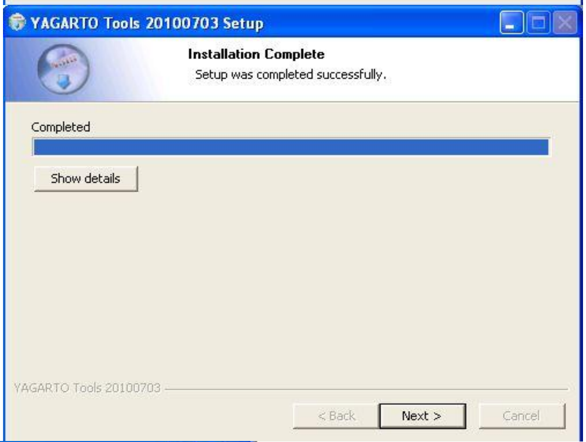

YAGARTO Tools 20100703 Setup
Welcome to the YAGARTO Tools 20100703 Setup Wizard
This wizard will guide you through the installation of YAGARTO Tools 20100703.
It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.
Click Next to continue.
Next >
Cancel


YAGARTO Tools 20100703 Setup
License Agreement
Please review the license terms before installing YAGARTO Tools 20100703.
Press Page Down to see the rest of the agreement.
GNU GENERAL PUBLIC LICENSE
Version 2, June 1991
Copyright (C) 1989, 1991 Free Software Foundation, Inc.
51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.
Preamble
If you accept the terms of the agreement, select the first option below. You must accept the agreement to install YAGARTO Tools 20100703. Click Next to continue.
I accept the terms of the License Agreement
I do not accept the terms of the License Agreement
YAGARTO Tools 20100703
< Back
Next >
Cancel


YAGARTO Tools 20100703 Setup
Choose Components
Choose which features of YAGARTO Tools 20100703 you want to install.
Check the components you want to install and uncheck the components you don't want to install. Click Next to continue.
Select components to install:
Make utils
Add the make utils to the PATH variable
Space required: 2.6MB
Description
Position your mouse over a component to see its description.
YAGARTO Tools 20100703
< Back
Next >
Cancel

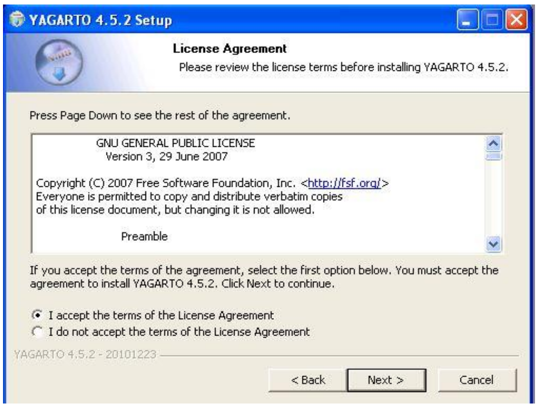

YAGARTO Tools 20100703 Setup
Choose Install Location
Choose the folder in which to install YAGARTO Tools 20100703.
Setup will install YAGARTO Tools 20100703 in the following folder. To install in a different folder, click Browse and select another folder. Click Next to continue.
Destination Folder
C:\Yagarto\yagarto-tools-20100703
Browse...
Space required: 2.6MB
Space available: 3.4GB
YAGARTO Tools 20100703
< Back
Next >
Cancel

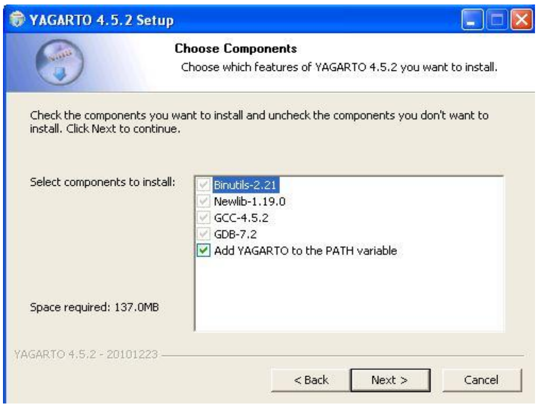

YAGARTO Tools 20100703 Setup
Choose Start Menu Folder
Choose a Start Menu folder for the YAGARTO Tools 20100703 shortcuts.
Select the Start Menu folder in which you would like to create the program's shortcuts. You can also enter a name to create a new folder.
YAGARTO Tools
Autostart
Avira
Cygwin
ElsterFormular
Microsoft Office
OpenOffice.org 3.3
Packard Bell ImageWriter
Panda Security
Sandboxie
Spiele
Verwaltung
Do not create shortcuts
YAGARTO Tools 20100703
< Back
Install
Cancel

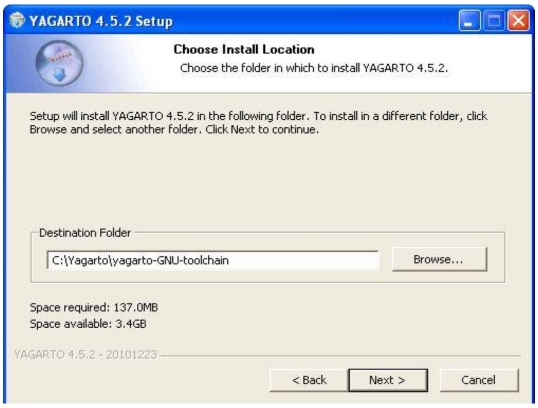

YAGARTO Tools 20100703 Setup
Installation Complete
Setup was completed successfully.
Completed
Show details
YAGARTO Tools 20100703
< Back
Next >
Cancel

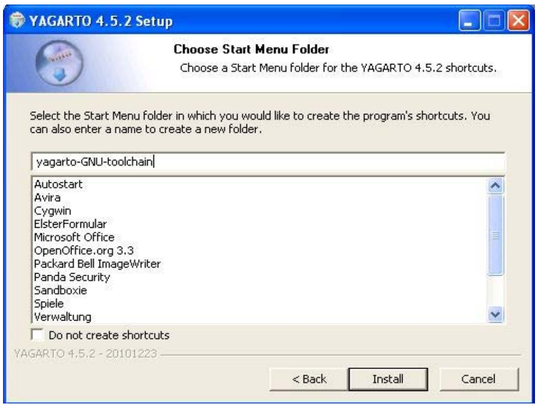

YAGARTO Tools 20100703 Setup
Completing the YAGARTO Tools 20100703 Setup Wizard
YAGARTO Tools 20100703 has been installed on your computer.
Click Finish to close this wizard.
Visit the YAGARTO site for the latest news.
< Back
Finish
Cancel

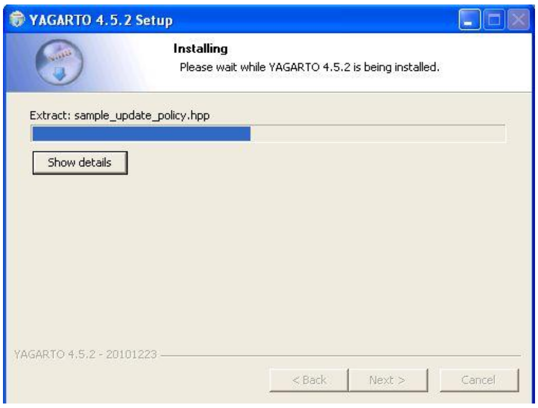
Next, start the following installation of the ARM compiler toolset "*yagarto-bu-2.21_gcc-4.5.2-c-c++_nl-1.19.0_gdb-7.2_eabi_20101223*" or newer.

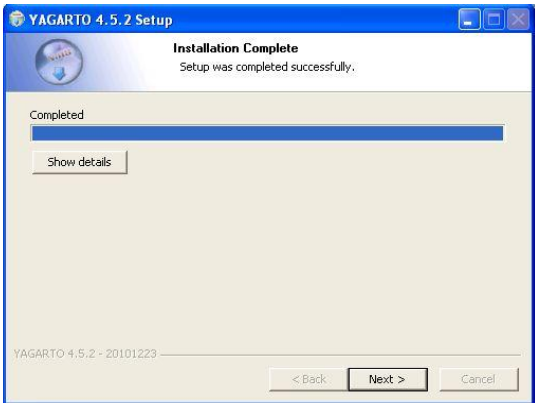












3 Driver

3.1 LibUSB

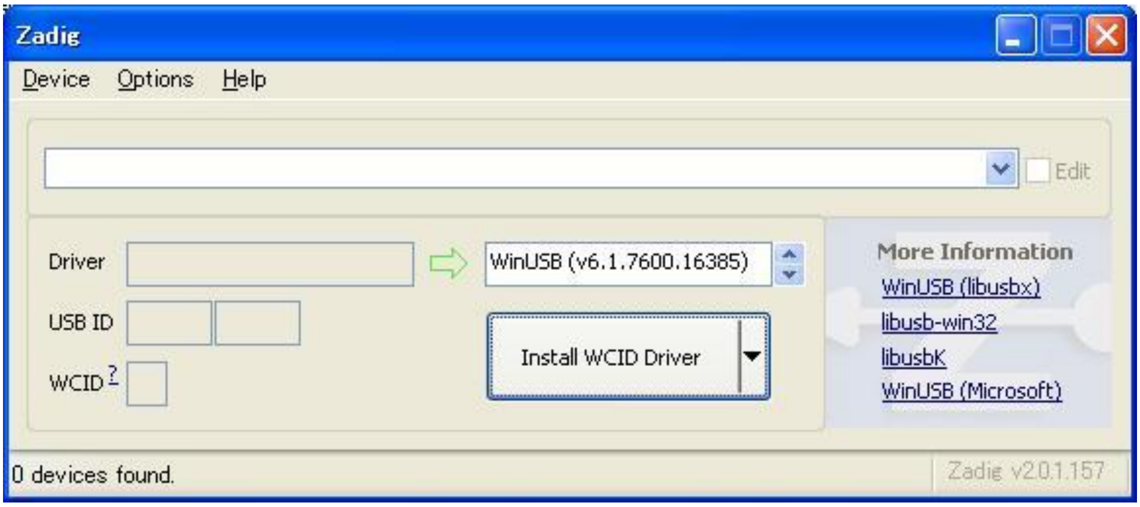
Note, this chapter describes the method which set a driver with J-Link.
But this method is common for ARM-USB-TINY.

J-Link must be set a driver to use OpenOCD. In this documentation, use "LibUSB" driver. Because ordinary J-Link driver doesn't support OpenOCD, it must be replaced in LibUSB. When replace it, using "Zadig" which is free tool (LGPL). Because LibUSB is included in Zadig beforehand, it doesn't need to download LibUSB in individual. Zadig can available from the following website.

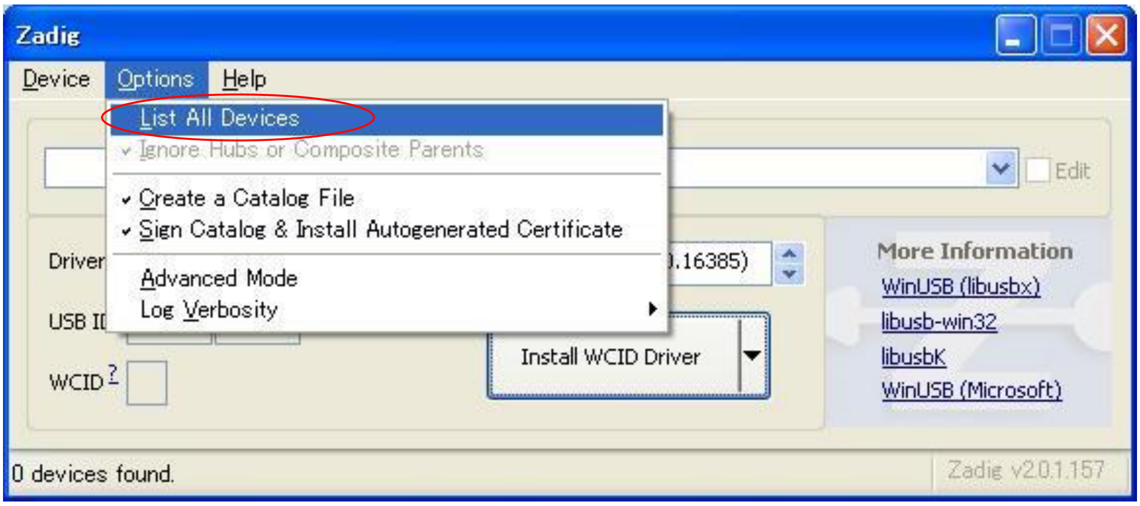
http://sourceforge.net/projects/libwdi/files/zadig/

3.2 Installing LibUSB

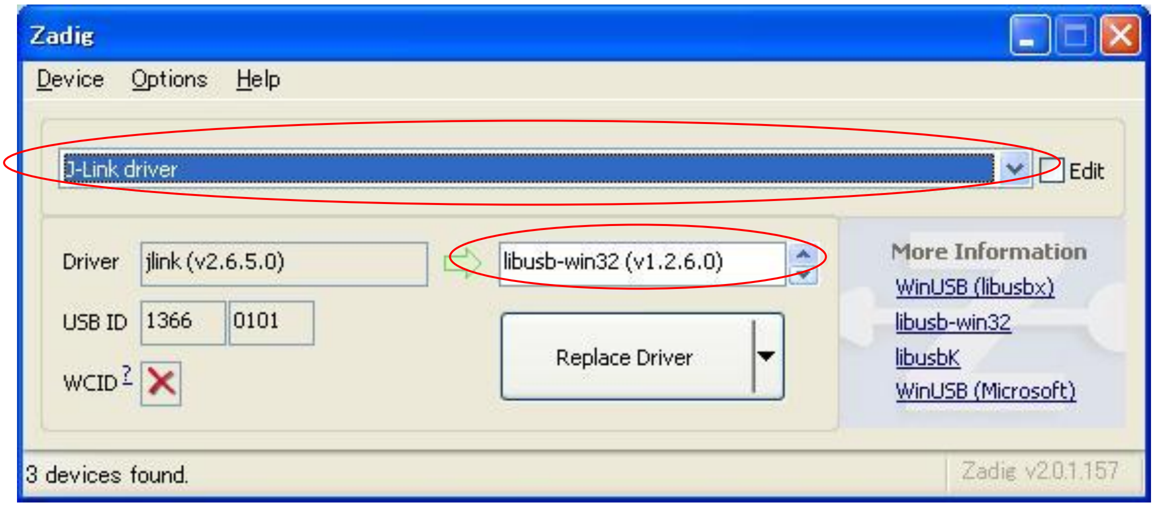
Please connect J-Link and your PC. If ordinary driver is set in J-Link, it doesn't need deleting. When Zadig starts, the next window below will be displayed.



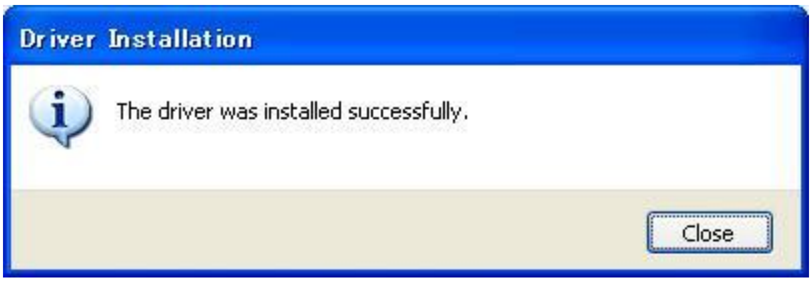
Please Click on *Option→List All Devices.*



Chose *J-Link driver* from pull-down menu, please set *libusb-win32 (v1.2.6.0)* in *Driver*.



If click on *Replace Driver*, replacing of driver will start.



Please confirm that *J-Link driver* is included in *libusb-win32 devices* from the device manager window.

4 Debugger

4.1 Open On-Chip Debugger (OpenOCD)

The Open On-Chip debugger is an open source software solution for accessing embedded ARM cores via JTAG hardware interface “JTAG dongle”.
OpenOCD support many of JTAG dongles. The most of this dongles are based of the FTDI USB device chip FT2232D from Future Technology Devices International Ltd.
This chapter describes the method to use OpenOCD.

4.2 Using LibUSB driver

4.2.1 Installing OpenOCD which supported LibUSB

The Windows installer program for the version of OpenOCD that support LibUSB driver can be downloaded from the website: (Please use OpenOCD 0.5.0 or later for FM3 family)
http://openocd.sourceforge.net/

For the next steps it is needed to recall the location of the folder, where OpenOCD was installed, e.g. *C:¥OpenOCD_LibUSB*.

4.2.2 Run OpenOCD

A configuration script file openocd.cfg for OpenOCD is also needed (This file is included in the software package of this application note). The OpenOCD configuration file openocd.cfg for the MB9BFD18T example is shown below

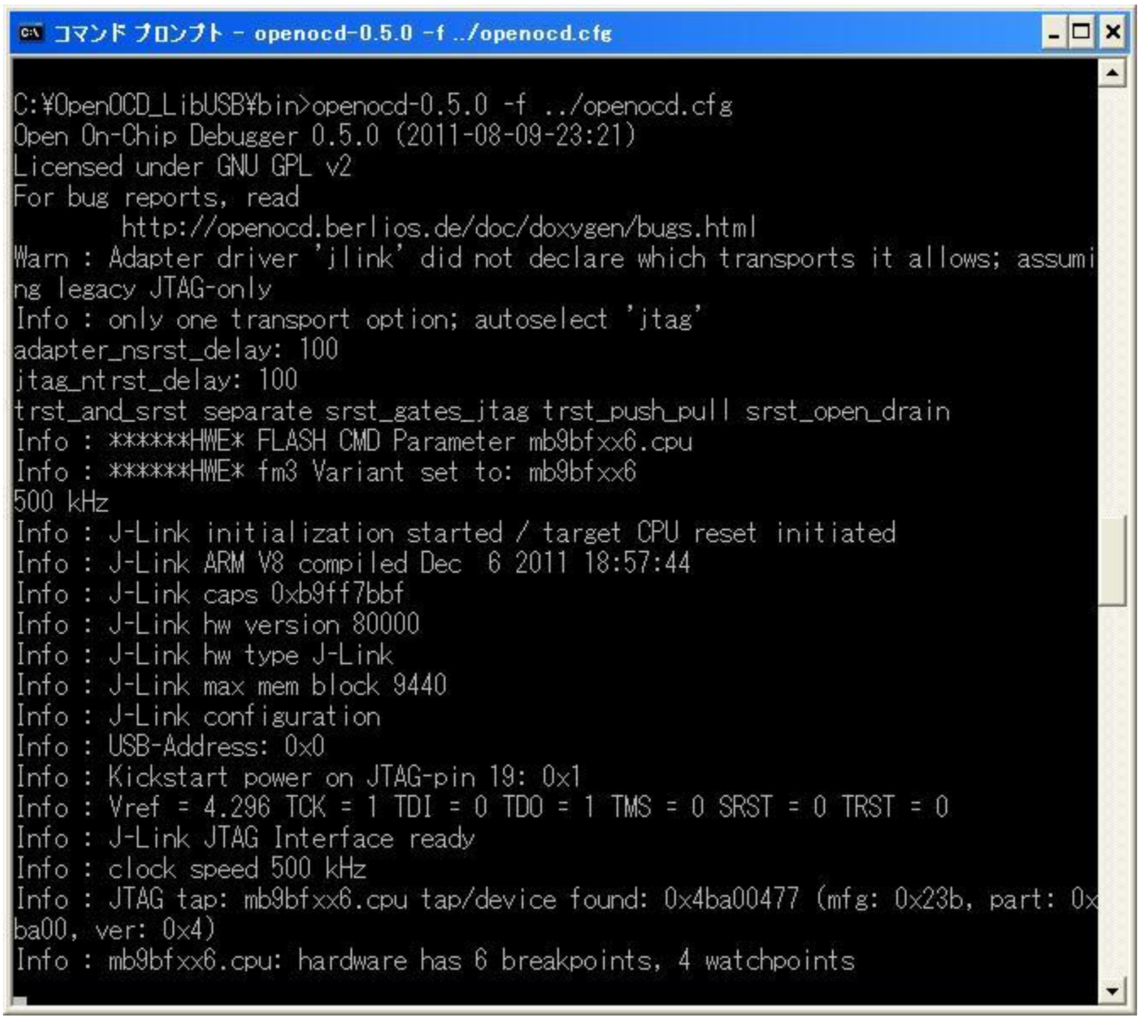
```
#interface jlink    } If using J-Link, please set this line enable.

#interface ft2232                                     } If using ARM-USB-TINY,
#ft2232_device_desc "Olimex OpenOCD JTAG TINY"        } please set these lines enable.
#ft2232_layout olimex-jtag
#ft2232_vid_pid 0x15ba 0x0004

# Fujitsu Cortex-M3 with 1MB Flash and 64*2 kB RAM

if { [info exists CHIPNAME] } {
        set  _CHIPNAME $CHIPNAME
} else {
        set  _CHIPNAME mb9bfxx6
}

if { [info exists ENDIAN] } {
        set  _ENDIAN $ENDIAN
} else {
        set  _ENDIAN little
}

if { [info exists CPUTAPID ] } {
        set _CPUTAPID $CPUTAPID
} else {
        set _CPUTAPID 0x4ba00477
}

#delays on reset lines
jtag_nsrst_delay 100
jtag_ntrst_delay 100

# Fujitsu cortex-M3 reset configuration
# reset_config trst_only
reset_config trst_and_srst

jtag newtap $_CHIPNAME cpu -irlen 4 -ircapture 0x1 -irmask 0xf -expected-id
$_CPUTAPID

set _TARGETNAME $_CHIPNAME.cpu
target create $_TARGETNAME cortex_m3 -endian $_ENDIAN -chain-position
$_TARGETNAME

# MB9BFD18 has 64*2kB of RAM on its main system bus
$_TARGETNAME configure -work-area-phys 0x1FFF0008 -work-area-size 0x8000
-work-area-backup 0
```

```
# MB9BFD18 has 1MB of user-available FLASH
# flash bank mb9bf500 <base> <size> 0 0 <target#> <variant> <cclk>
[calc_checksum]

set _FLASHNAME $_CHIPNAME.flash
flash bank $_FLASHNAME fm3 0 0 0 0 $_TARGETNAME mb9bfxx6

# 4MHz / 6 = 666kHz, so use 500
jtag_khz 500
```

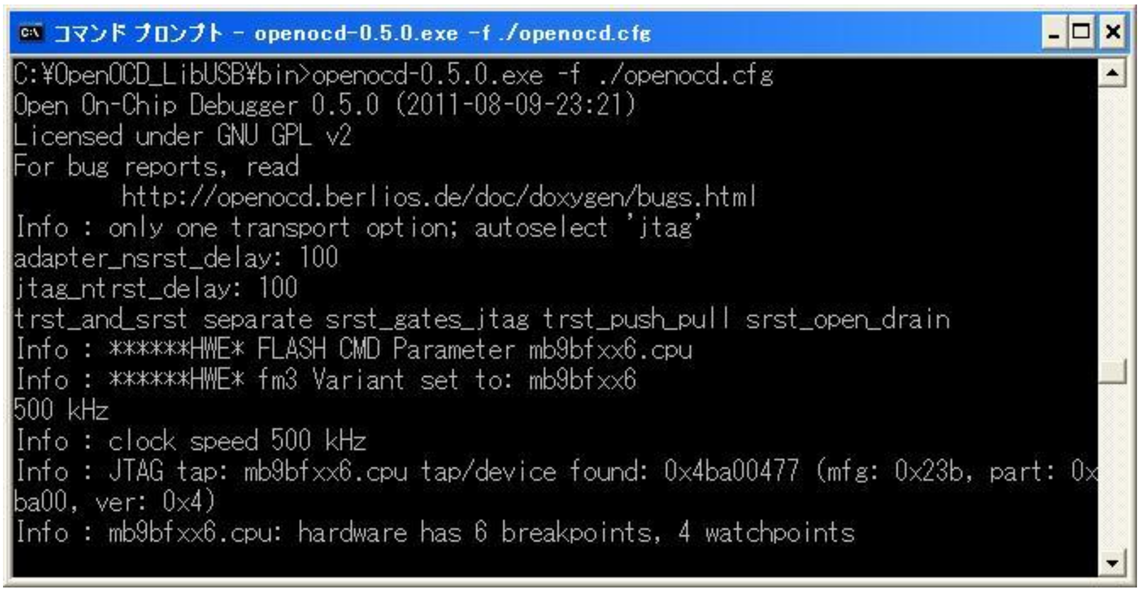
To run the OpenOCD server, start the windows prompt and go to the folder, where the OpenOCD executable file was generated, and run this program with the `-f` argument with the path to the configuration file above. For example:

```
>Openocd-0.5.0 -f <Your path to the Eclipse workspace project>/openocd.cfg
```

If using J-Link, please confirm that the following window is displayed.



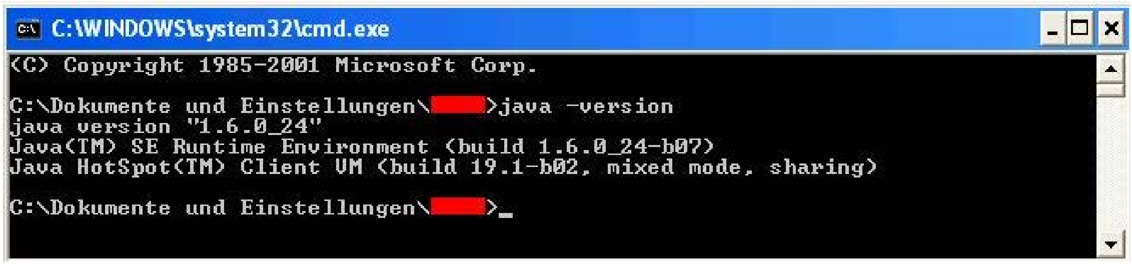
If using ARM-USB-TINY, please confirm that the following window is displayed.



5 Java Runtime Environment (JRE)

5.1 Checking for JRE

The installation of Eclipse requires the availability of Java as a virtual machine on system. To check, that Java already exists on the system, type the command **`Java –version`** on DOS console.

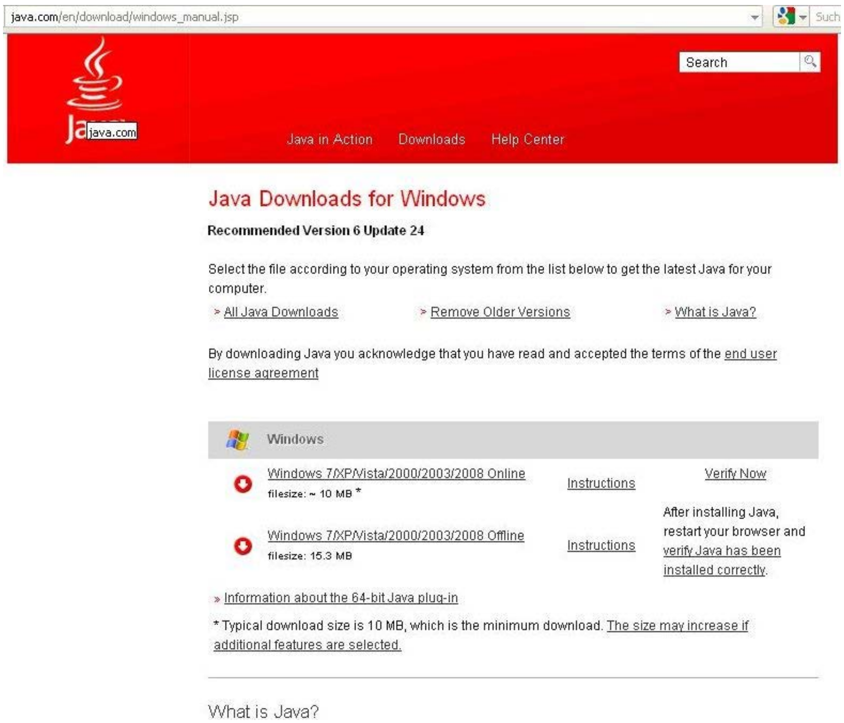


If windows cannot recognize this command, Java Runtime Environment (JRE) is needed to be installed.

5.2 Installing Java JRE

Download JRE from following URL:

http://java.com/

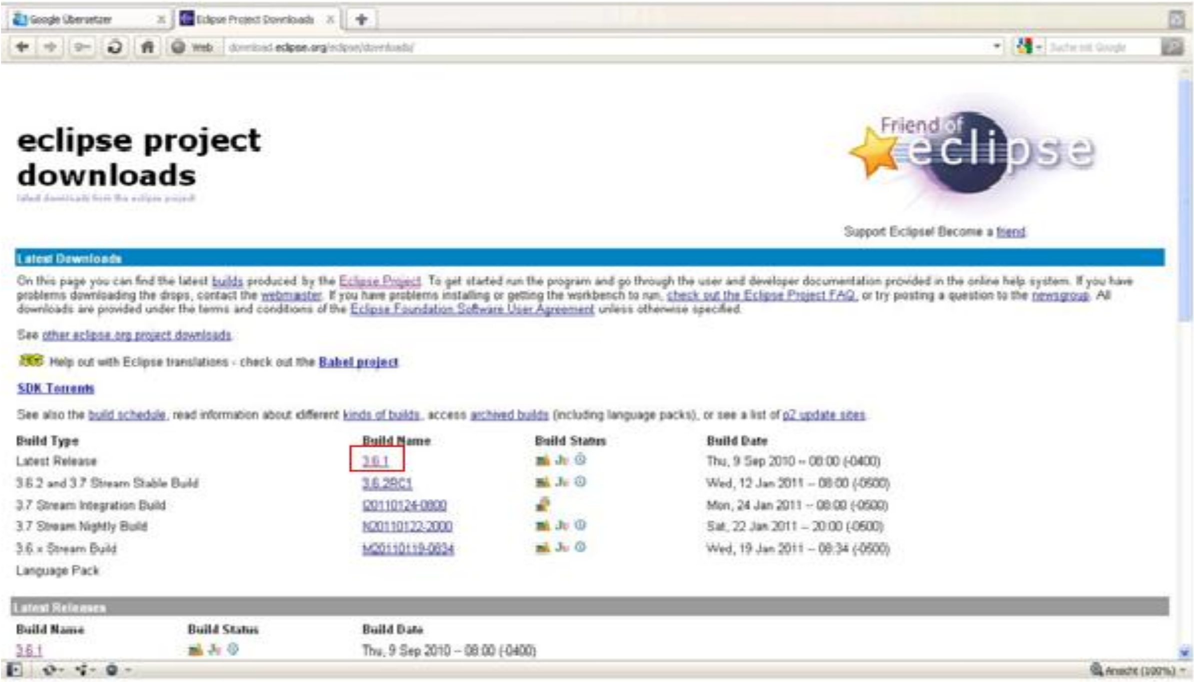


Java installation can be done online or offline. Download one of the installation programs and start the installation procedure to install JRE.

6 Eclipse platform

6.1 Eclipse platform

The latest release of eclipse is available to download from the web site:
http://download.eclipse.org/eclipse/downloads/



The Helios release 3.6.1 (or later) consists of various packages. These packages are available on the left menu of the download website of the Eclipse project.
For our system it is required a minimum eclipse platform to be realized. The package needed for this can be found by the menu section “Platform Runtime Binary”.

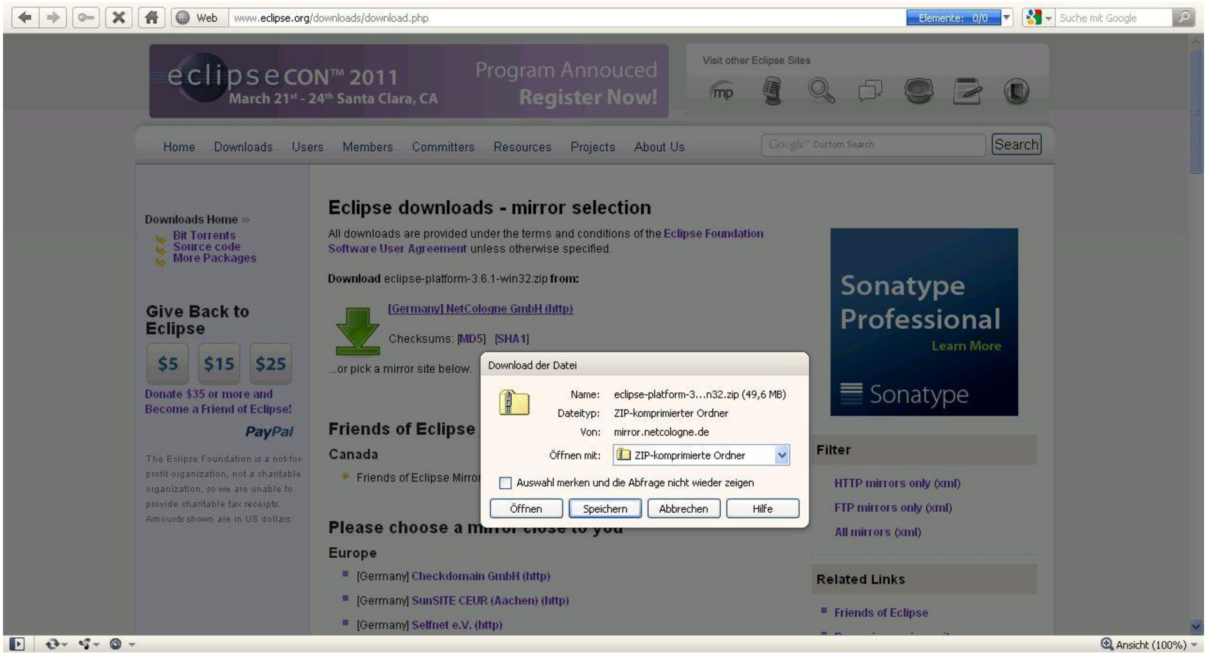
The Eclipse platform binary package is available for many operating systems.

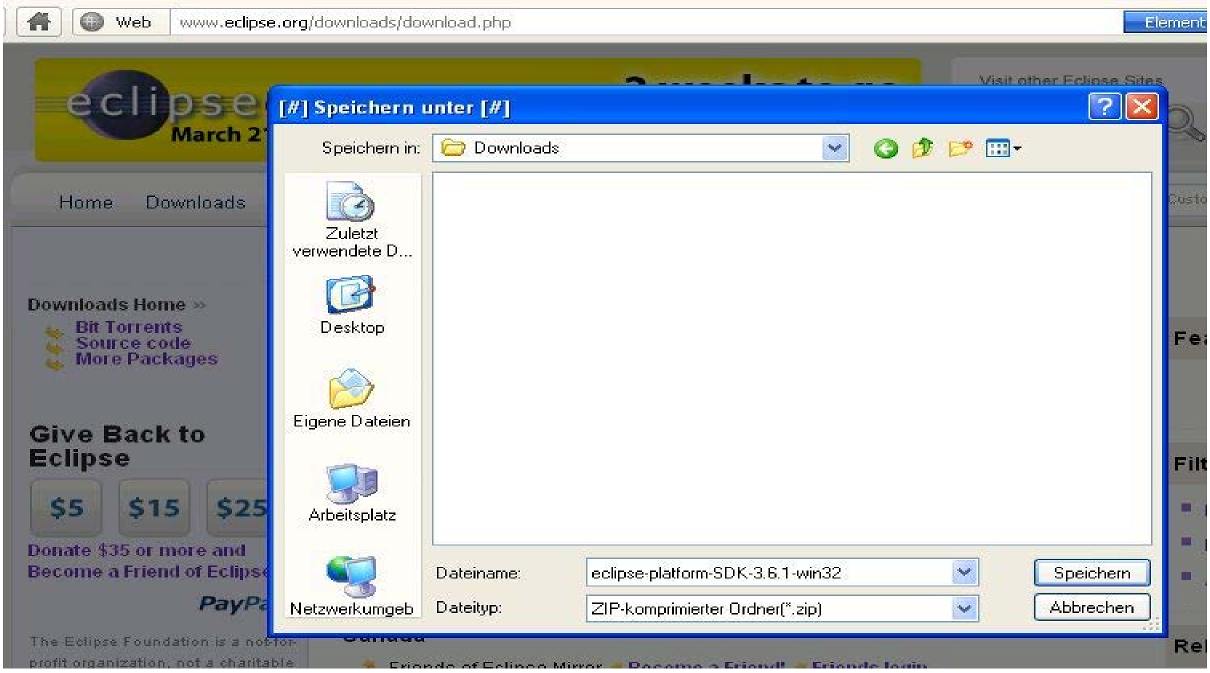
For Windows systems with 32 bit CPU use the first "http" location of this list to download the adequate Eclipse package for this system.

Platform Runtime Binary

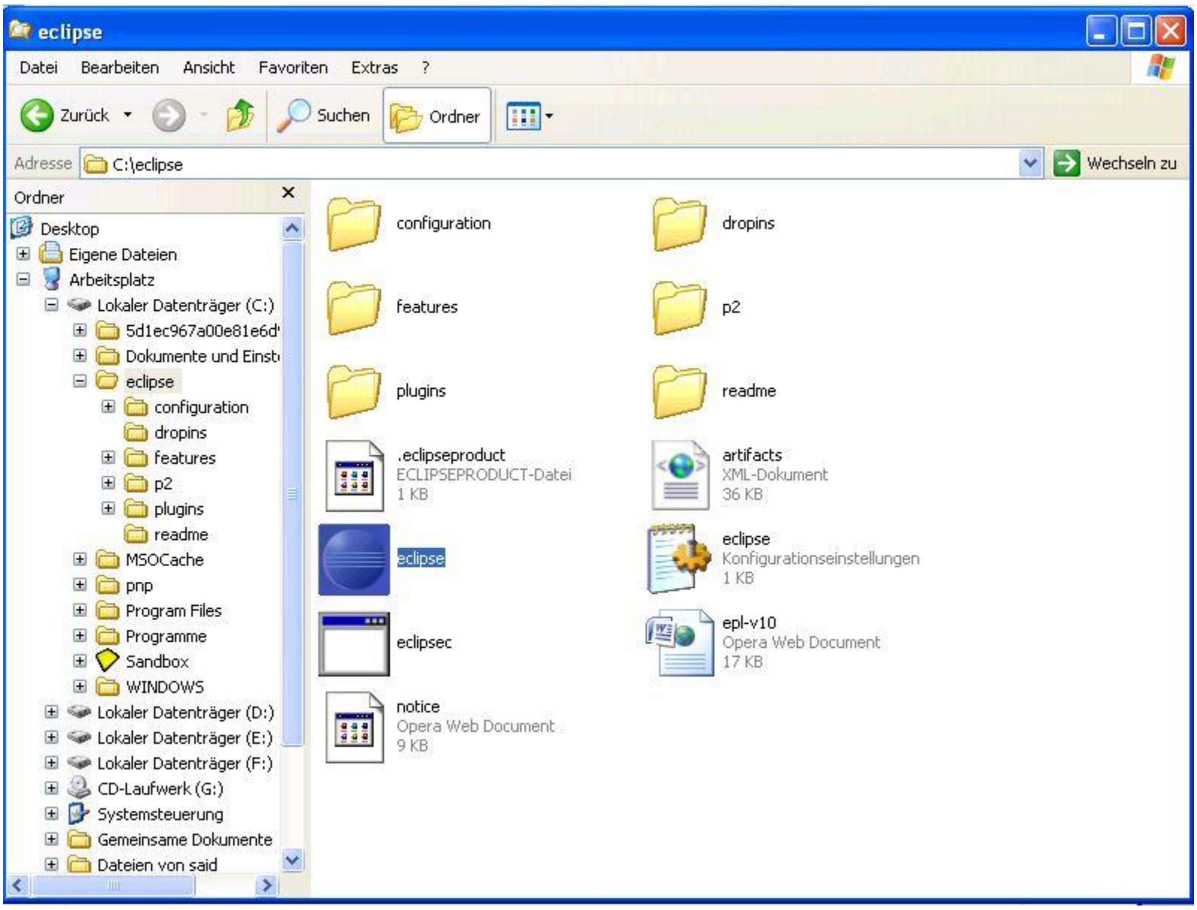
Status	Platform	Download	Size
	Windows (Supported Versions)	(http)	57 MB
	Windows (x86_64) (Supported Versions)	(http)	57 MB
	Linux (x86/GTK 2) (Supported Versions)	(http)	57 MB
	Linux (x86_64/GTK 2) (Supported Versions)	(http)	57 MB

The Eclipse platform binary is available from many http mirrors. After choosing one of these mirrors the software can be downloaded.





After downloading and saving the zip file eclipse-platform-SDK-3.6.1-win32.zip, decompress this file, to e.g. *C:¥*

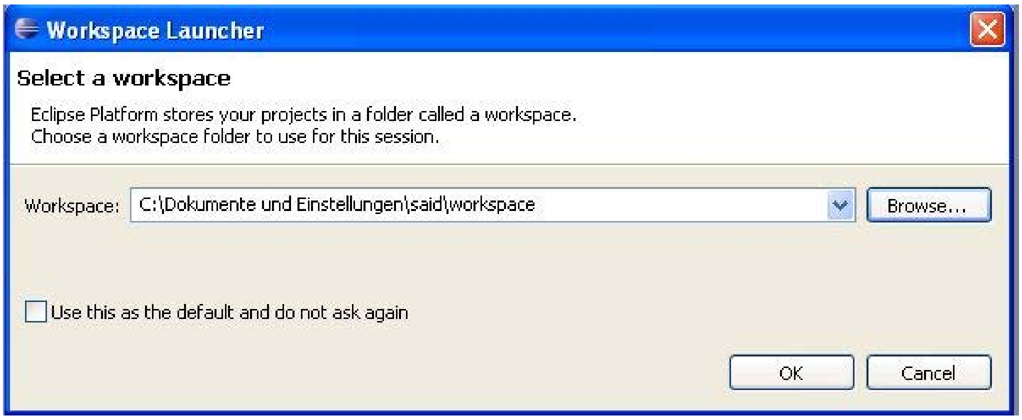


With the installation of Eclipse platform runtime binary, this installation of Eclipse is finished.

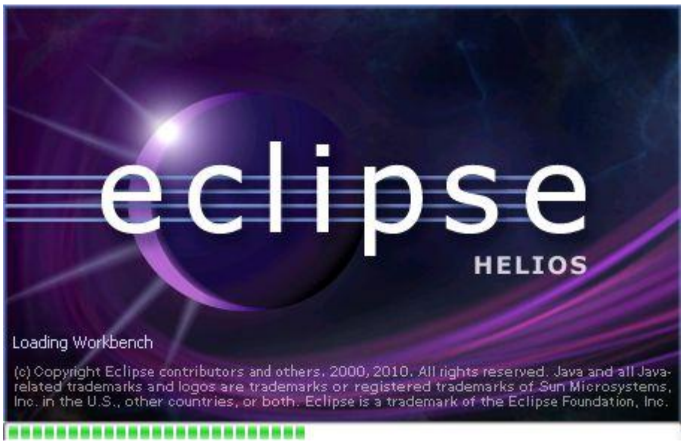
6.2 Start Eclipse IDE

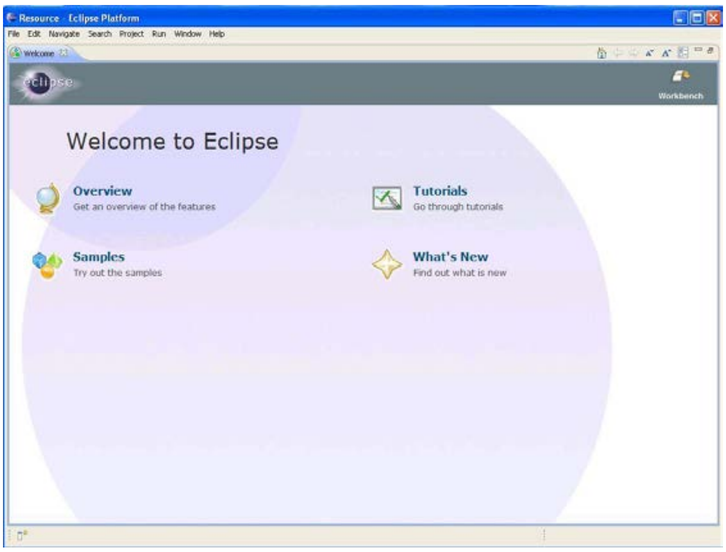
The Eclipse IDE is now ready to start; for this start eclipse.exe from the folder *C:¥eclipse*.

At first the workspace, where Eclipse should store the project files, has to be specified.



After the selection of the workspace, Eclipse starts.

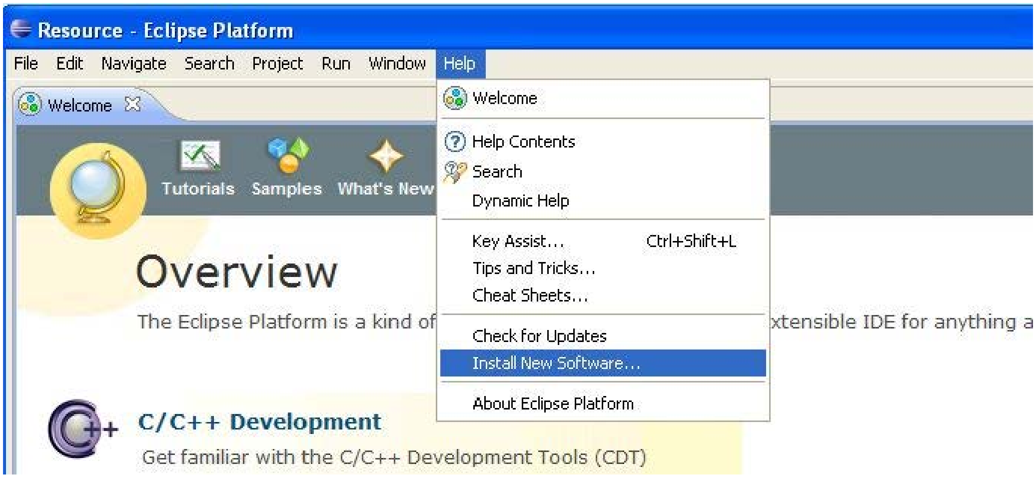




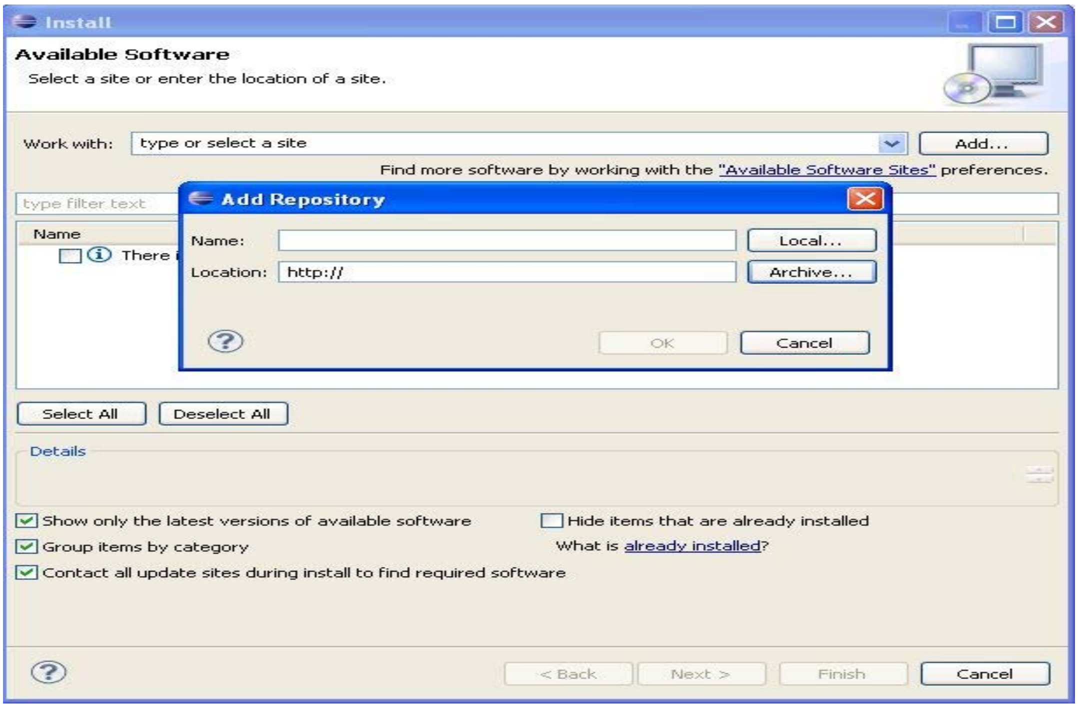
7 C/C++ Development Tooling CDT

7.1 Installation of new software on Eclipse

After the installation of Eclipse, it is necessary to import the CDT package to Eclipse for developing C or C++ applications. The CDT package is available as a plug-in.
To install new software on Eclipse, start Eclipse and follow the installation instruction via the *Help→Install* New Software menu.

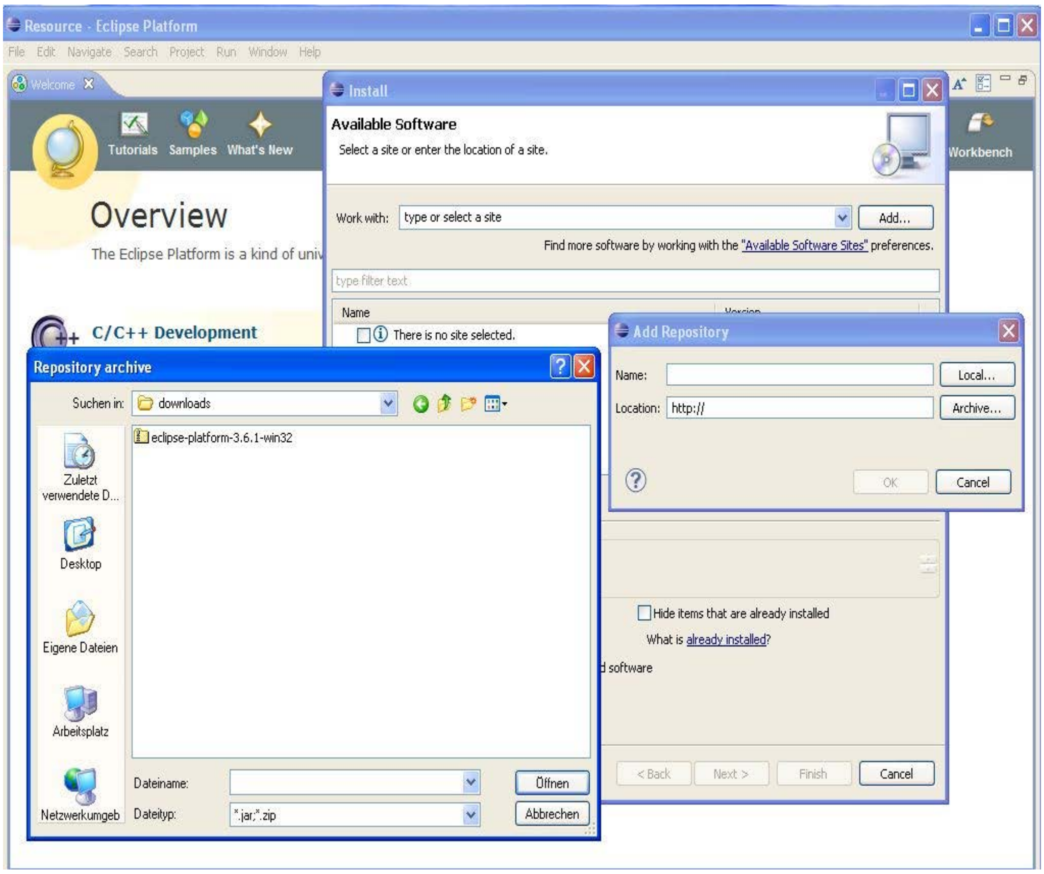


The installation of CDT plug-in or any another package to the Eclipse platform depends on the procedure, which the user selects to add this software to the platform. After clicking of the *add* button the *Add Repository* window appears.

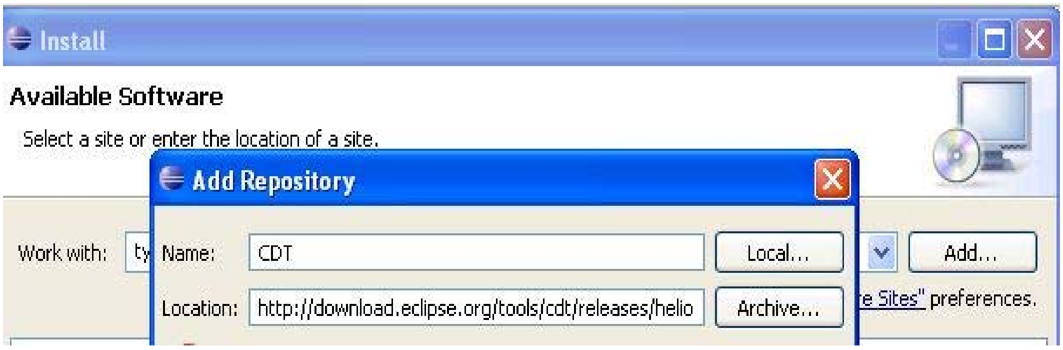


Eclipse supports two different methods to implement new plug-ins to the platform:

When the plug-in is available locally on the system as *JAR* or *ZIP* file, the installation can be done offline.



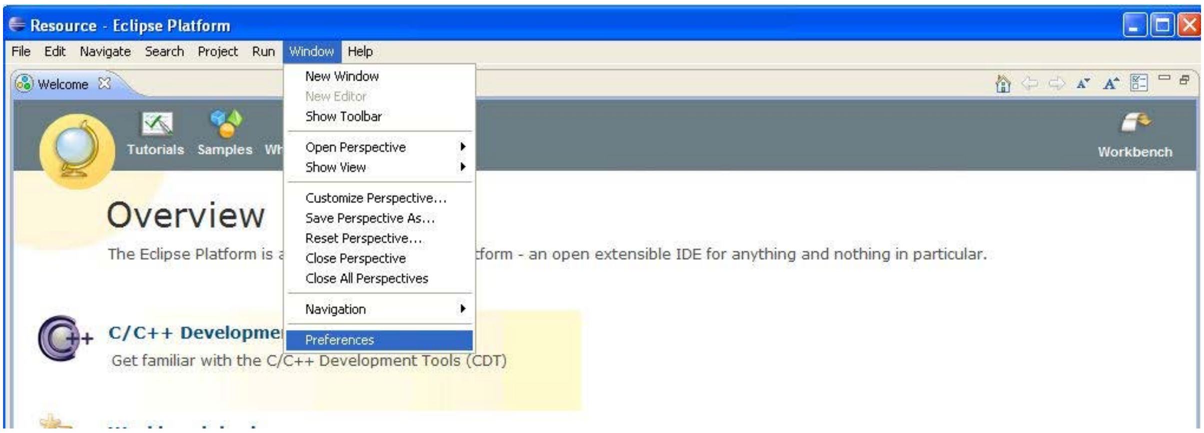
When the plug-in is available from an http project website, a new installation or update of this software is done online.



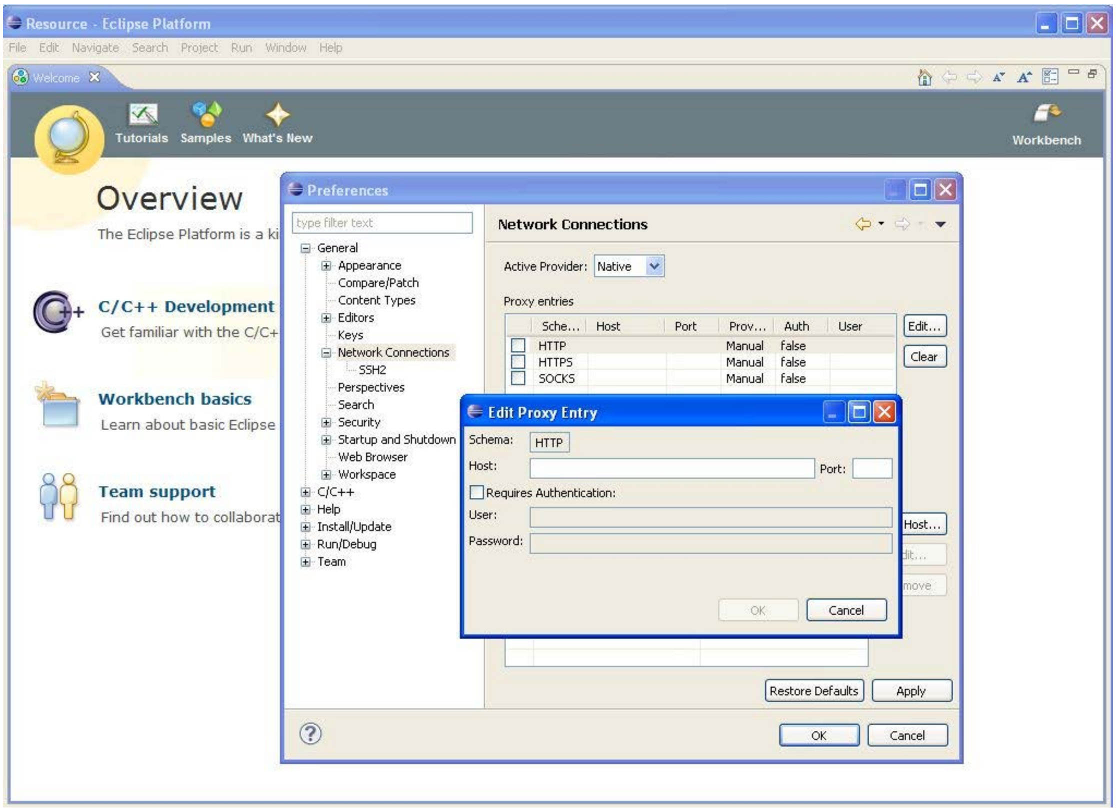
The online method is recommended. For this procedure first adapt the Eclipse network settings to the network configuration before initiate the installation procedure.

7.2 Eclipse Network Configuration

From the Eclipse sub menu *Preferences* on the category *Window*, configure the settings for your network.



The configuration of the network can be realized from the *Network connections* field. From this field, edit the network setting entry and do the necessary changes to enable for Eclipse the communication to the internet.



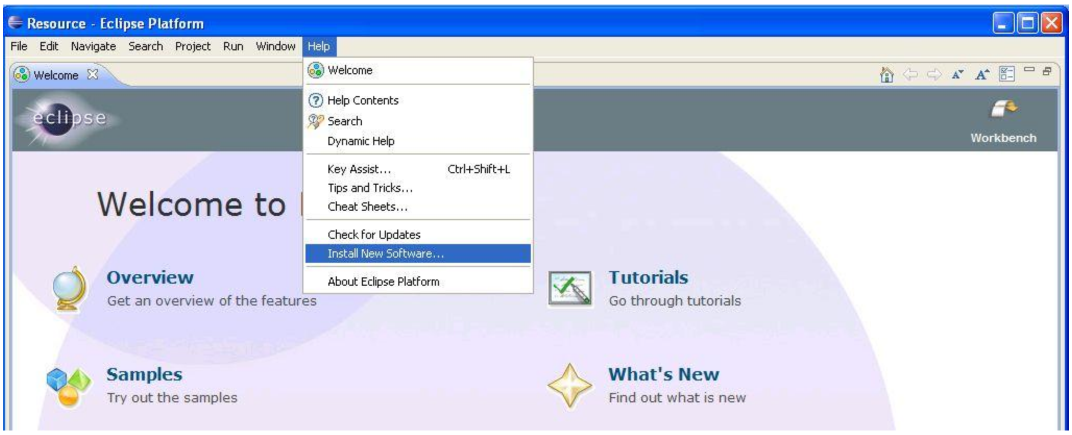
After this change click the *Apply* button to save the new network configuration. Now the online installation of the CDT plug-in can be done.

7.3 Eclipse CDT Plug-In

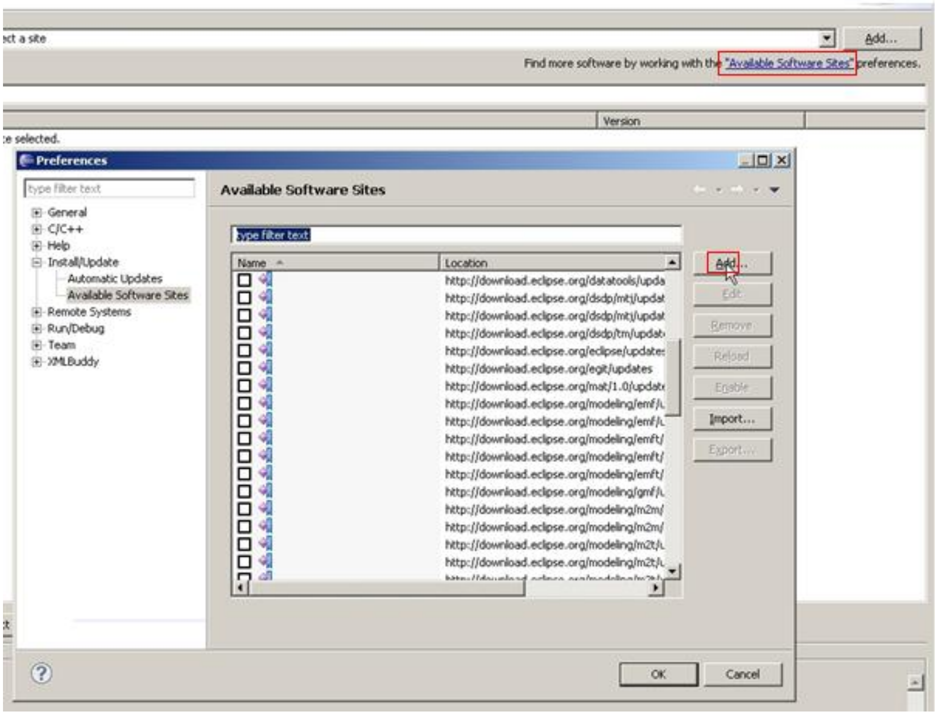
The CDT plug-in exists in a Standard and a Zylin version, but only the installation of one version is required.

For the integration of new CDT plug-ins on eclipse-platform, the demonstration of this installation follows below.

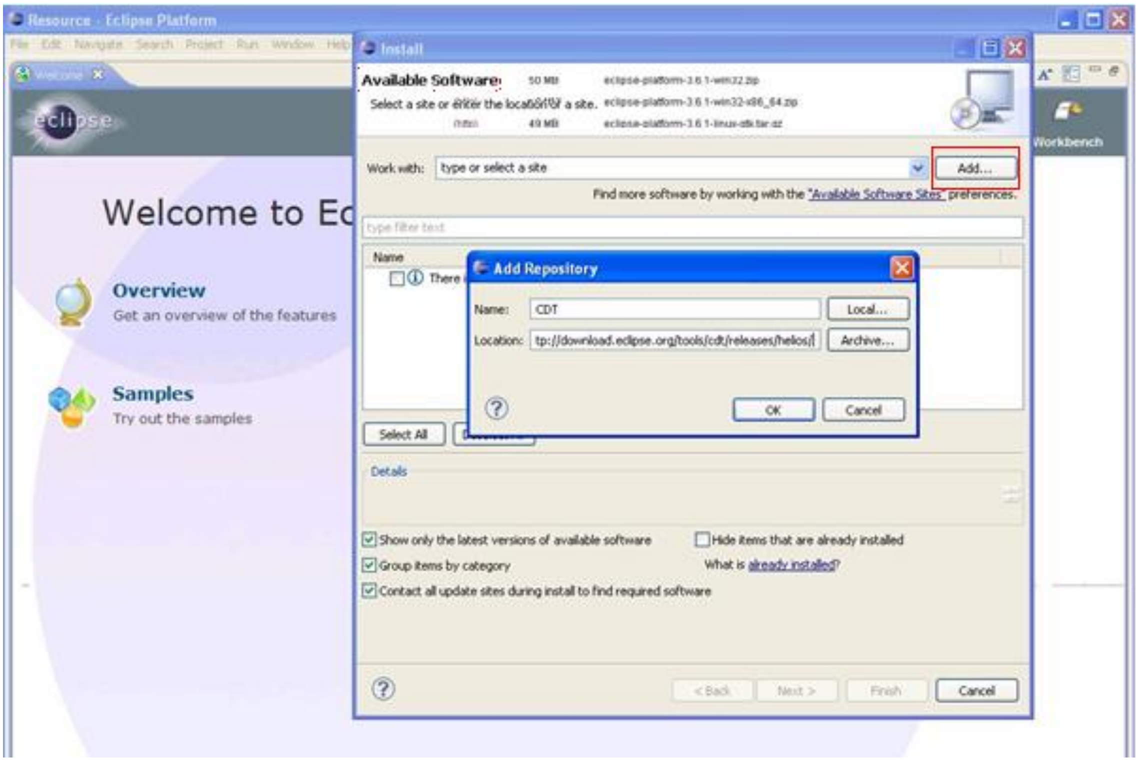
Under *Help* menu, click on *Install New Software...* .



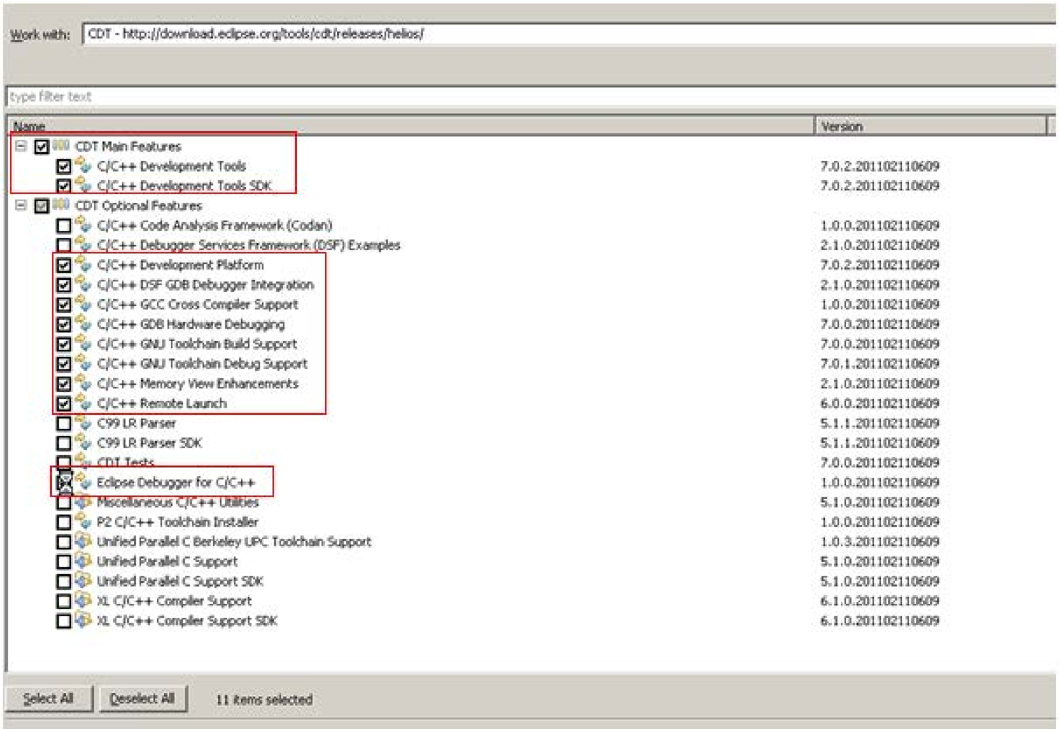
On the next window, click on *Available Software Sites* to look for a CDT downloading mirror, if existing. The mirror is: http://download.eclipse.org/tools/cdt/releases/helios/.



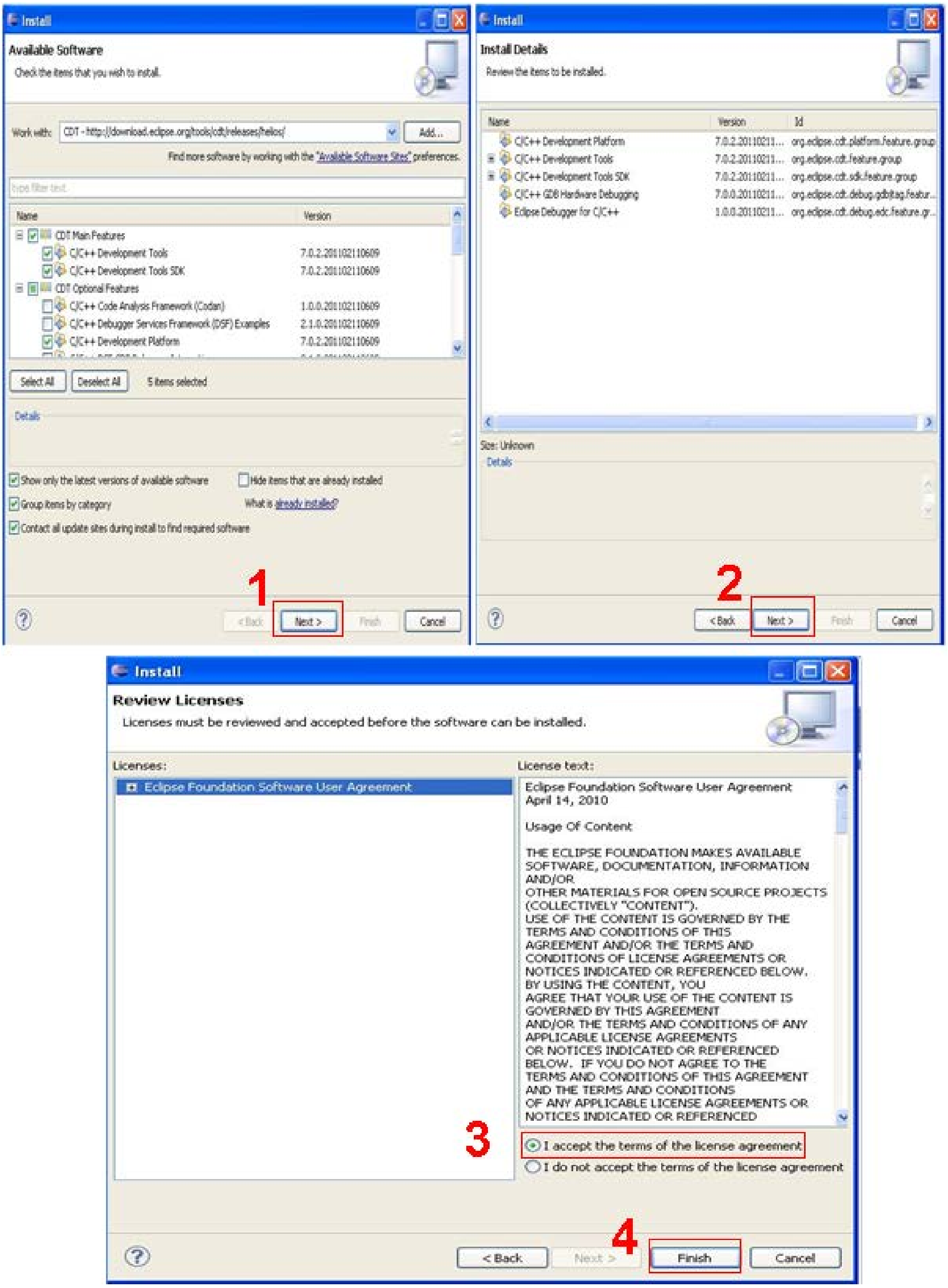
Otherwise click on *Add* to set the required mirror. Enter *CDT* for the name and http://download.eclipse.org/tools/cdt/releases/helios/ for the web location.



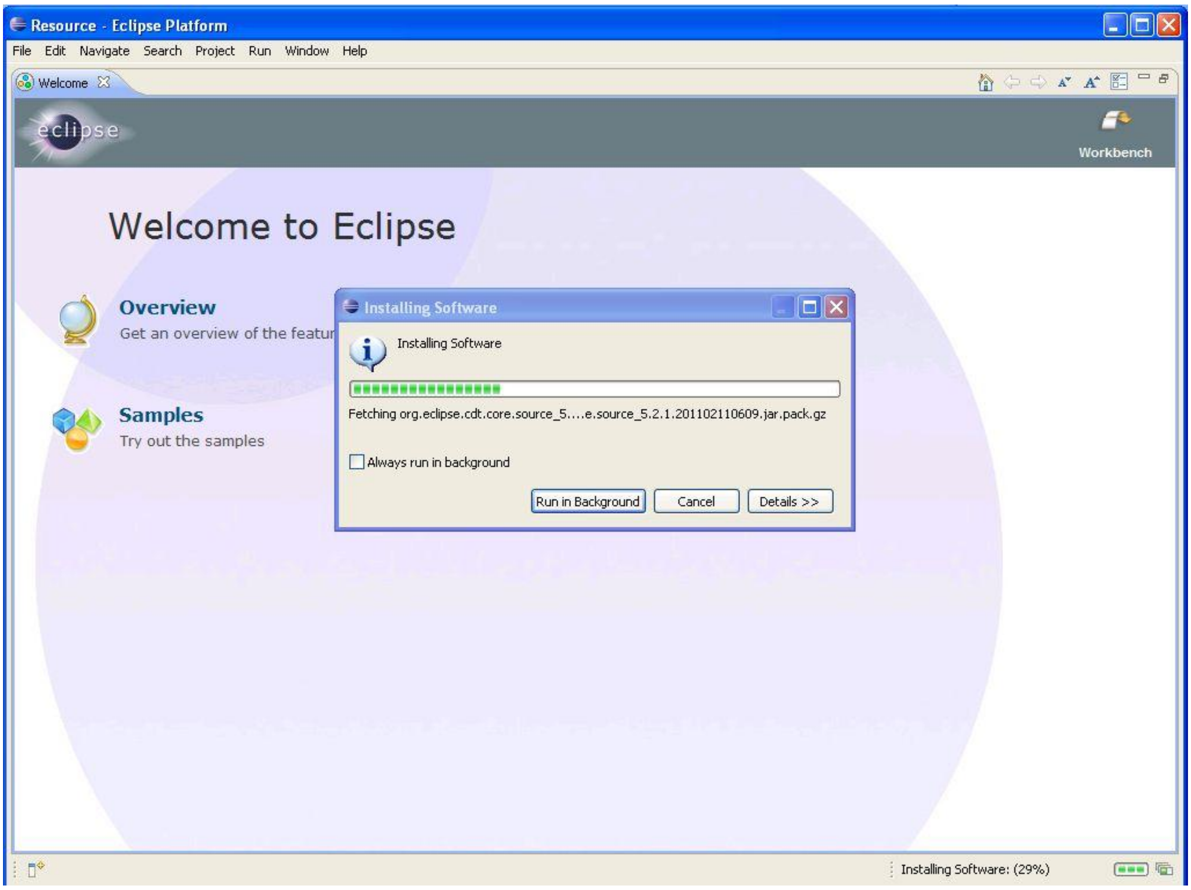
Click on *OK* and the next window below will be displayed. Select both *CDT MAIN Features* and the *CDT Optional Features* listed below only.



Follow the next steps to start the plug-in installation.



Eclipse starts then the installation of CDT plug-in.



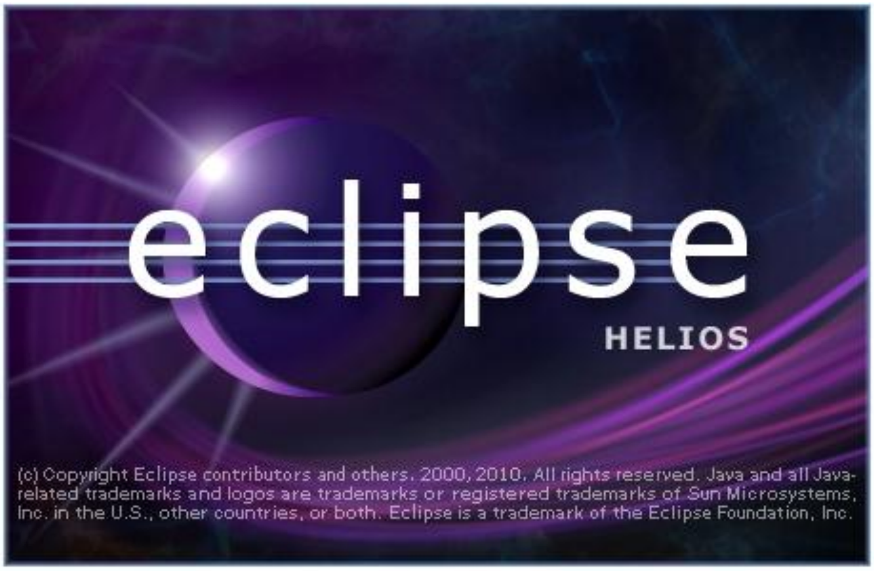
When the plug-in installation has finished, restart Eclipse IDE.

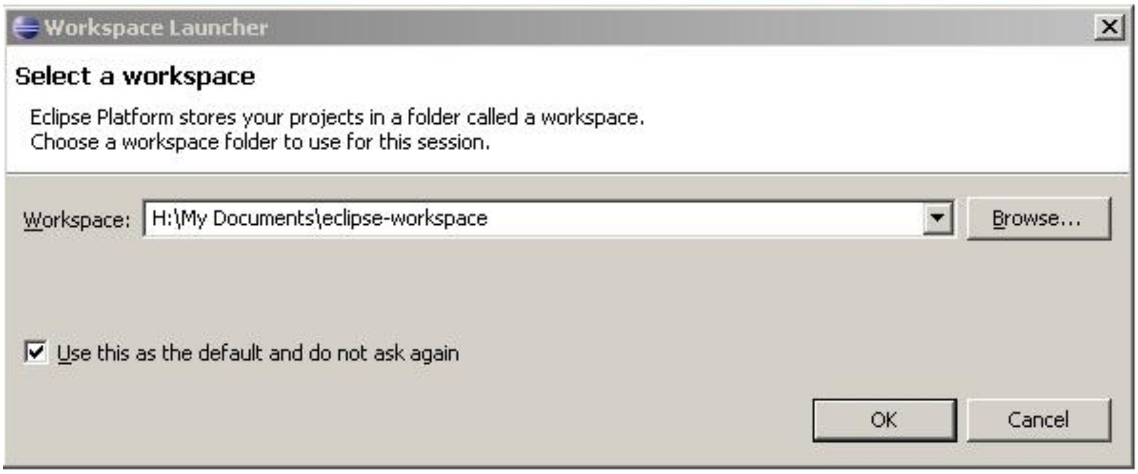
8 Working with the Eclipse IDE

8.1 C/C++ perspective

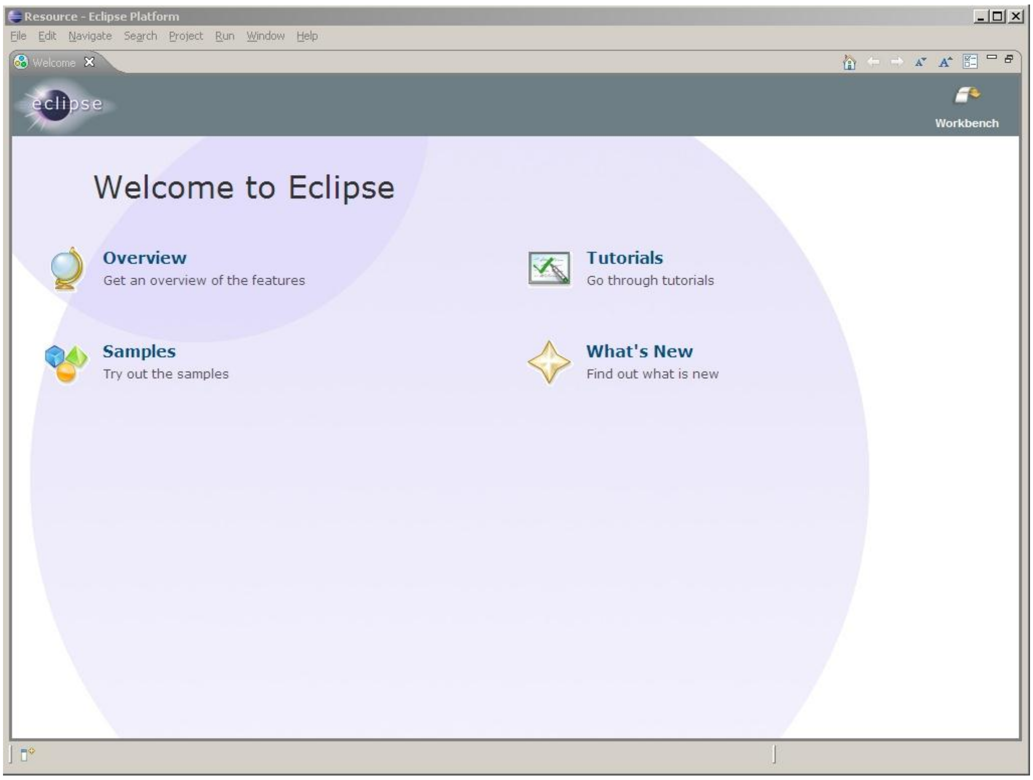
Start the Eclipse IDE.

At this point, Eclipse will present a "Workspace Launcher" dialog, shown below. This is where you specify the location of the "workspace" that will hold your Eclipse/CDT projects (see also the previous chapter 6.2)



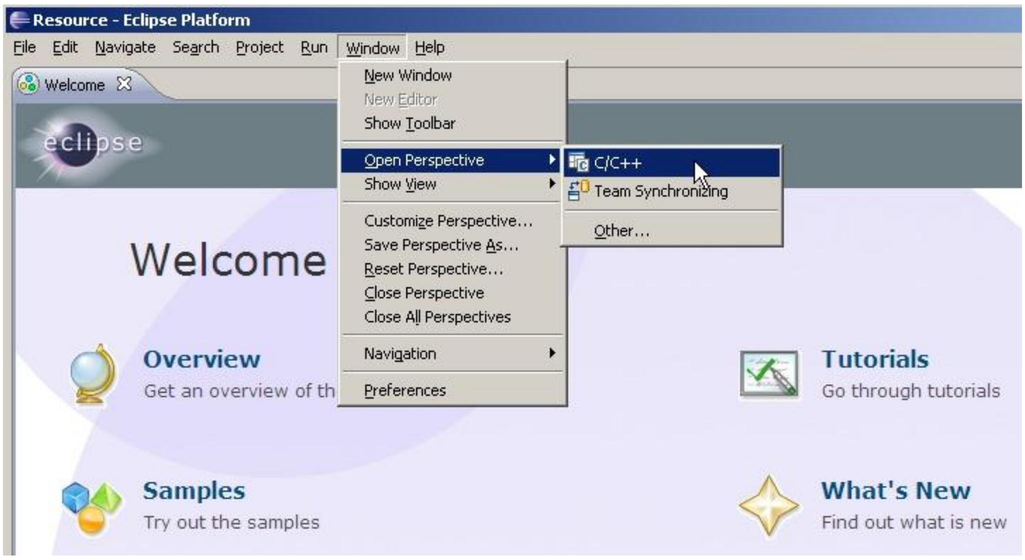


Now Eclipse will officially start and show the “Welcome” page shown below.



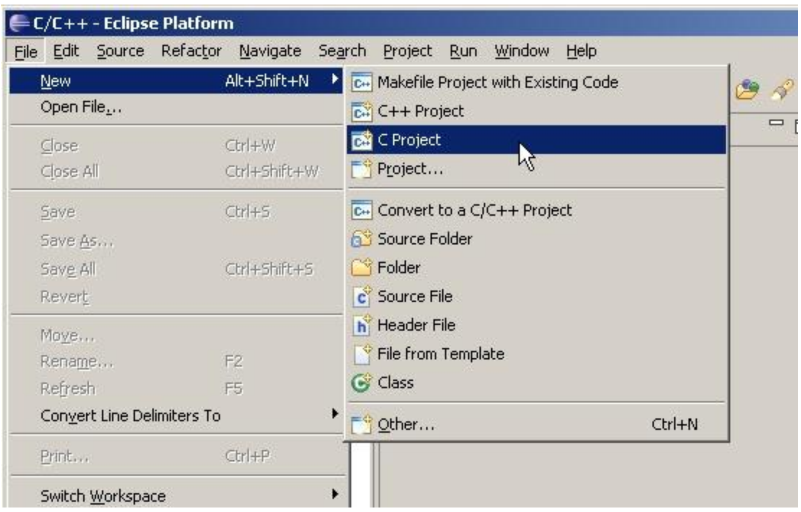
For project developing on C/C++, switch to the C/C++ perspective.

Choose *Window→Open Perspective*, then click on *C/C++* to open Eclipse in the C/C++ perspective.



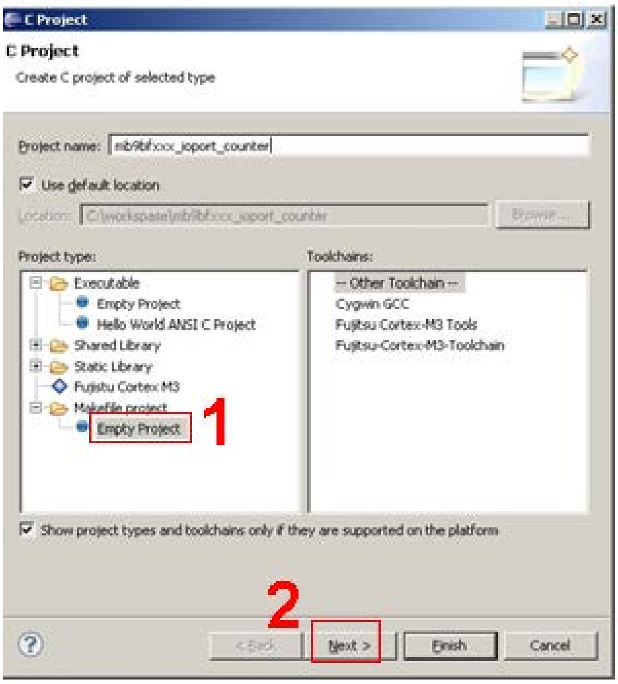
8.2 Creating a C or C++ project with Eclipse

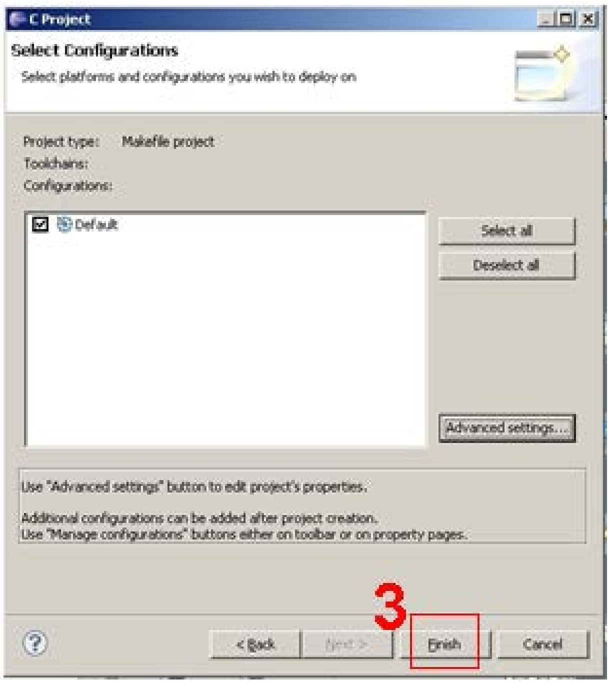
In the Eclipse C/C++ perspective a new project for your target can be created, here: Spansion Cortex M3. For this choose *File→New→C Project*.



In the "New Project wizard" shown below-left, expand the *Makefile project* branch by clicking on it's "+" sign and then select *Empty Project*. Enter the sample project name e.g. "*mb9bfxxx_ioport_counter*". Then click on *Next* to continue.

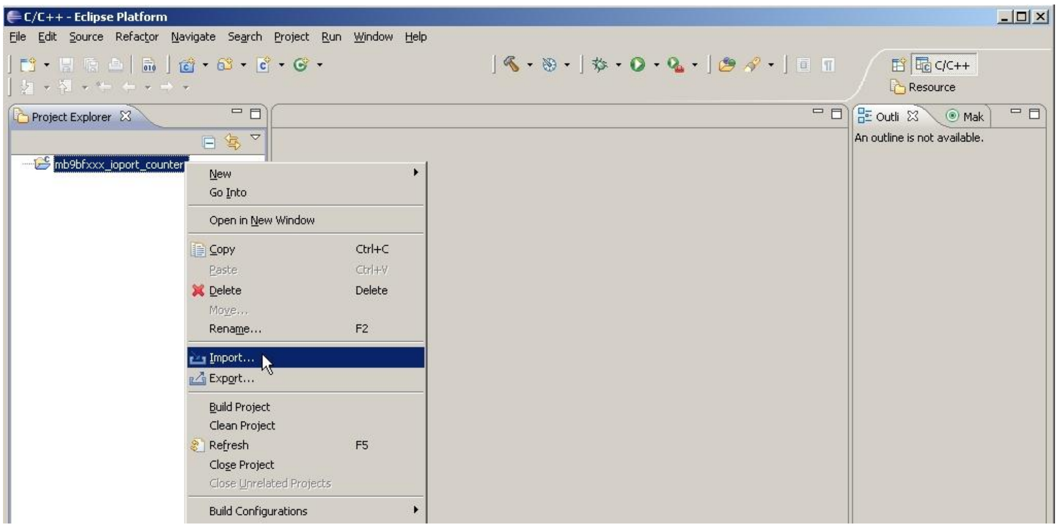
On the below-right window just close the wizard with *Finish*.



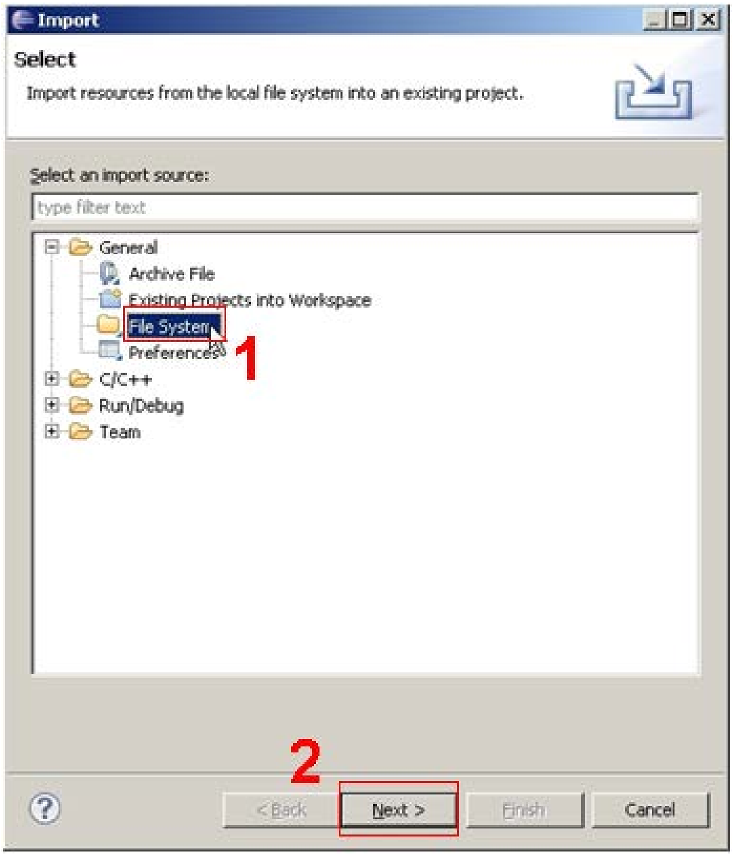


Now the C/C++ perspective shows a valid project, as shown below in the C/C++ Projects view on the left, but there are no source files in that project. Normally you would select *File→New→Source File* and enter a file name and start typing. This time, however, we will import already existing source files.

In the Eclipse screen below click on *File→Import…* . This will bring up the file import dialog.

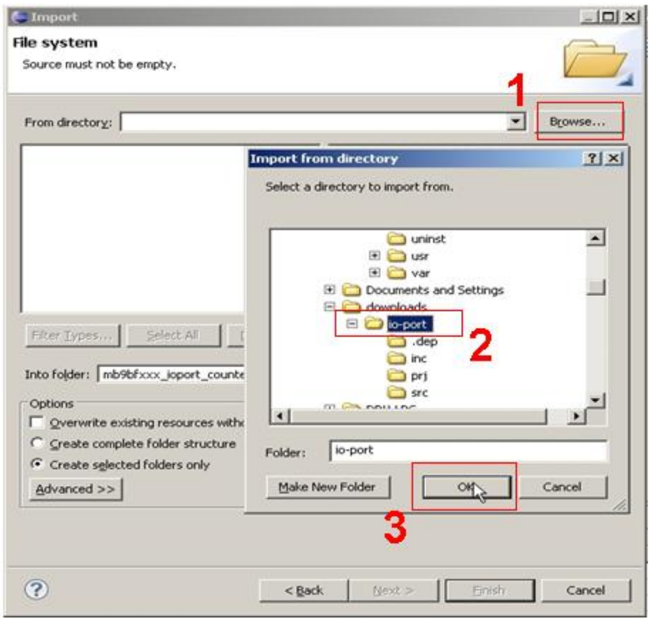


In the “Import” screen below, click on *File System* and then click *Next* to continue.

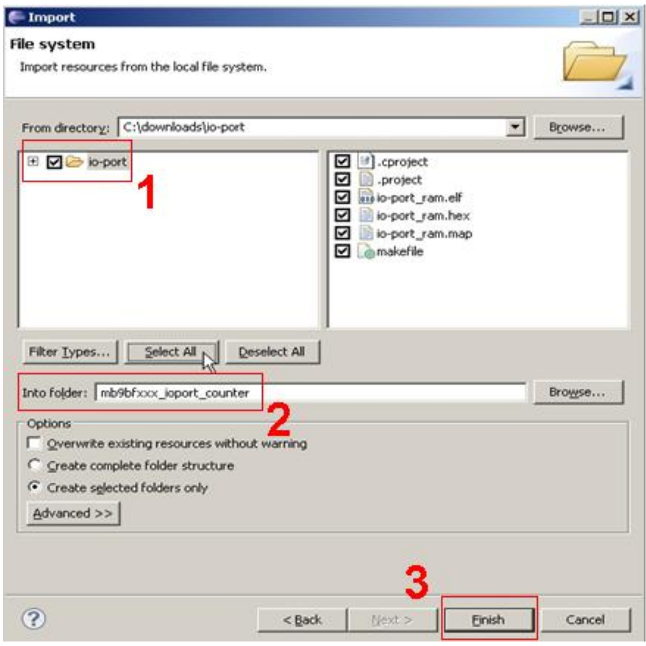


In the *Import→File System* screen below, use the *Browse* button associated with the *From directory* text box to search for the sample project to be imported.

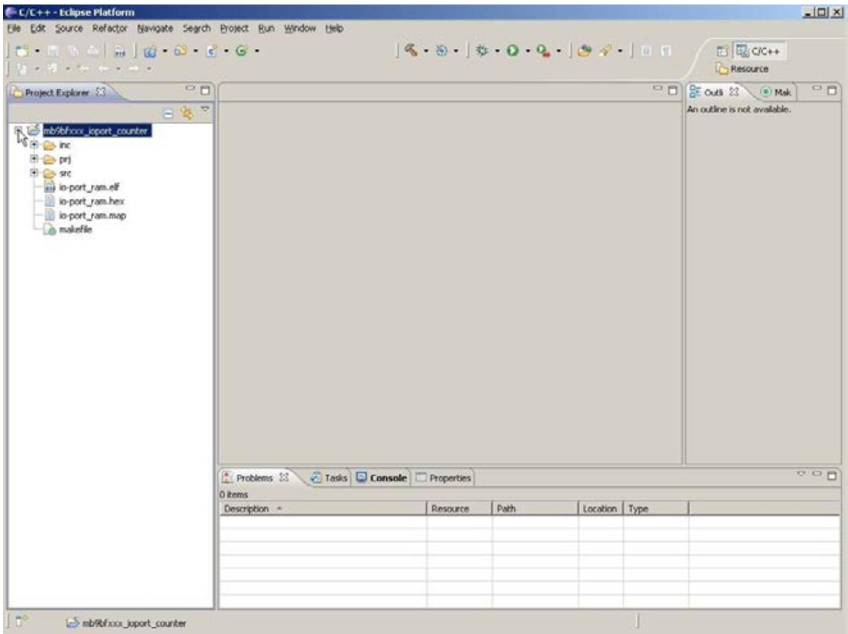
The project template *io-port* used in this application note, which is included in the note’s software package archive. The sample project *io-port* should be then saved, in a directory folder e. g. *C:¥downloads¥io-port*.



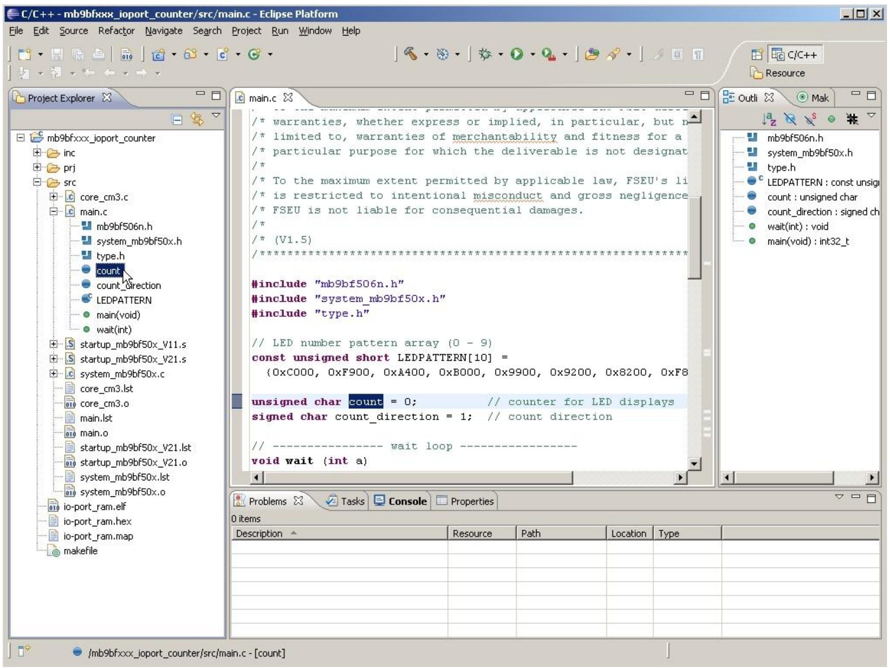
Check the box for the folder of the *io-port* example and then click the ***Select All*** button below because we want to import each of its files. Click *Finish* to start the file import operation.



Expanding the *mb9bfxxx_ioport_counter* project in the *C/C++ Projects* view seen below, shows that all the source files, which have been imported into the project. By clicking on the "+" sign on the project name in the C/C++ Projects panel on the left, the imported files are expanded in a tree view.

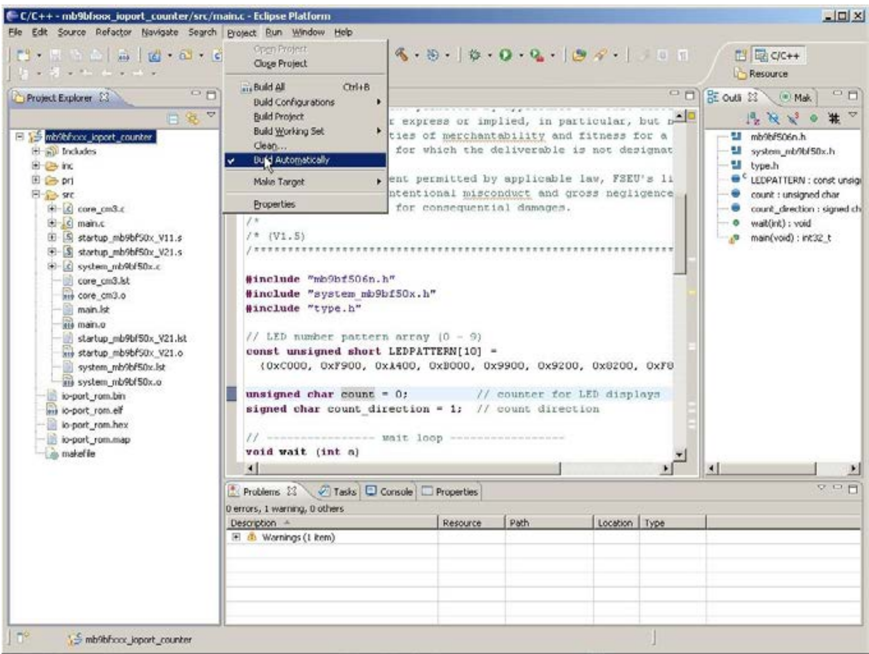


In the Eclipse window below, the *main.c* file has been selected by clicking on it and it thus be displayed in the source file editor view in the center. In the project explorer window the *main.c* module is expanded to reveal its variables and functions. By clicking e. g. on the variable `count`, the source window jumps to the definition of that variable.

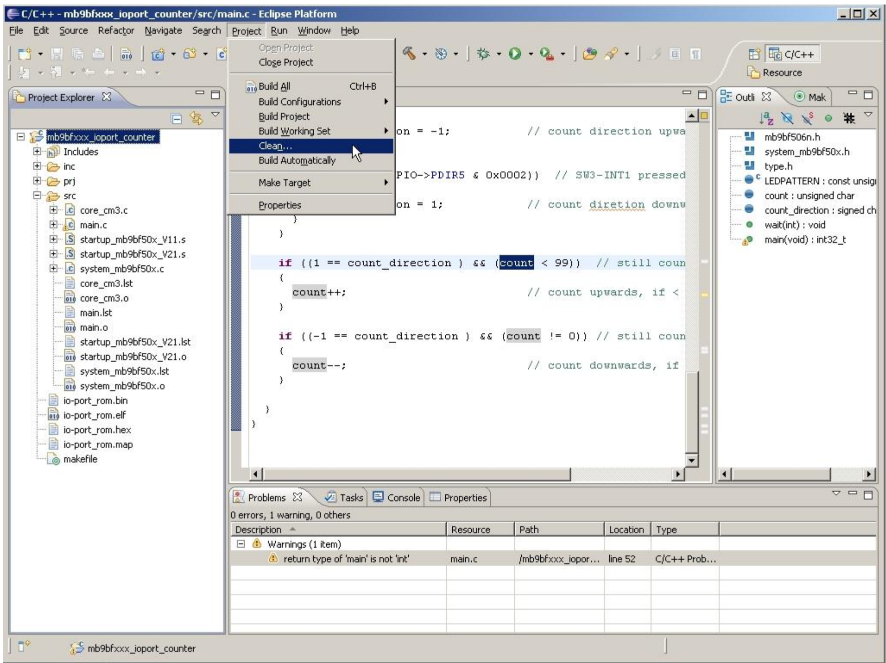


8.3 Cleaning the selected project

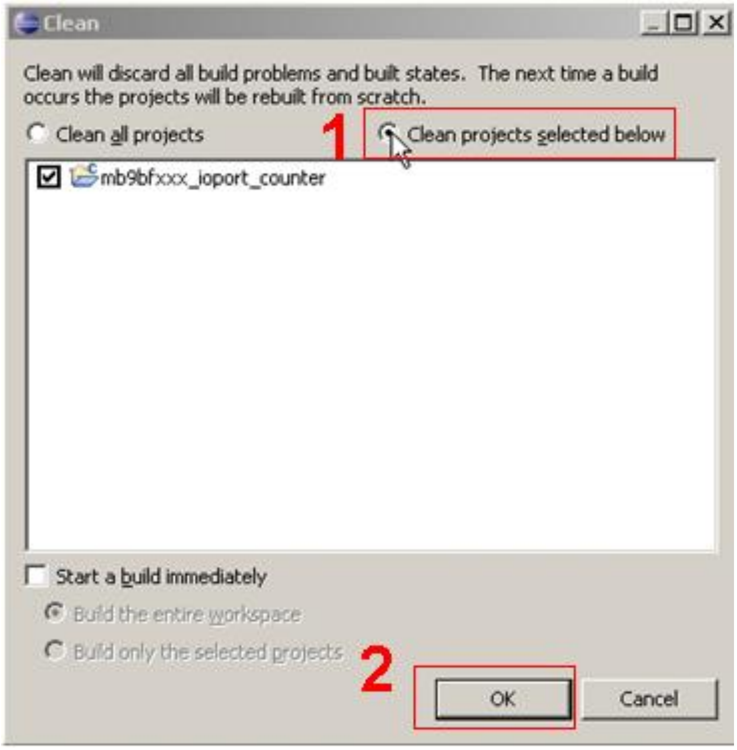
For compiling a project, first disable the automatically build. Select the project and from the category *Project* on the IDE menu uncheck *Build Automatically*.



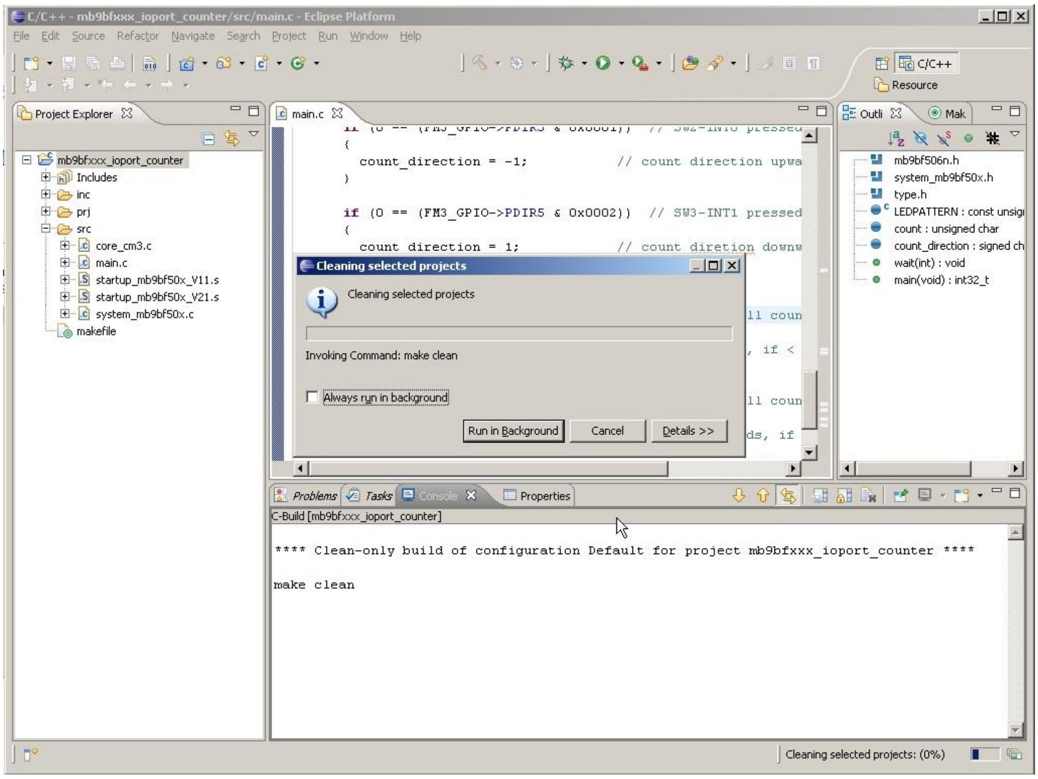
Now clean the project. In the same way select the project *mb9bfxxx_ioport_counter* from the project explorer window the category *Project*, and on the IDE menu choose *Clean...* .



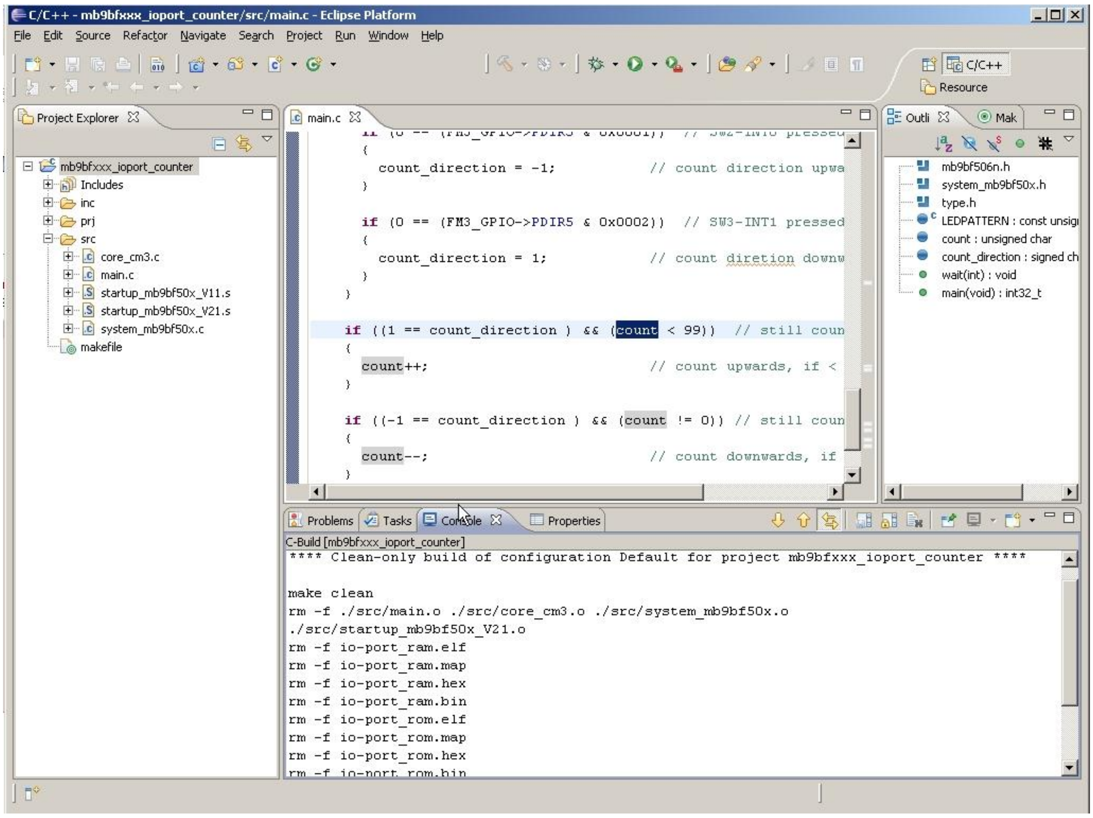
On the clean window deselect the option *Clean all projects* and select our project. Deselect also the option *Start a build immediately*.



Finish the configuration by clicking on the *OK* button and the clean process will start.



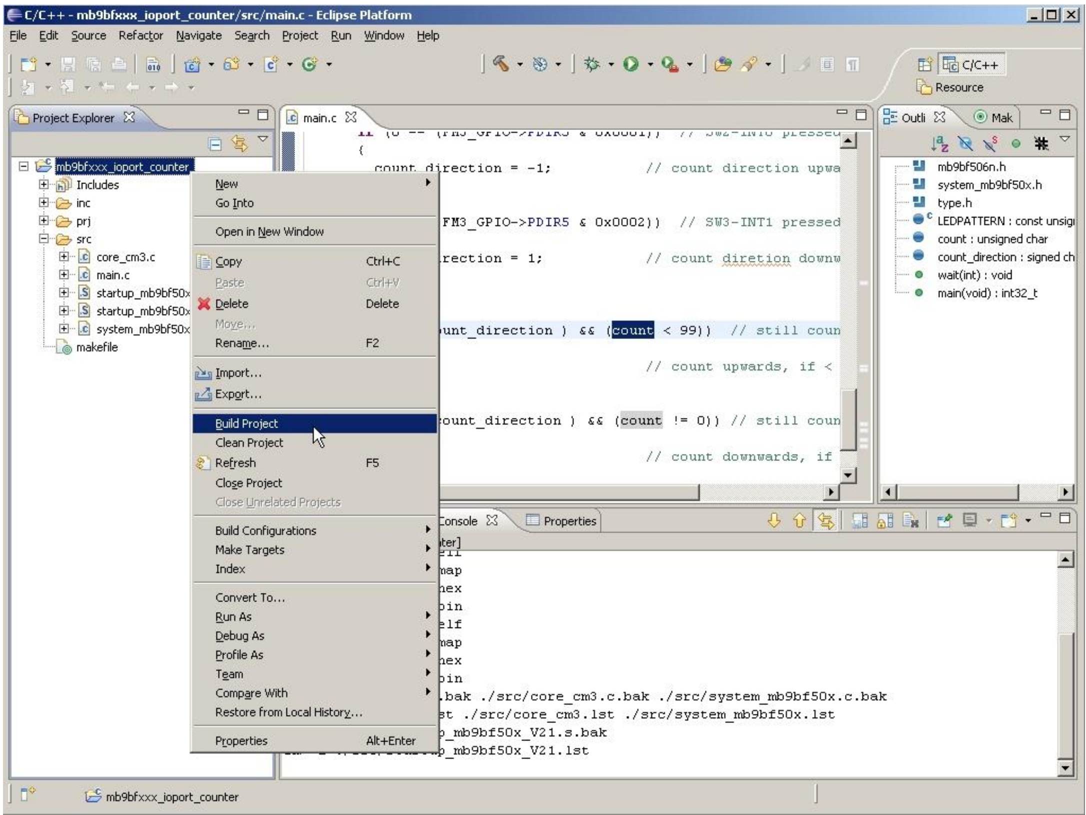
To show the results of the clean process, look at the “Console” panel located below.



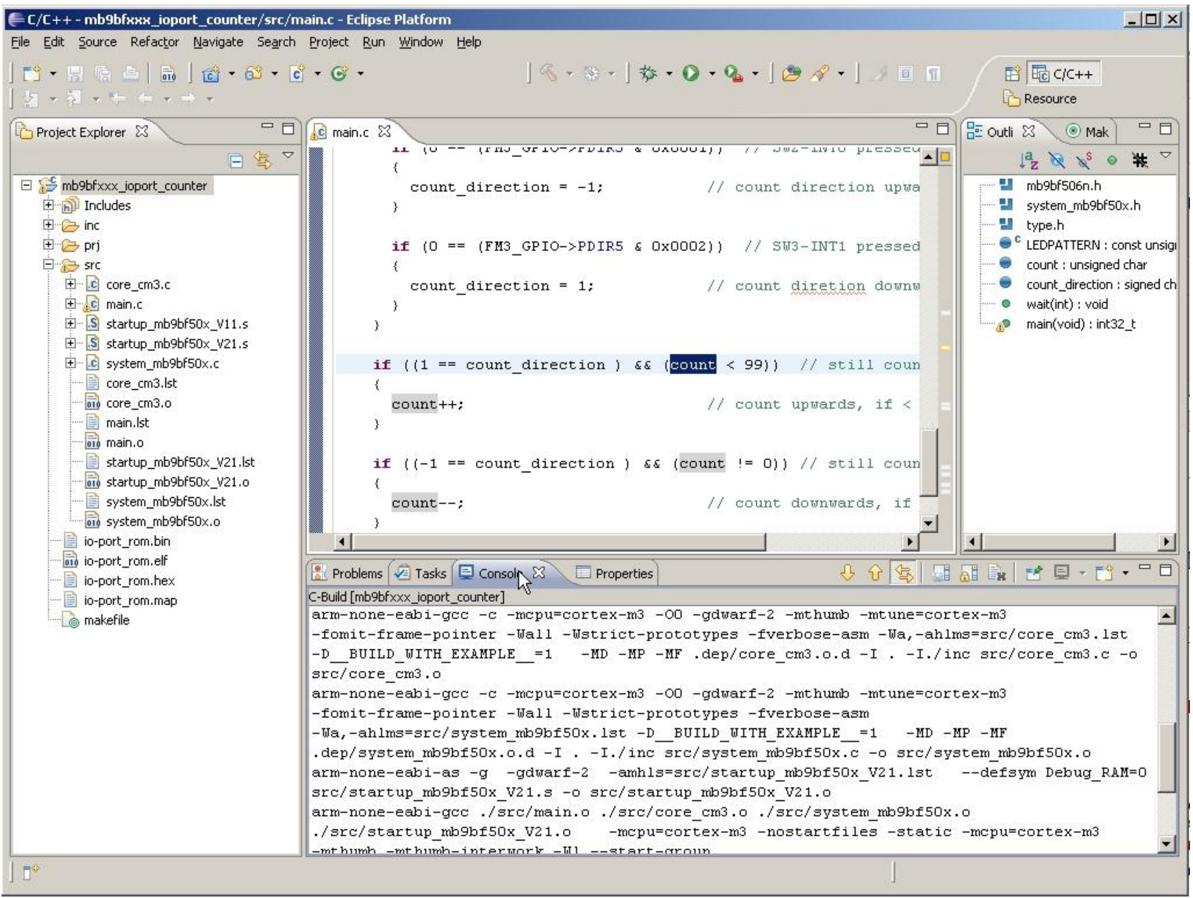
8.4 Building the selected project

Important note: If you use the makefile of the software package of this application note, check all paths (e.g. to *OpenOCD*) and modify them to your individual installation paths!

The project *mb9bfxxx_ioport_counter* can be compiled with the preinstalled Yagarto tool chain. To start this procedure, select the project *mb9bfxxx_ioport_counter* on the "Project Explorer" view. With a click on the right mouse button on the selected project start the build process with *Build project*.



The result will be than show on the IDE "Console" like below.



On the "Project Explorer" view, it can be seen that the project output files (*.bin*, *.elf* ...) are generated.

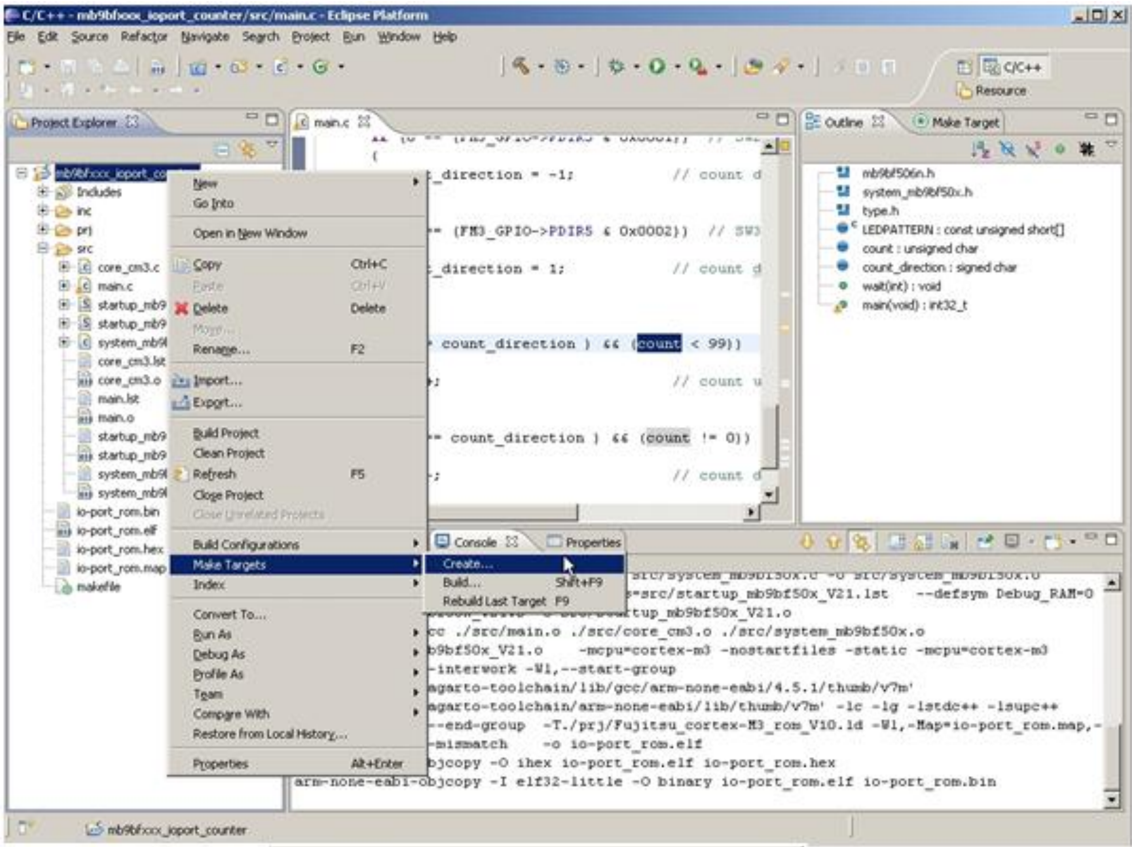
8.5 Create make target

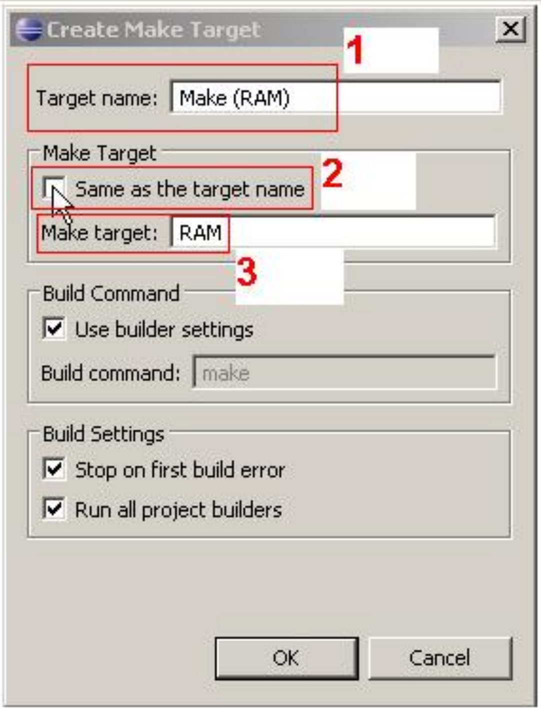
The make targets are pre-defined in the example project *mb9bfxxx_ioport_counter*. This paragraph shows the creation process, if a new project is set-up or the targets were deleted accidentally.

The make file for the project *mb9bfxxx_ioport_counter* manages the project build process. This file generates output files for debugging in RAM and ROM. The make file generates also the final output file for programming the Flash with an external tool like the Spansion Flash Programmer.

It is needed to create a make target to separate the build processes for RAM and ROM (Flash). Also add the clean process to “Make Target”.

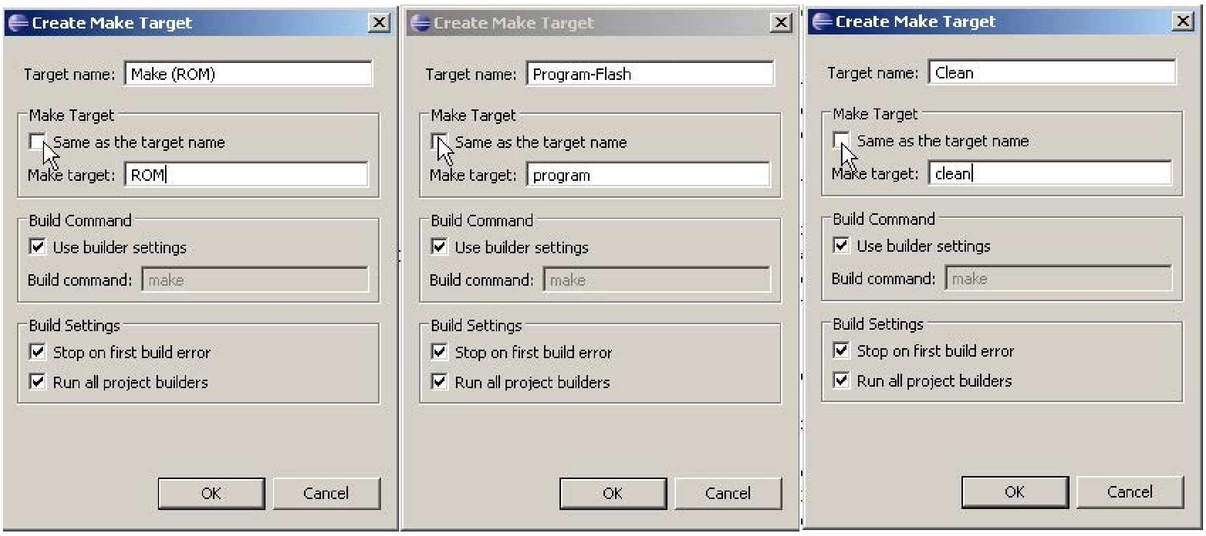
To create a make target, select the project *mb9bfxxx_ioport_counter* on the “Project Explorer” view. Click with the right mouse button on the selected project and select *Make Targets*.



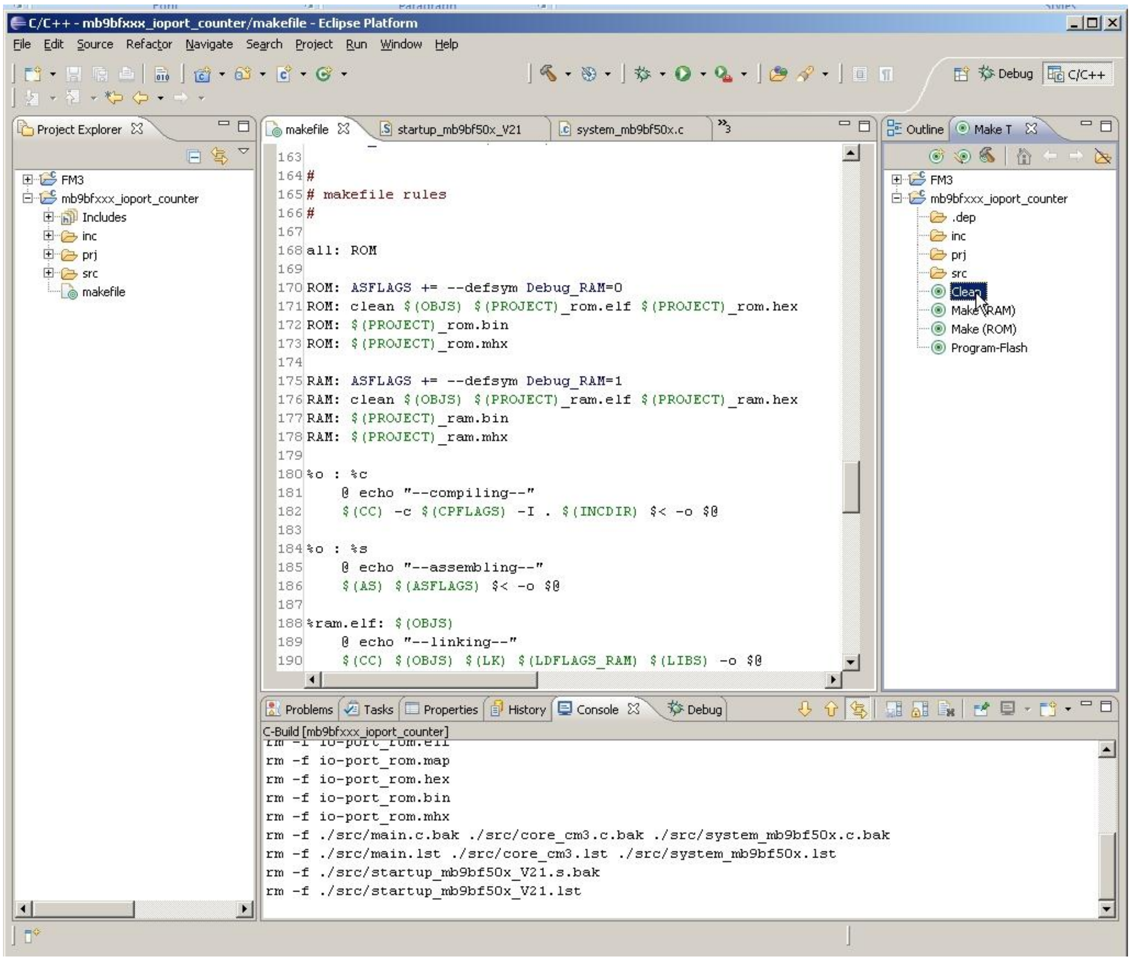


Enter “Make (RAM)” for the target name, uncheck *Same as the target name* and write “RAM” in the text box “Make target”. Click on *OK* to create a “Make (RAM)” build target.

On the same way, create a make target for “Make (ROM)”, “Program-Flash” and “Clean”.



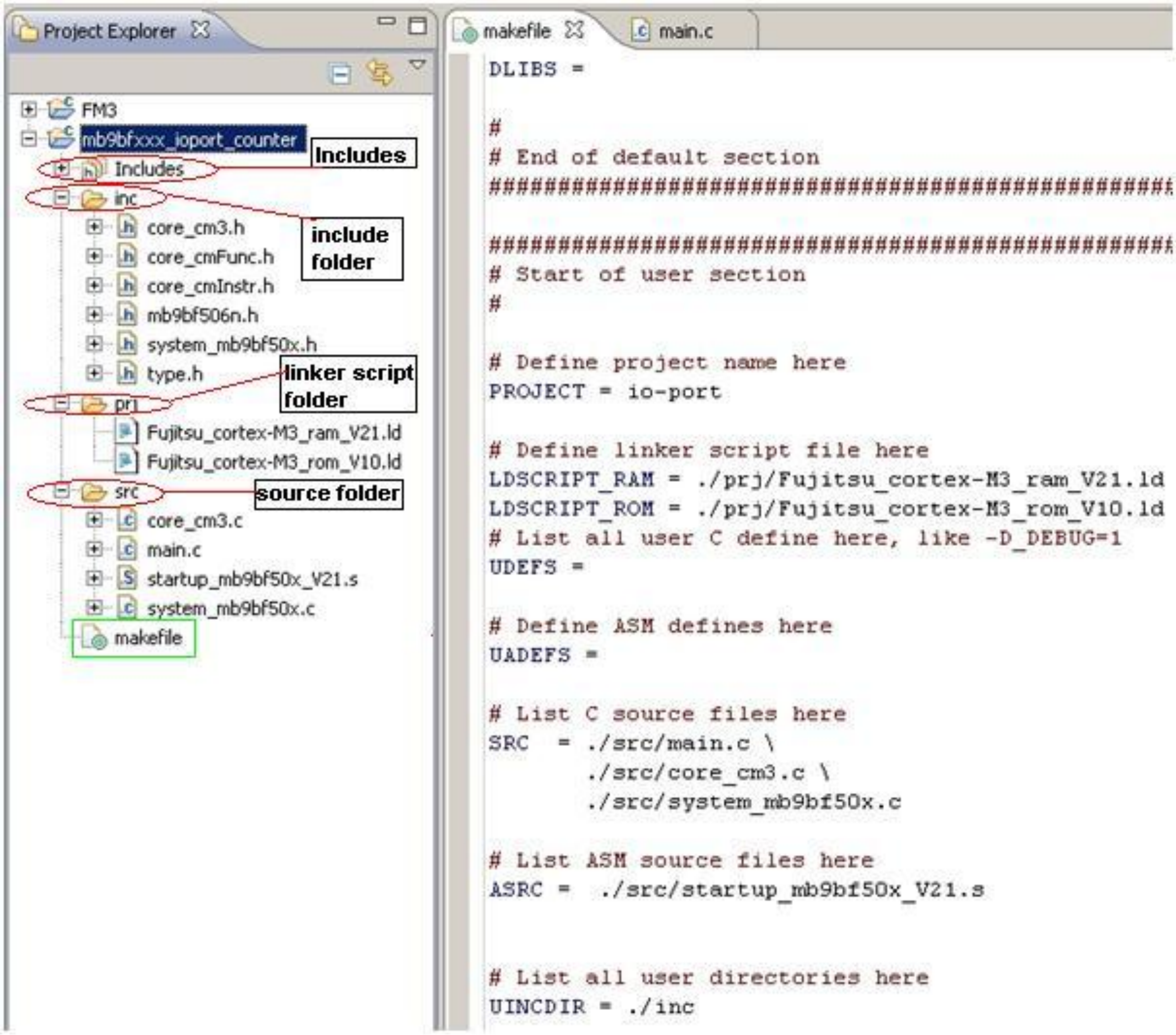
On the next figure the “Make Target” view can be seen. To start the build process for “Make (RAM)”, “Make (ROM)” or “Clean”, simply double click on the respective target.



On the IDE “Console” view, the output shows that the clean process was successfully done.

9 Example Eclipse Project Template

The project template used in this application note has the following structure:



The ***inc*** folder consists of the FM3 I/O header file used with all projects. Also the CMSIS header files and system start-up header are included here. The ***prj*** folder contains the linker script files and in ***src*** are located the source files.

The ***makefile*** is also included to the template.

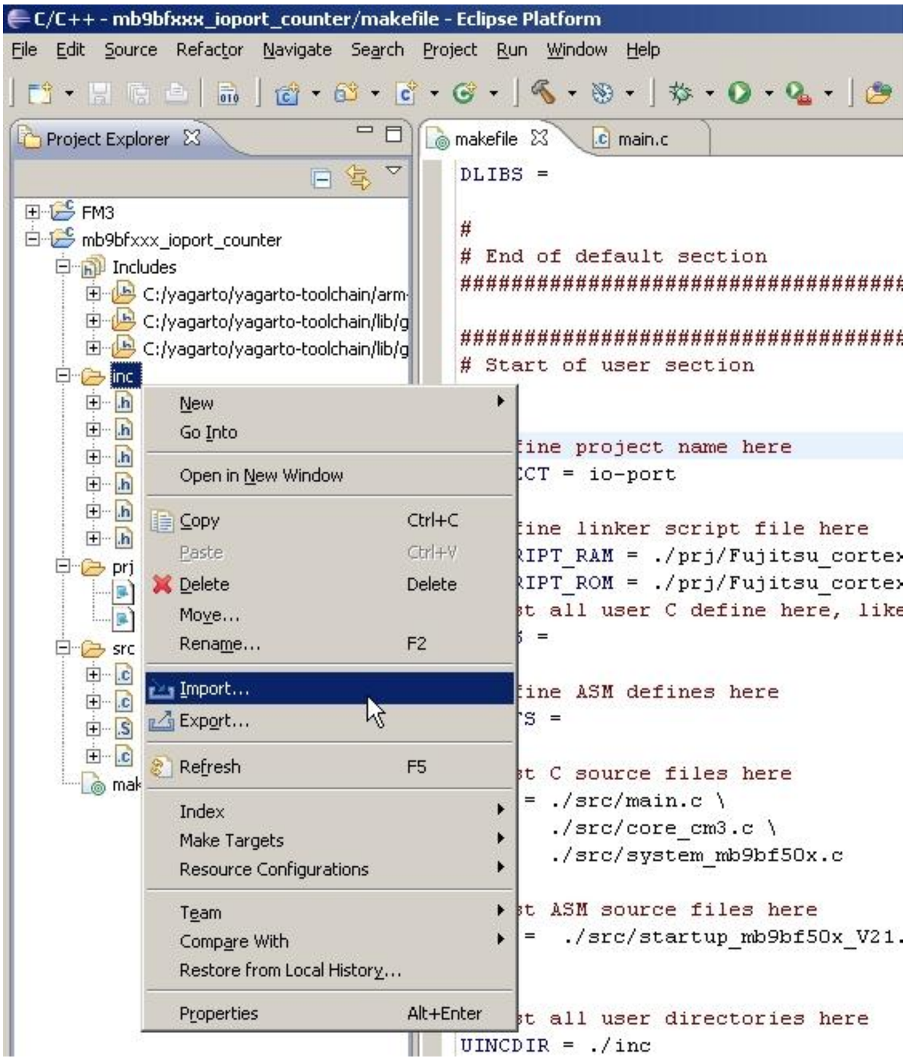
The ***Includes*** directory contains the Yagarto libraries (e.g. *stdint.h*) needed during the build process. To add other sources file use the folder ***src***.

New header files can be added to the folder ***inc*** or to the ***Includes*** directory.

Important note: Check all paths (e.g. to *OpenOCD*) in the makefile(s) and modify them to your individual installation paths!

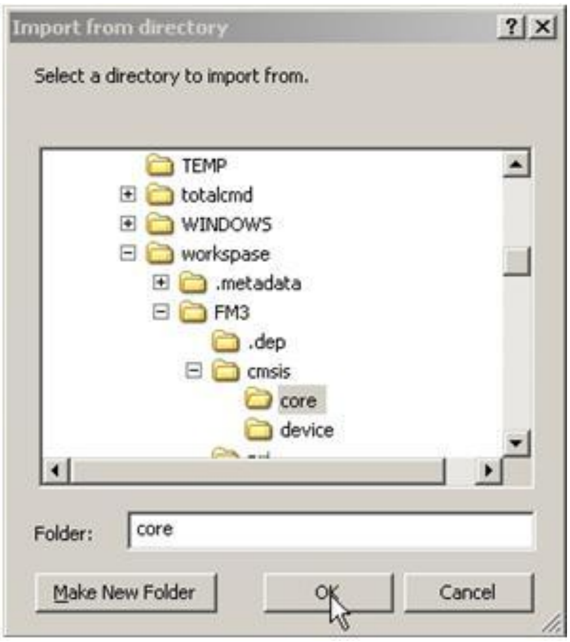
9.1 Add other Files to the Template Folder

Open the selected project folder, where new files should be added. Click with the right mouse key on the selected folder and use *Import*.

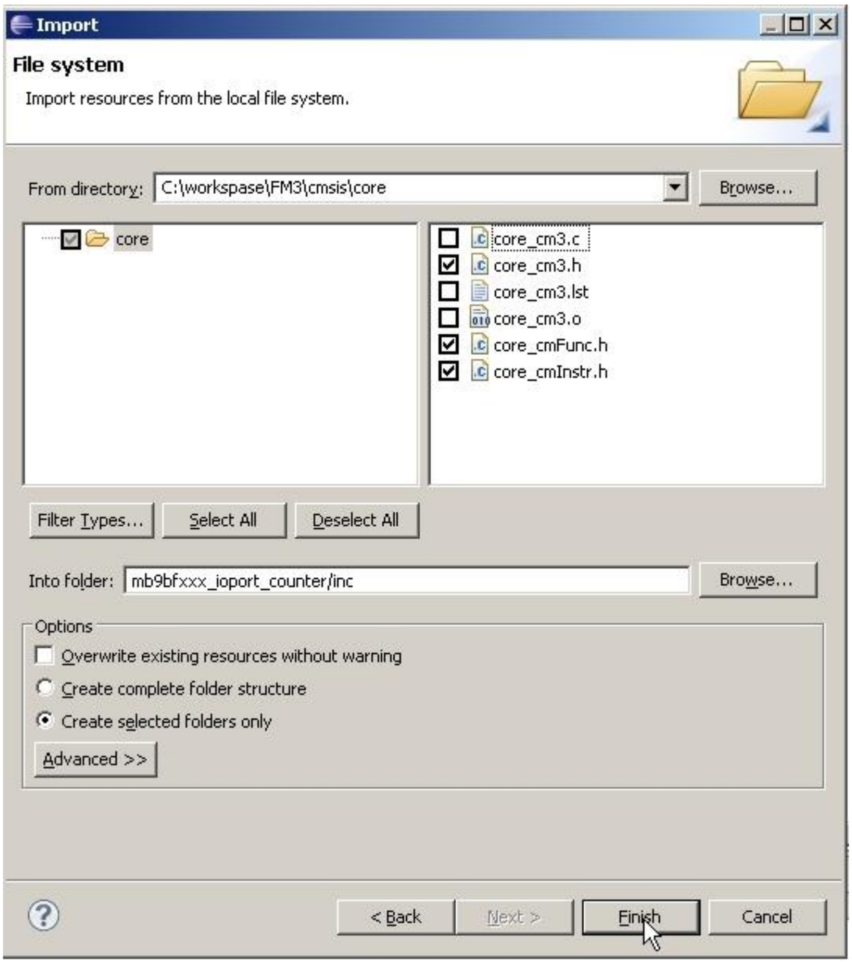


Select *File System* and click on the *Next* button.

Click on the *Browse* button to locate the new files.



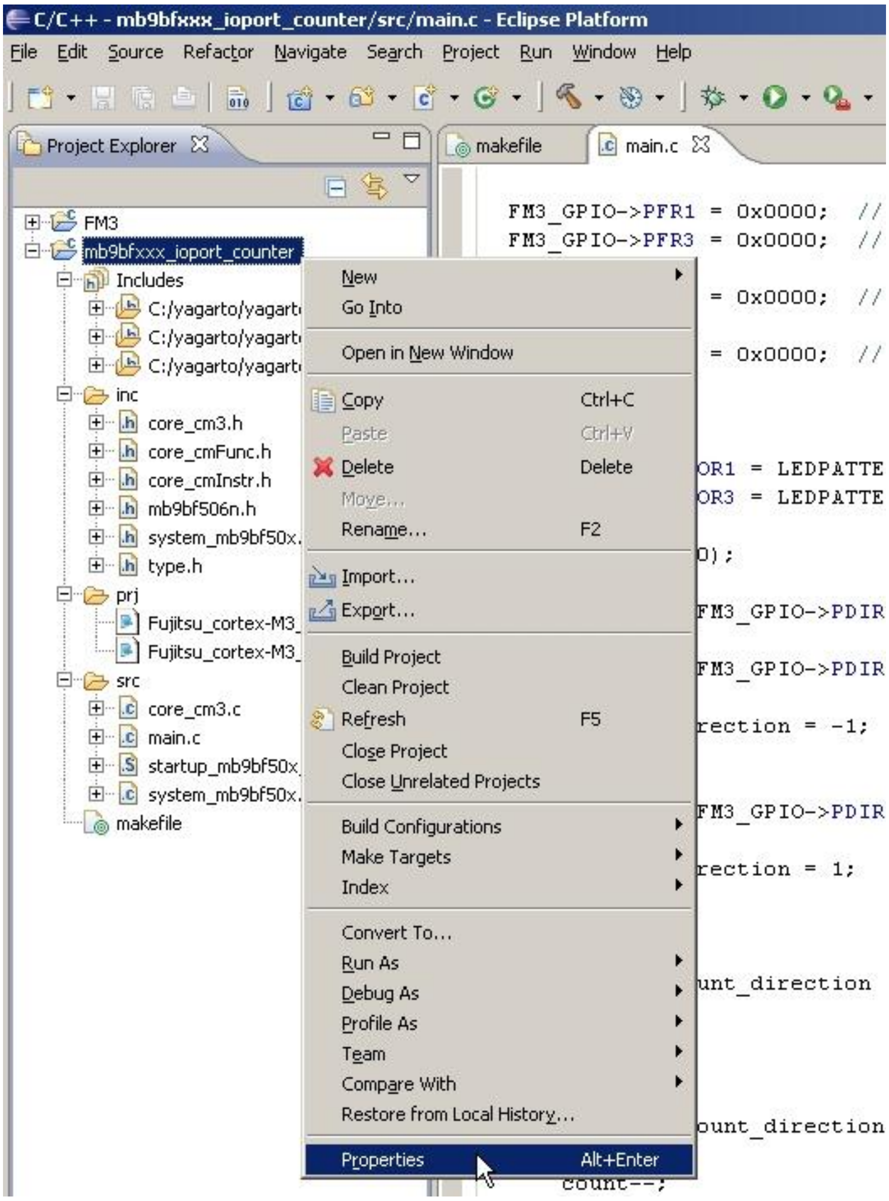
After selecting the folder, check the files which should be imported inthe list.



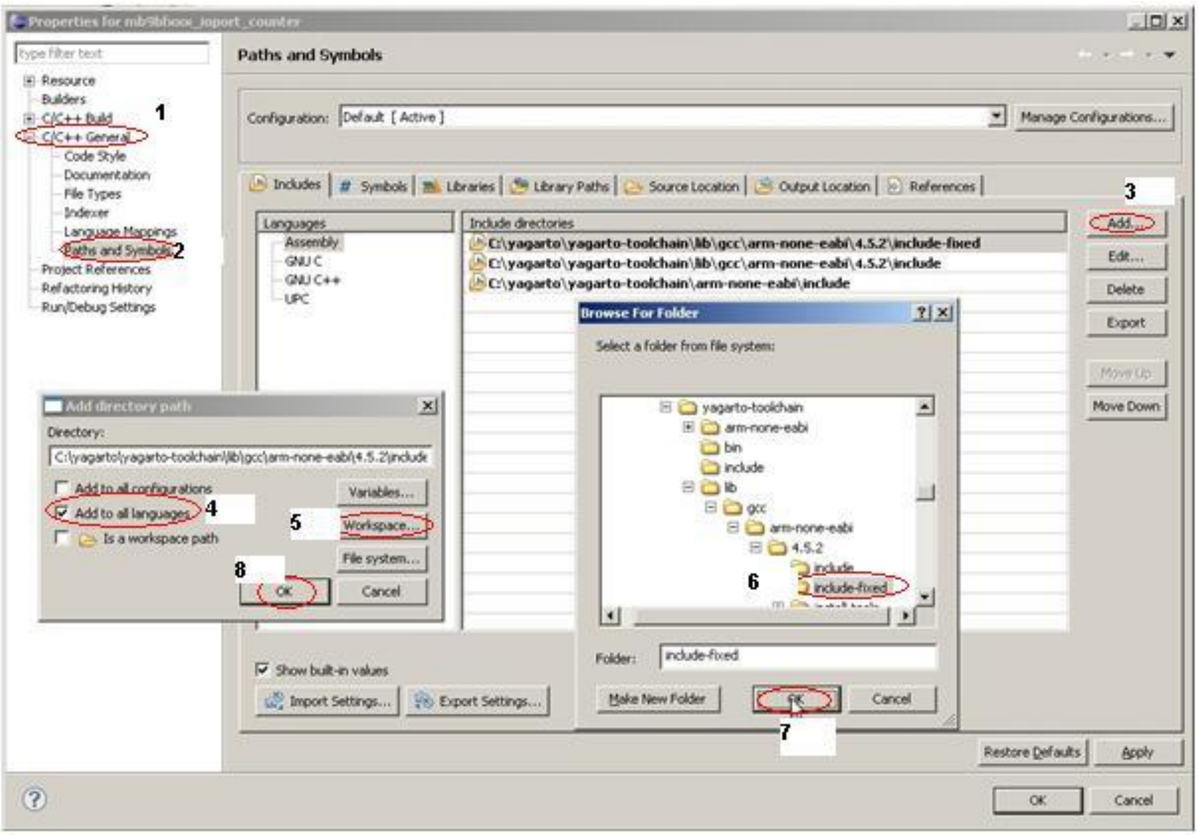
With a click to *Finish*, the selected header files are added to the folder *inc*.

9.2 Add other Libraries to the "Includes" Directory

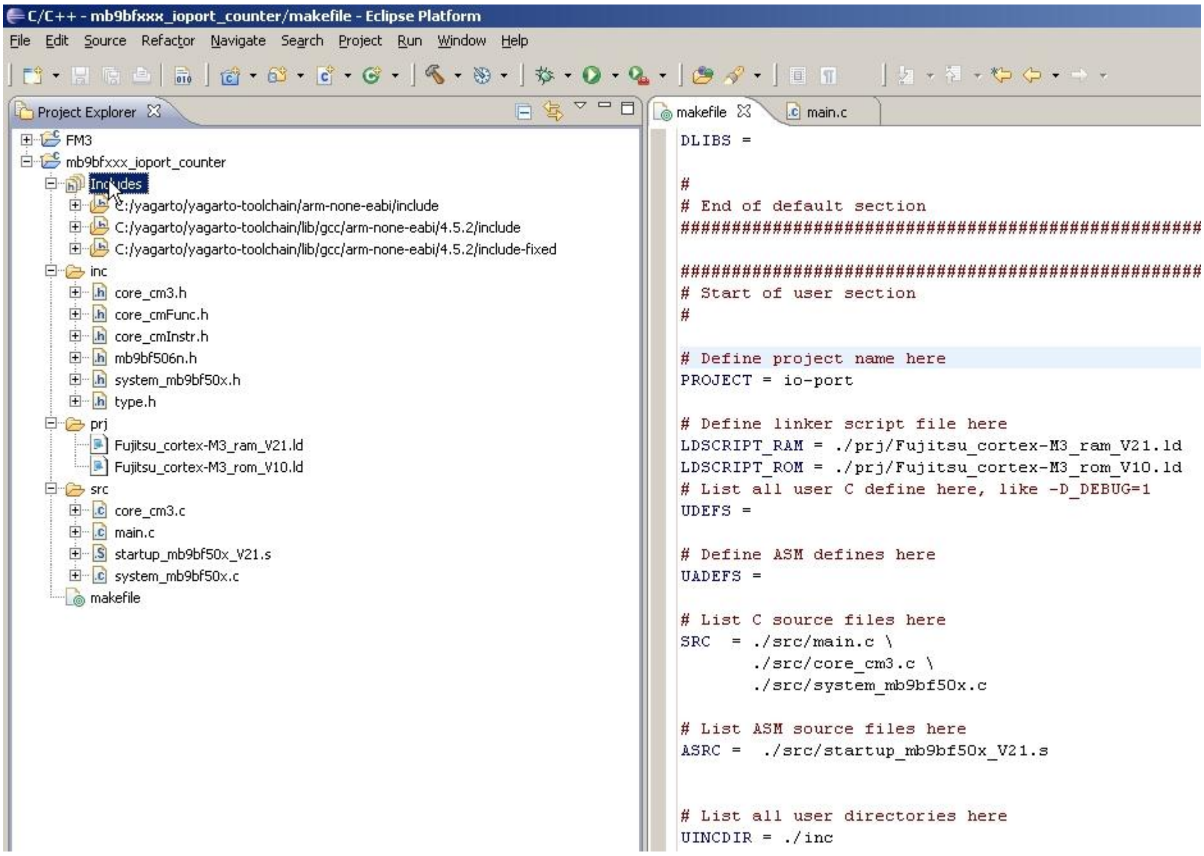
Some library headers (e.g. "*stdint.h*") must be included explicitly from the Yagarto installation directory. To set the *Includes* directory in your template or to add new libraries in this directory, select the project and click with the right mouse key to *Proprieties*. Here changes to the configuration options for the selected project can be done.



1. Select *C/C++ General*
2. Double click on *Paths and Symbols*
3. Click on *Add*
4. Enable the box *Add to all languages*
5. Select *File system* to locate the include directory
6. Select the include directory
7. Click on *OK* in the “browser” child window
8. Click on *OK* in the “Add directory path” child window



The new libraries folder is newly added to the *Includes* directory.



9.3 Make File

The make file is composed of many instructions to the GNU make tool. These instructions are used to set the information needed by the make builder and to initiate the project build process. It can be found in the application note's software package archive.

The make file instructions are described below in detail. The make file is divided here into many parts to get a better overview about the meaning of these instructions.

In the first part of the make file the GNU tools needed to compile (*arm-none-eabi-gcc.exe*), assemble (*arm-none-eabi-as.exe*) and link (*arm-none-eabi-ld.exe*) the project are set. The files created by compiling and assembling are so-called object files (*.o*). In addition to the GNU compiler and assembler, it is needed to set the GNU tool (*arm-none-eabi-objcopy.exe*) to create out of the output file (*.elf*), generated by the linker, another formats, e.g. hex file (*.hex*) or binary file (*.bin*).

```
TRGT = arm-none-eabi-
CC   = $(TRGT)gcc
AS   = $(TRGT)as
LD   = $(TRGT)ld -v
CP   = $(TRGT)objcopy
```

It is here considered that all needed GNU tools are installed and added to Windows path by the Yagarto installation procedure described in the chapter2. These tools can be found on the folder *bin* of the Yagarto GNU ARM tool chain installation directory.

Next statements on the make file are the options needed for the GNU *Objcopy* tool to create other format from the GNU linker generated output file (*.elf*).

The first line is to create the Intel-format hex file (*.hex*). The second one is to generate the binary file (*.bin*) and the last one for the Motorola S-record hex format (*.mhx*).

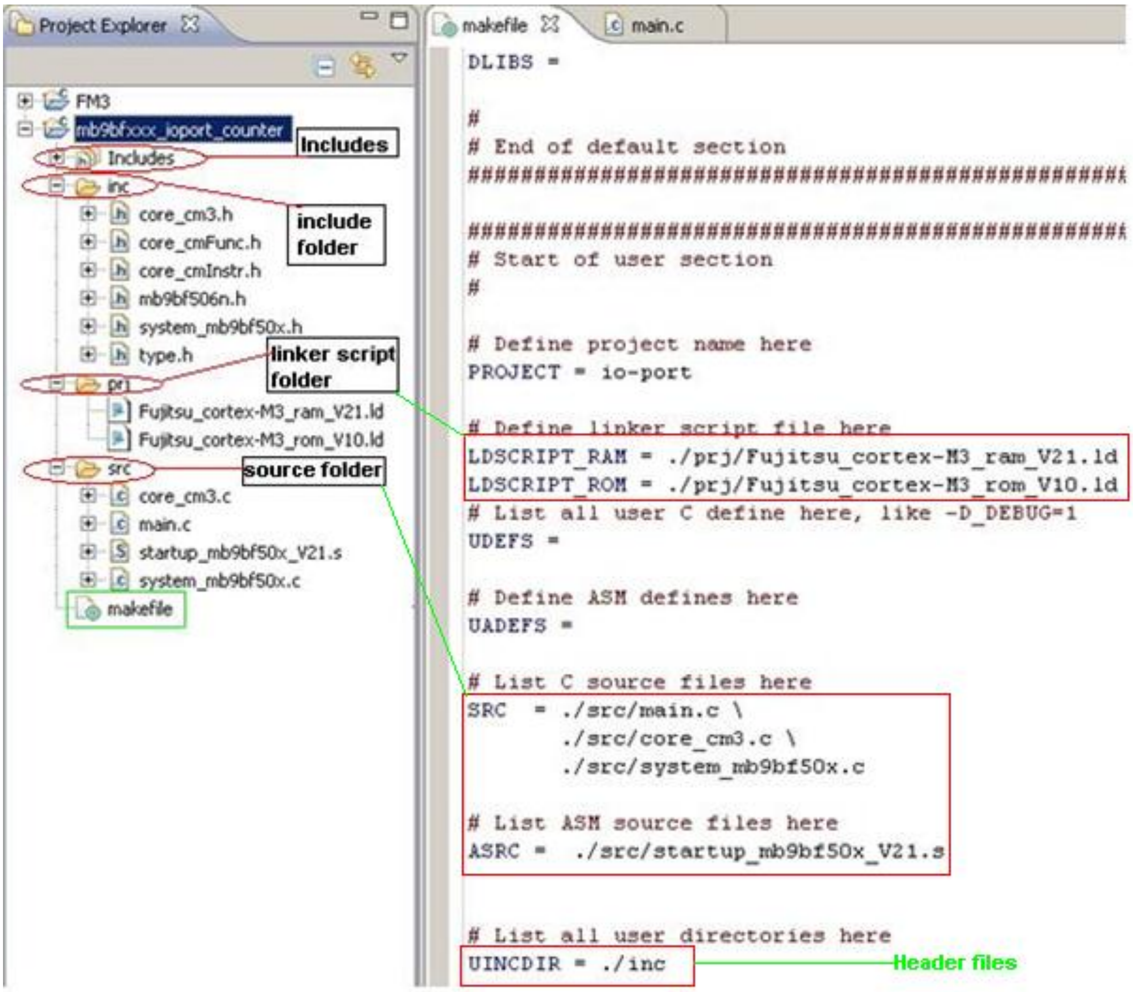
```
HEX  = $(CP) -O ihex
BIN  = $(CP) -I elf32-little -O binary
SREC = $(CP) -O srec
```

The next lines define the over-all project name. This name will then be used to the for the output file generated by the GNU linker and copied to other format by the GNU *Objcopy* tool.

```
# Define project name here
PROJECT = io-port
```

The example Eclipse project template consists of the following project folder:

- ***inc***: includes all the header files
- ***prj***: includes all the linker script files
- ***src***: includes all source files (**.c* and **.s*)



This folder structure is defined as follows:

```
# Define linker script file here
LDSCRIPT_RAM = ./prj/MB9BFD18_ram.ld
LDSCRIPT_ROM = ./prj/MB9BFD18_rom.ld

# List C source files here
SRC  = ./src/main.c ¥
        ./src/core_cm3.c ¥
      ./src/system_mb9bfd1x.c

# List ASM source files here
ASRC = ./src/ startup_mb9bfd1x.s

# List all user directories here
UINCDIR = ./inc
```

The next part isn't used. If the user has the intention to add some defines or library modules, this makefile part can be used.

```
#########################################################################
# Start of default and user defines
#

# List all default C defines here, like -D_DEBUG=1
DDEFS =

# List all default ASM defines here, like -D_DEBUG=1
DADEFS =

# List all default directories to look for include files here
DINCDIR =

# List the default directory to look for the libraries here
DLIBDIR =

# List all default libraries here
DLIBS =

# List all user C define here, like -D_DEBUG=1
UDEFS =

# Define all user ASM defines here
UADEFS =

# List the user directory to look for the libraries here
ULIBDIR =

# List all user libraries here
ULIBS =

#
# End of default and user defines
#########################################################################
```

The added defines and locations, where the included header files and the used library modules are located, are provided in the next makefile part to the compiler, assembler and linker as options used by building the project.

- `INCDIR`: Compiler directories options, e.g. the C-headers are in "UINCDIR=./**inc**"
- `LIBDIR`: Linker libraries directories options
- `DEFS`: Compiler defines options
- `ADEFS`: Assembler defines options
- `LIBS`: Linker libraries options

This part does not need to be changed. All definitions are set in the previously makefile part (default and user defines).

```
INCDIR  = $(patsubst %,-I%,$(DINCDIR) $(UINCDIR))
LIBDIR  = $(patsubst %,-L%,$(DLIBDIR) $(ULIBDIR))
DEFS    = $(DDEFS) $(UDEFS)
ADEFS   = $(DADEFS) $(UADEFS)
LIBS    = $(DLIBS) $(ULIBS)
```

The next lines determine the object files, which will be created by compiling and assembling the project; from all C and assembler (*.s) files located in "**src**" folder are object files (*.o) generated.

```
OBJS    = $(SRC:.c=.o) $(ASRC:.s=.o)
```

Next the compiler optimization level option is set.

```
# Define optimization level here
OPT = -O0
```

The following instructions specify the name of the target ARM processor (`cortex-m3`). The compiler and assembler uses this option to determine what instruction set to be used, when generating the assembly code.

```
MCU     = cortex-m3
MCFLAGS = -mcpu=$(MCU)
```

All options used by the GNU Compiler are started in the next part.

```
CPFLAGS                    = $(MCFLAGS)
CPFLAGS                    += $(OPT)
CPFLAGS                    += -gdwarf-2
CPFLAGS                    += -mthumb
CPFLAGS                    += -mapcs-frame
CPFLAGS                    += -msoft-float
CPFLAGS                    += -mno-sched-prolog
CPFLAGS                    += -fno-hosted
CPFLAGS                    += -mtune=cortex-m3
CPFLAGS                    += -mfix-cortex-m3-ldrd
CPFLAGS                    += -ffunction-sections
CPFLAGS                    += -fdata-sections
CPFLAGS                    += -fomit-frame-pointer
CPFLAGS                    += -Wall
CPFLAGS                    += -Wstrict-prototypes
CPFLAGS                    += -fverbose-asm
CPFLAGS                    += -Wa,-ahlms=$(<:.c=.lst)
CPFLAGS                    += $(DEFS)
```

To generate dependency information between the C sources files and the header files included in this source files, a compiler flag to generate these information is enabled. The generating information will then be deleted by cleaning the project with *make clean*.

```
# Generate dependency information
CPFLAGS                    += -MD -MP -MF .dep/$(@F).d
```

The following lines are the GNU assembler flags.

```
ASFLAGS              = $(MCFLAGS)
ASFLAGS              += -g
ASFLAGS              += -gdwarf-2
ASFLAGS              += -mthumb
ASFLAGS              += -amhls=$(<:.s=.lst)
ASFLAGS              += $(ADEFS)
```

The next part determines the general linker flags.

```
LK                   =  -static -mcpu=cortex-m3 -mthumb -mthumb-interwork
LK                   += -nostartfiles
LK                   += -Wl,--start-group
LK                   += -lc -lg -lstdc++ -lsupc++
LK                   += -lgcc -lm
LK                   += -Wl,--end-group
```

Because this makefile manages the building process to generate output files (*.elf*) for RAM and ROM debugging, a linker script file for each debugging configuration must be set individually.

1. Set the RAM linker script file *Fujitsu_cortex-M3_ram_V21.ld* located in *prj* folder and provided with the makefile instruction `LDSCRIPT_RAM`
2. Generate a map file (*.map*)
3. Provide the library directories, if they are set in the defines part

```
LDFLAGS_RAM = -T$(LDSCRIPT_RAM)
LDFLAGS_RAM += -Wl,-Map=$(PROJECT)_ram.map,--cref,--no-warn-mismatch
LDFLAGS_RAM += $(LIBDIR)
```

The next instructions set **ROM** linker flags:

1. Set the ROM linker script file *Fujitsu_cortex-M3_rom_V10.ld* located in *prj* folder and provided with the makefile instruction `LDSCRIPT_ROM`
2. Generate a map file (*.map*)
3. Provide the library directories, if they are set in the defines part

```
LDFLAGS_ROM = -T$(LDSCRIPT_ROM)
LDFLAGS_ROM += -Wl,-Map=$(PROJECT)_rom.map,--cref,--no-warn-mismatch
LDFLAGS_ROM += $(LIBDIR)
```

In the next part follow the make rules to create a **RAM** target. By building the RAM target, all object files (*.o*) and output files (*.elf, *.bin, *.hex, *.mhx*) will be created.

1. The first definition flag is dedicated to the assembler to set the variable `Debug_RAM` to `1`. This variable is implemented in the "if case" at the *startup_mb9bfd1x.s* file to differentiate between the RAM and ROM initialization routine.
2. A target clean is made before starting building the object files (`$(OBJS)`)
3. Starting building the output file (*.elf*)
4. Starting building the output file (*.hex*)
5. Starting building the output file (*.bin*)
6. Starting building the output file (*.mhx*)

```
RAM: ASFLAGS += --defsym Debug_RAM=1
RAM: clean $(OBJS) $(PROJECT)_ram.elf $(PROJECT)_ram.hex
RAM: $(PROJECT)_ram.bin
RAM: $(PROJECT)_ram.mhx
```

Here the **ROM** target definition is described. The ROM target is defined as default make target. By giving *make all* the building process for ROM target will be started.

```
all: ROM
```

To the `Debug_RAM` variable is now set to `0`. Other instruction lines are similar to the RAM target, only the output files are ROM based (*_rom.elf, *_rom.hex*, etc.).

```
ROM: ASFLAGS += --defsym Debug_RAM=0
ROM: clean $(OBJS) $(PROJECT)_rom.elf $(PROJECT)_rom.hex
ROM: $(PROJECT)_rom.bin
```

By starting the building process the object files (**.o*) will be generated from all source files (**.c* and **.s*).

By compiling the (**.c*) files, the GNU compiler (`CC=arm-none-eabi-gcc.exe`) is called. The flags (`CPFLAGS`) are provided to the compiler and the directory, where the header files are located, is also provided.

```
%o : %c
       @ echo "--compiling--"
       $(CC) -c $(CPFLAGS) -I . $(INCDIR) $< -o $@
```

Next lines are the assembling procedure.

The GNU assembler (`AS=arm-none-eabi-as.exe`) will be started to create the object files. The `ASFLAGS` are the flags which were defined for ROM or RAM building configuration before.

```
%o : %s
       @ echo "--assembling--"
       $(AS) $(ASFLAGS) $< -o $@
```

For the linking procedure the GNU compiler (`CC=arm-none-eabi-gcc.exe`) combines all object files (`$(OBJS)=*.o`) generated by compiling and assembling to an output file (**.elf*).

For the **ROM** target build, the GNU linker uses the options `$(LDFLAGS_ROM)` (`LDFLAGS_ROM = -T$(LDSCRIPT_ROM)`) to identify the ROM linker script file.

```
%rom.elf: $(OBJS)
       @ echo "--linking--"
       $(CC) $(OBJS) $(LK) $(LDFLAGS_ROM) $(LIBS) -o $@
```

For the **RAM** target build, the GNU linker uses the options `$(LDFLAGS_RAM)` (`LDFLAGS_RAM = -T$(LDSCRIPT_RAM)`) to identify the RAM linker script file `LDSCRIPT_RAM = ./`**`prj`**`/MB9BFD18_ram.ld`

```
%ram.elf: $(OBJS)
	@ echo "--linking--"
	$(CC) $(OBJS) $(LK) $(LDFLAGS_RAM) $(LIBS) -o $@
```

In the next part, the output file (**.elf*) will be converted to other formats (**.hex, *.bin, *.mhx*). The GNU utility (`CP=arm-none-eabi-objcopy.exe`) can be used by the building process to generate the respective format.
The GNU *Objcopy* tool is called with the macros `HEX`, `BIN` and `SREC` on begin of this makefile. The *Objcopy* options are also set with these macros.

```
%hex: %elf
	$(HEX) $< $@

%bin: %elf
	$(BIN) $< $@

%mhx: %elf
	$(SREC) $< $@
```

The clean target is managed with the rule `clean`. Assuming the command *make clean* will delete all object files (**.o*), the related file (**.lst*) and the output files (**.elf*, **.hex*, **.bin* and **.mhx*) generated by building the project. The clean rule is also called every time, when a RAM or ROM target will be build.

```
clean:
	-rm -f $(OBJS)
	-rm -f $(PROJECT)_ram.elf
	-rm -f $(PROJECT)_ram.map
	-rm -f $(PROJECT)_ram.hex
	-rm -f $(PROJECT)_ram.bin
	-rm -f $(PROJECT)_ram.mhx
	-rm -f $(PROJECT)_rom.elf
	-rm -f $(PROJECT)_rom.map
	-rm -f $(PROJECT)_rom.hex
	-rm -f $(PROJECT)_rom.bin
	-rm -f $(PROJECT)_rom.mhx
	-rm -f $(SRC:.c=.c.bak)
	-rm -f $(SRC:.c=.lst)
	-rm -f $(ASRC:.s=.s.bak)
	-rm -f $(ASRC:.s=.lst)
```

The next part of the makefile is used to program the internal flash with OpenOCD. This part is also not needed, when the user prefers to download and debug the output file (**.elf*) with J-Link GDB Server.

With the first macro the location where the OpenOCD executable will be found is set.

The second macro will set the OpenOCD server (*openocd.exe*). Because this server needs mandatorily a script configuration, the configuration script (*openocd.cfg*) in the project directory (`./`) may be used.

```
# specify the directory where openocd executable and configuration files reside
OPENOCD_DIR = <HERE YOUR PATH TO OPENOCD>/openocd-0.5.0/src

# specify OpenOCD executable
OPENOCD = $(OPENOCD_DIR)openocd-0.5.0.exe

# specify OpenOCD configuration file (pick the one for your device)
OPENOCD_CFG = -f ./openocd.cfg
```

In the next part follows the OpenOCD commands used to program the flash on the FM3

```
# specify OpenOCD flash programing commandos for FM3
OPENOCD_C += -c init
OPENOCD_C += -c jtag_khz 500
OPENOCD_C += -c reset init
OPENOCD_C += -c verify_ircapture disable
OPENOCD_C += -c halt
OPENOCD_C += -c poll
OPENOCD_C += -c 'FM3 mass_erase 0'
OPENOCD_C += -c 'flash write_image $(PROJECT)_rom.bin 0x0 bin'
OPENOCD_C += -c reset run
OPENOCD_C += -c shutdown
```

The second to last part implements the target rule `program`.

First the server will be started with the assigned configuration script (*openocd.cfg*). After this the server will execute the giving commands. When the programming achieved the server will be shutdown and eclipse console will display the message:

`"Flash Programming Finished."`

```
# program the FM3 internal flash memory
program:
	@echo "Flash Programming with OpenOCD..."

	$(OPENOCD) $(OPENOCD_CFG) $(OPENOCD_C)
	@echo "Flash Programming Finished."
```

10 Programing the Flash memory

10.1 OpenOCD and Flash Programming

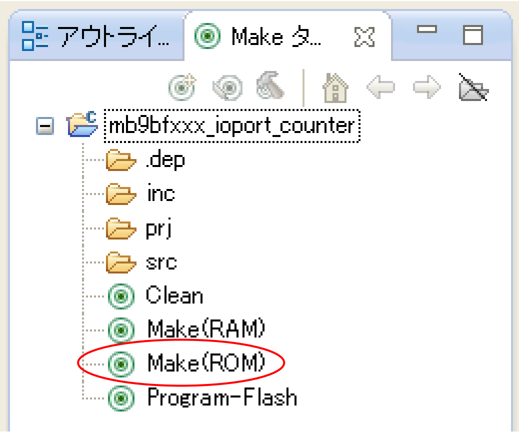
To use OpenOCD for programming the internal Flash memory, a target *Program-Flash* was already created. See chapter 8.5 for usage.

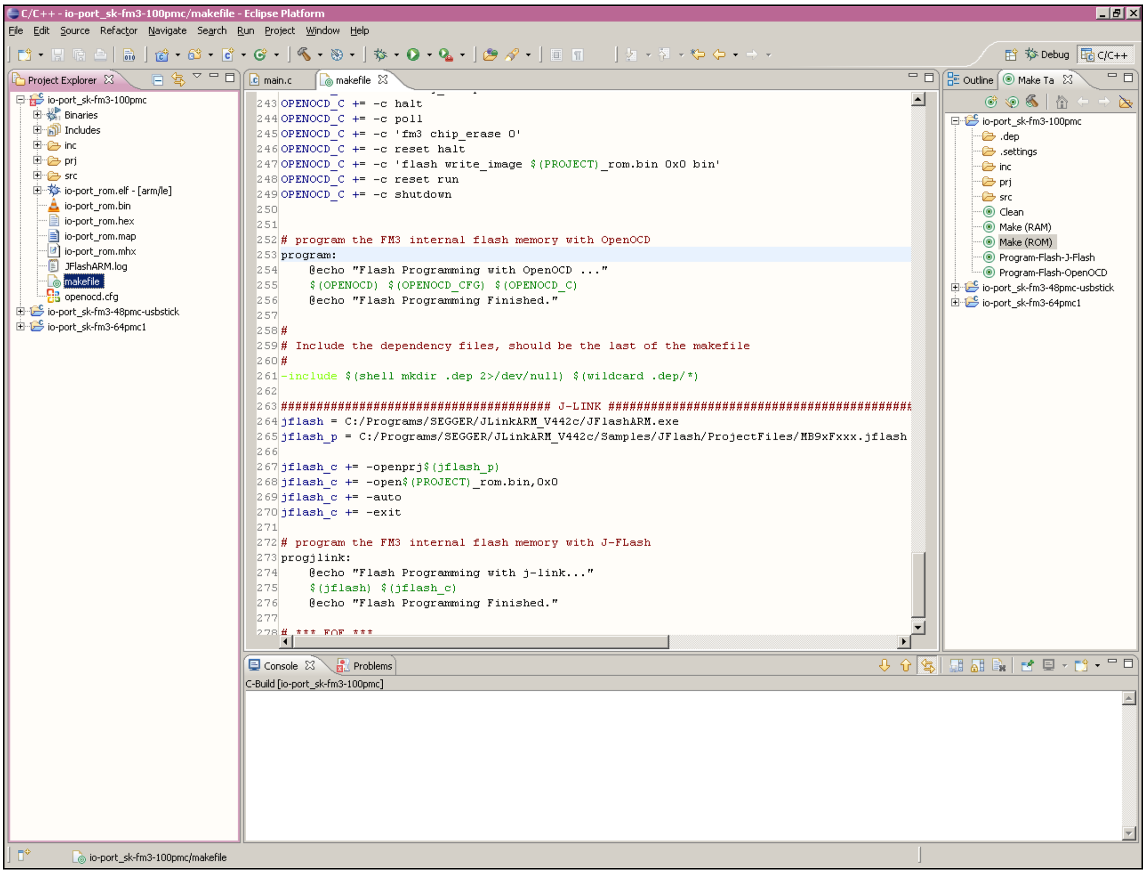
In chapter 9.3 a description of all section used in the makefile was given. The last section implemented in this makefile manages the make target *Program-Flash* used on Eclipse "C/C++ perspective" to program the internal Flash.

Connect the SK-FM3-176PMC-ETHERNET board via JTAG interface to the USB interface of your computer.

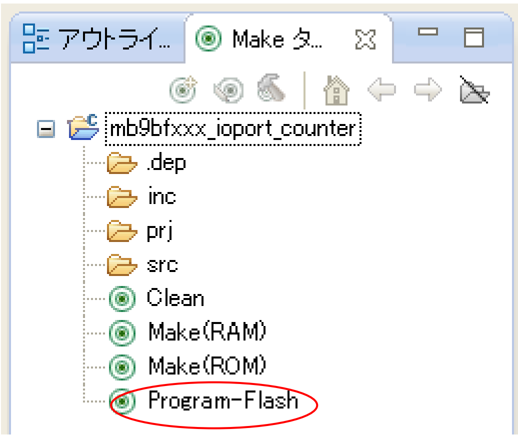
To program the internal Flash, first it is needed to build the target *Make (ROM)*. The binary file *io-port_rom.bin* will be then generated. See chapter 8.5 for usage.

Click on the target *Make (ROM)*.

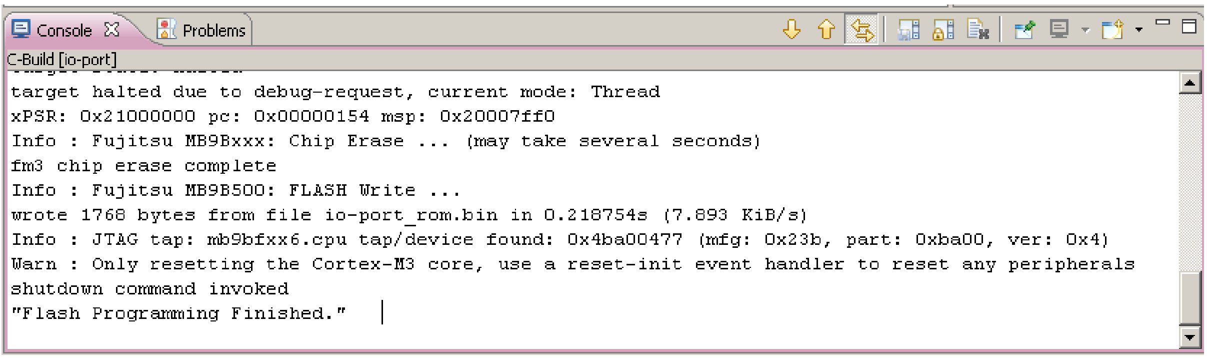




After building the project, the target *Program-Flash* now can be build. Click on it, start the Flash programming with OpenOCD.



The next figure shows the messages displayed on the Eclipse console during the Flash programming realized via OpenOCD.

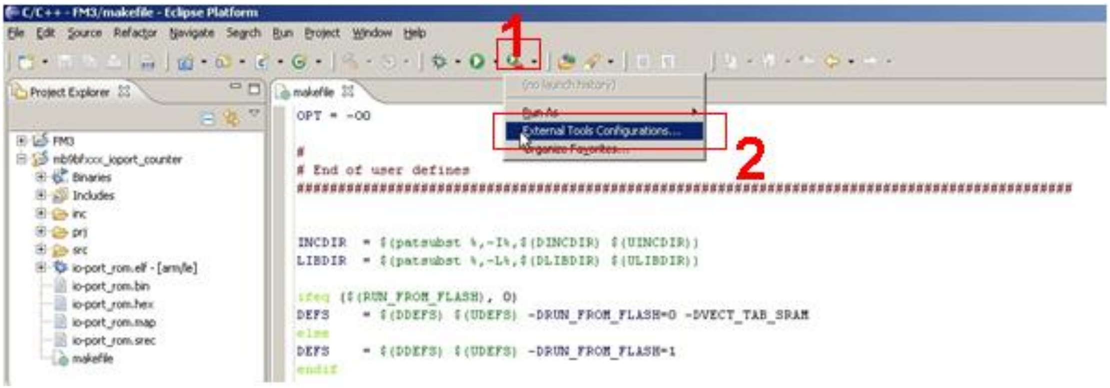


11 Set up Eclipse External Tools

11.1 Further External Tools

Note, that all configurations described below use the paths from the chapter 4. Use *your* individual installation paths instead, when setting up the configurations!

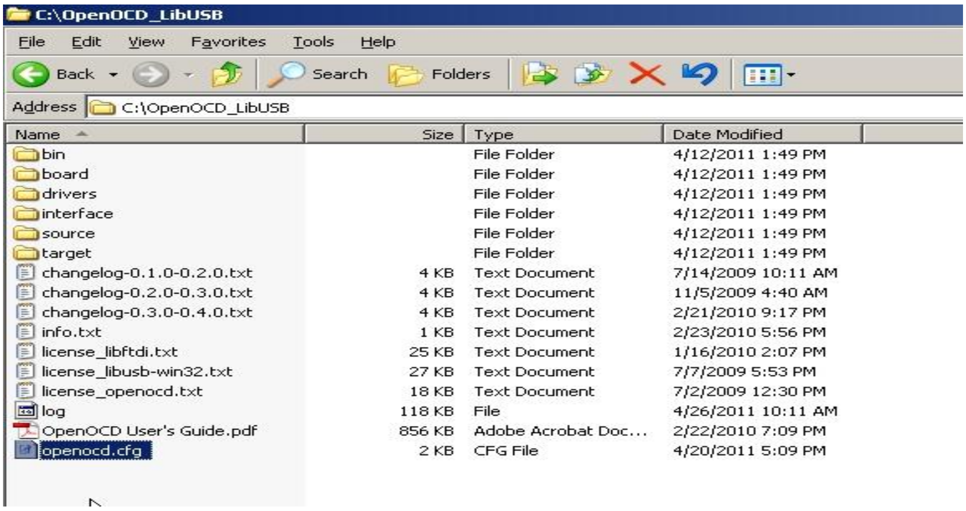
The tools installed by *External Tools Configurations...* menu can be conveniently started from the *Run* pull-down menu or via a toolbar button.



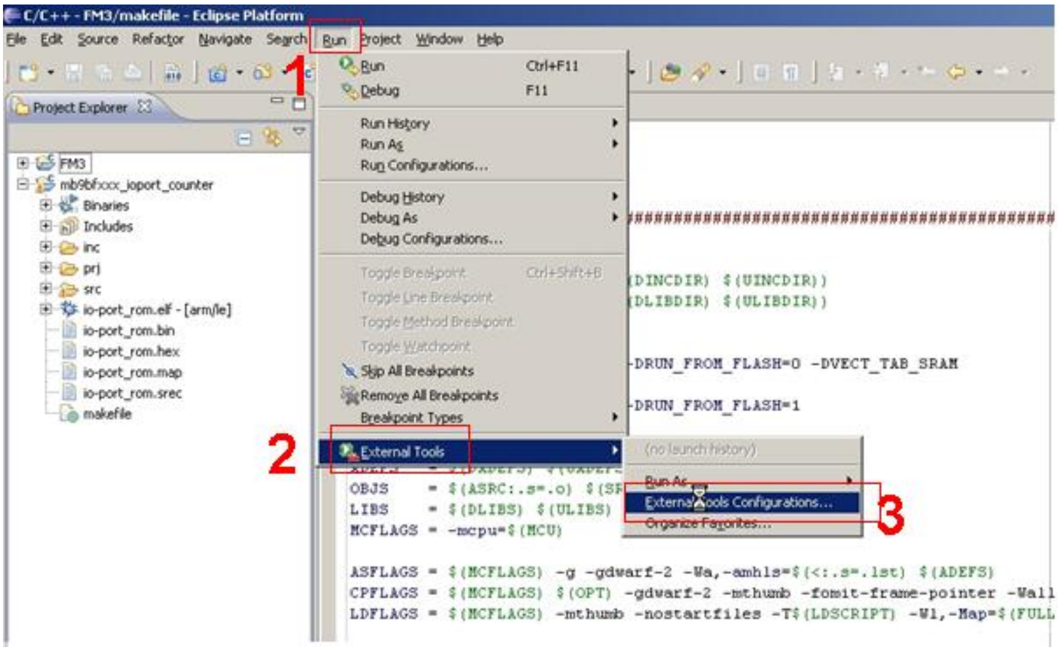
11.2 OpenOCD as an Eclipse external tool

If using J-Link in JTAG interface, OpenOCD must be set as external tool for using J-Link with it.

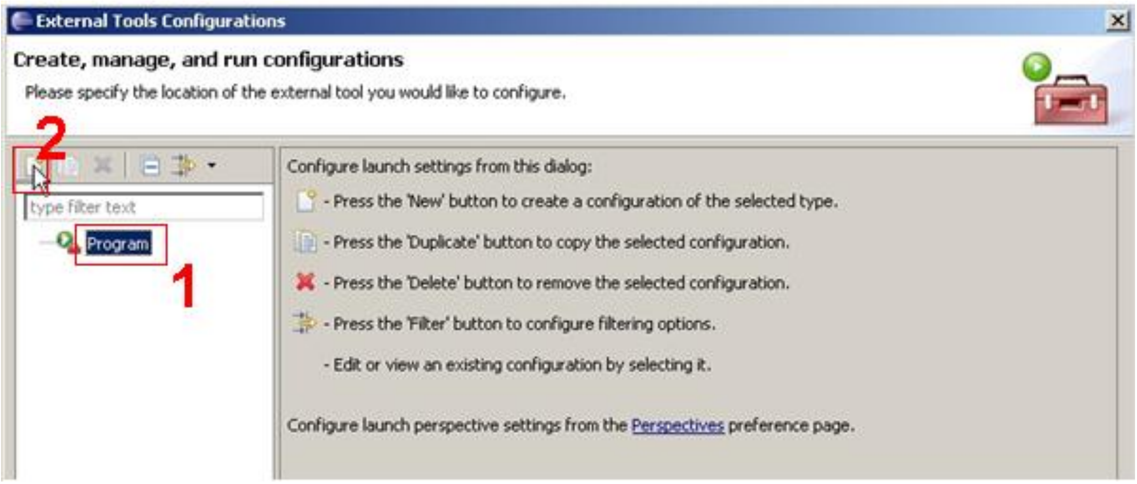
Beforehand, please copy configuration file *openocd.cfg* in the directory ¥*OpneOCD_LibUSB* (*C:¥OpenOCD_LibUSB*).



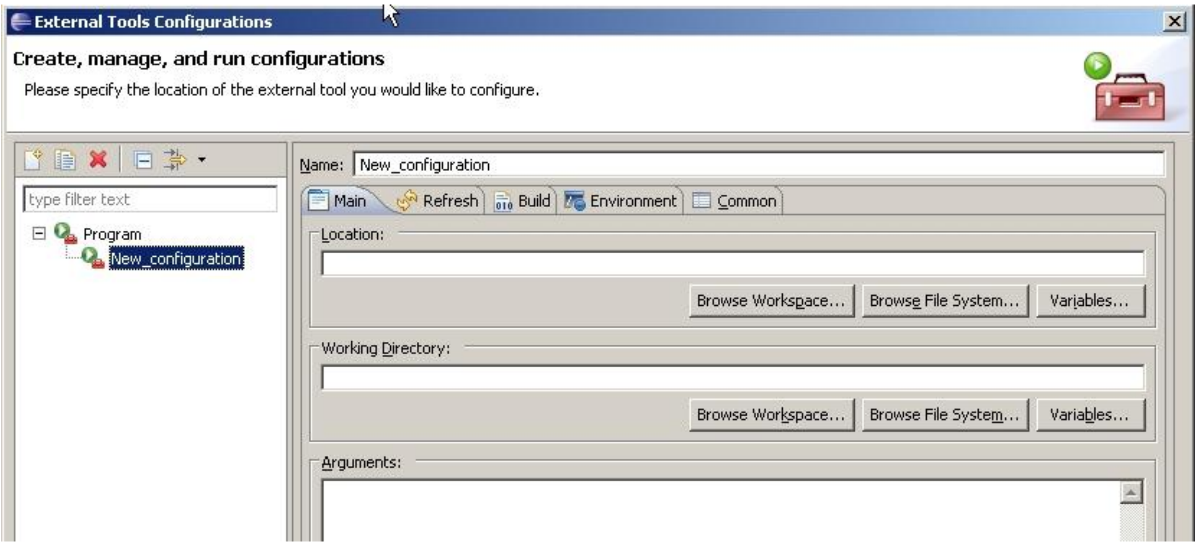
Click on *Run→External Tools→External Tools Configurations...* .



The "External Tools" window will appear. Click on *Program* and then *New* button to establish a new external tool.



Double click *Program*.



Fill out the “External Tools” form exactly as described below.

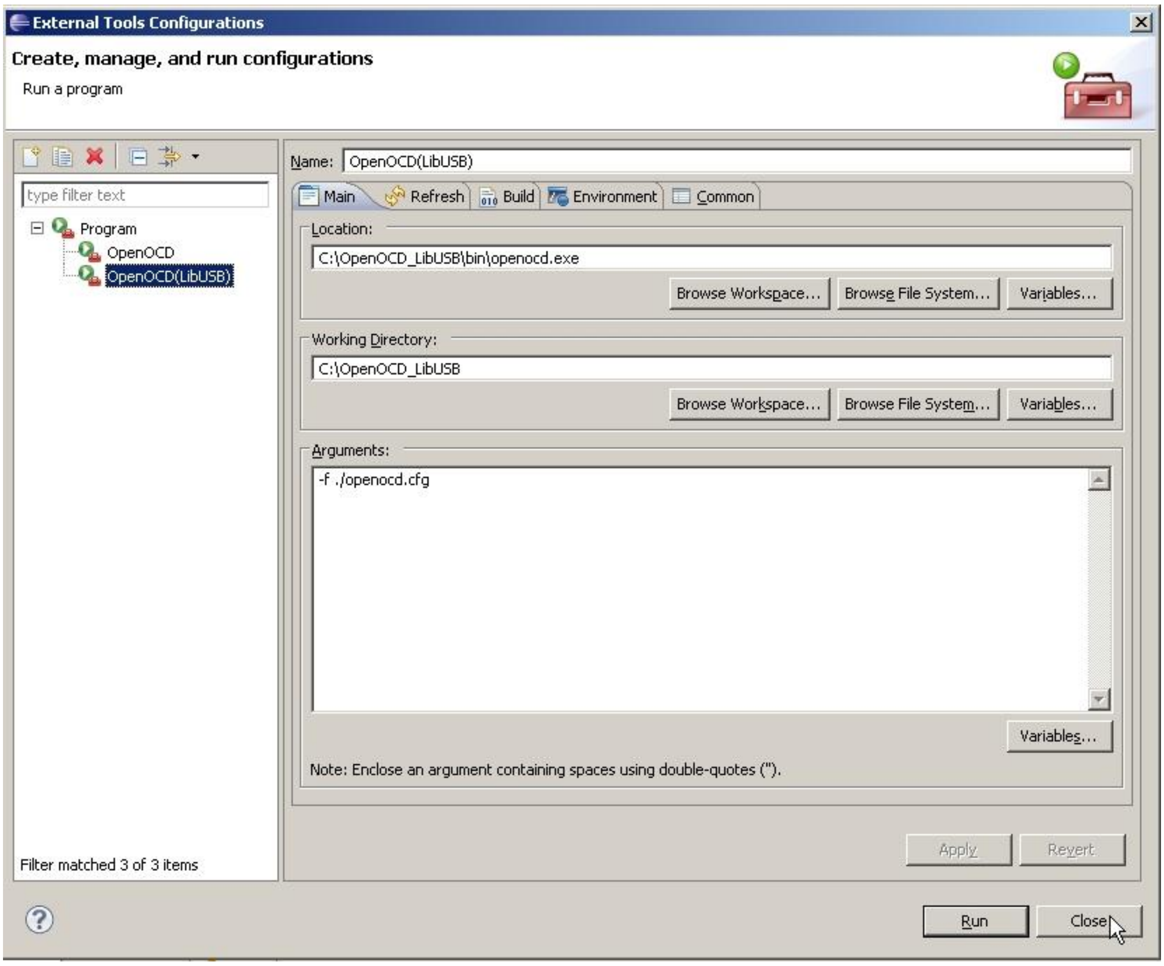
In the “Name” text box call this external tool “OpenOCD”

In the "Location:" pane, use the *Browse File System...* button to search for the OpenOCD executable. It is located in the following folder:

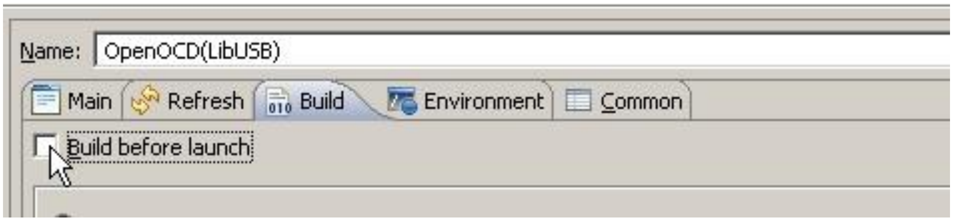
C:¥OpenOCD_LibUSB¥bin¥openocd-0.5.0.exe

In the "Working Directory" pane, use the *Browse File System...* button to specify *C:¥OpenOCD_LibUSB* as the working directory.

In the "Arguments" pane, enter the argument "-f <*your project path*>¥openocd.cfg" to specify the OpenOCD configuration file.



In the *Build* tab uncheck *Build before launch*.



No changes are required to the other tabs in the other forms (*Refresh*, *Environment*, and *Common*).

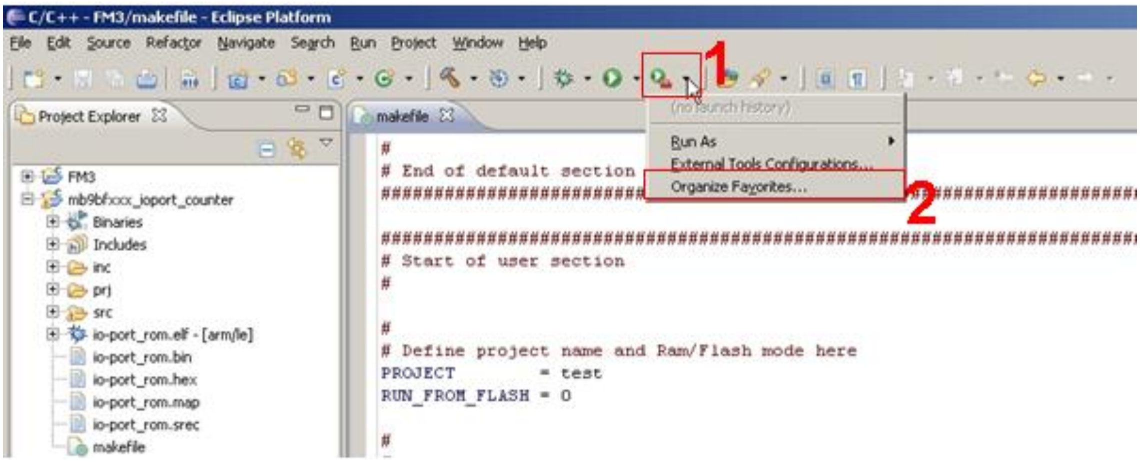
Click on *Apply* and *Close* to register OpenOCD as an external tool.

To check this setup, choose *Run→External Tools→External Tools Configurations...* then select *OpenOCD*.

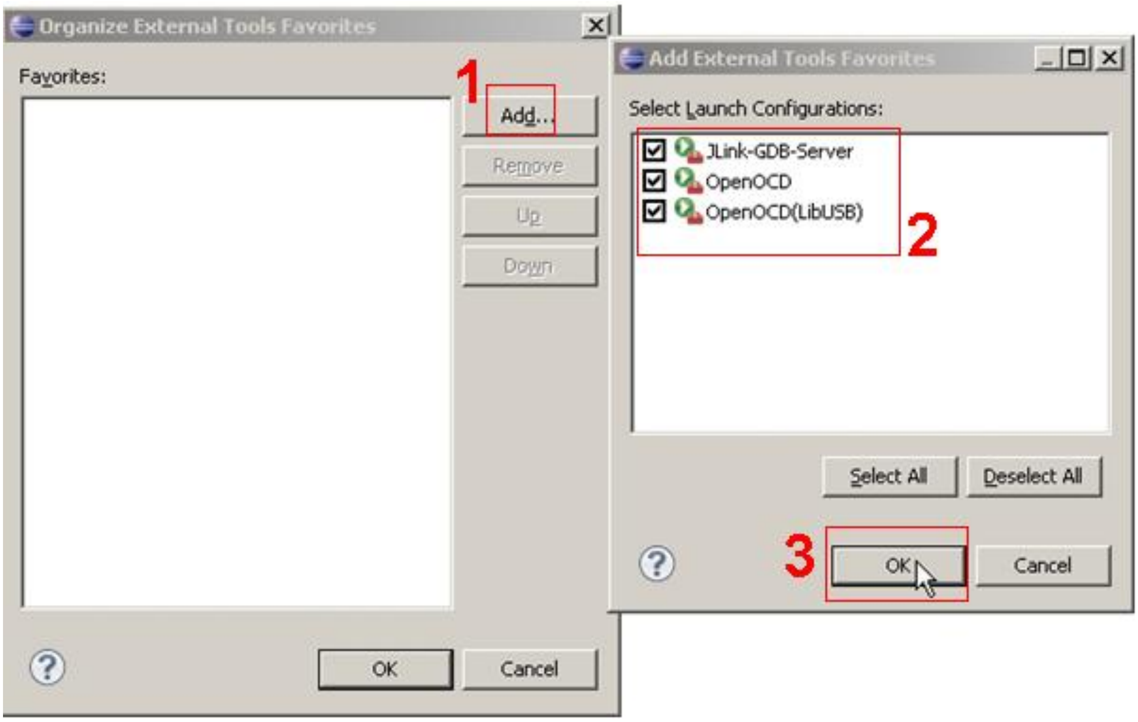
Now organize all external tools needed for debugging.

From the bar menu select the following configuration window:

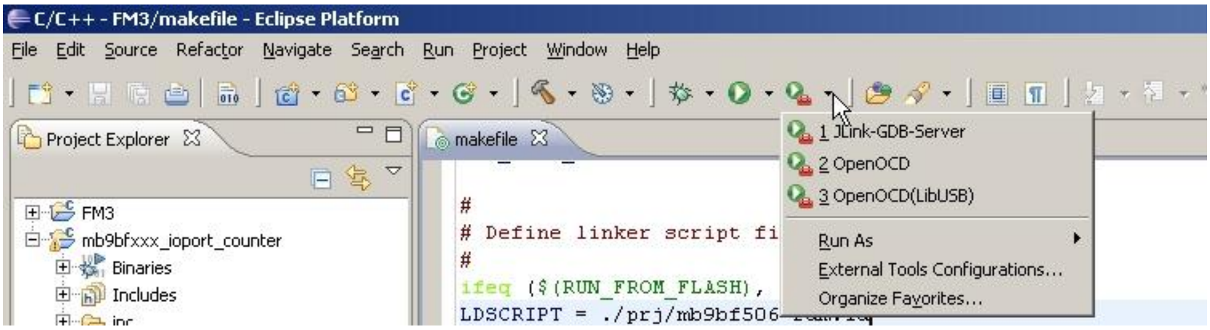
Click on *Organize Favourites......*



Click on *Add* and select all tools.



Click on *Ok* to save the configuration. The external tools are added as favorites. They can be then started from the bar menu as shown below.

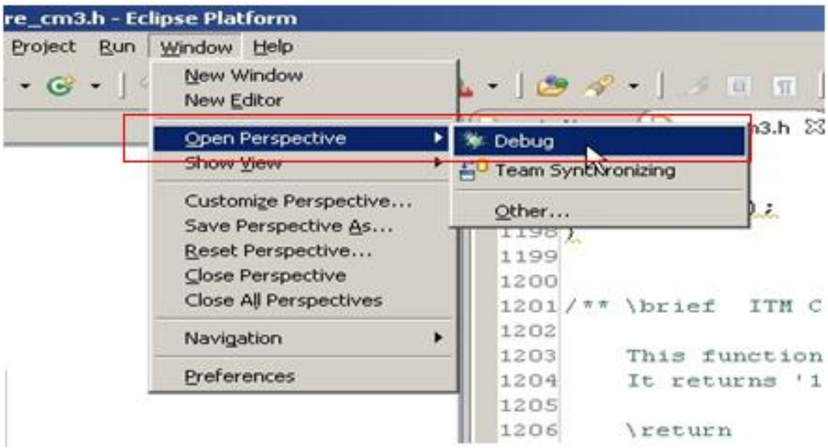


12 Eclipse CDT Debug Perspective

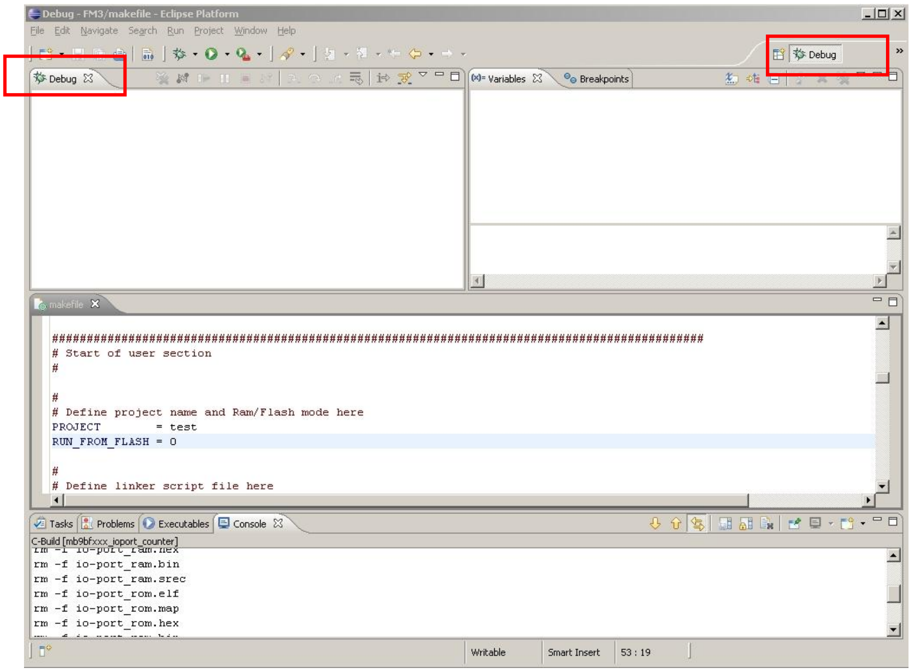
In chapter 8 a sample FM3 project was created and the build process to create all application output files (**.bin*, **.mhx* or **.hex*) needed to program the Flash was explained. These output files include also debug information files (**.elf*) needed for debugging program code in Flash or RAM.

To start the debug process, first change from Eclipse CDT “C/C++ Perspective” to “Debug Perspective”.

Select from Eclipse menu *Windows* and go to *Open perspective*. Click on *Debug*. The debug Perspective can be also found under *Other… .*



After this the following window will be displayed.

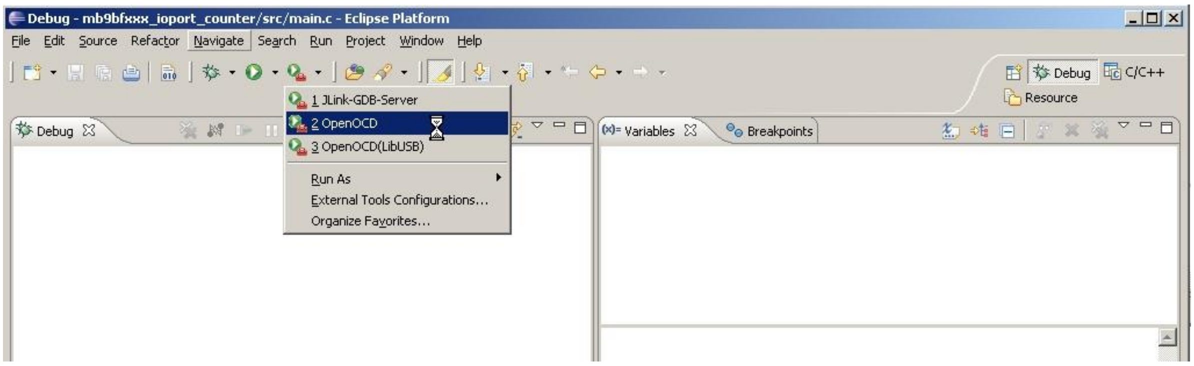


12.1 Using the OpenOCD Server to debug a Flash Application

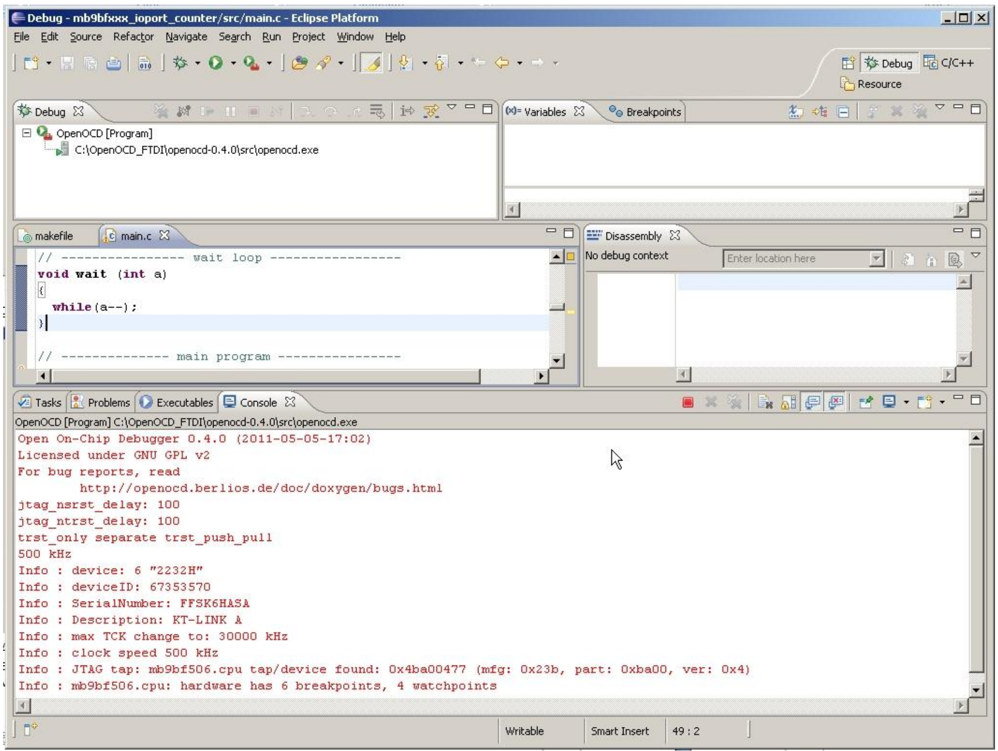
Connect the SK-FM3-176PMC-ETHERNET board via JTAG interface to the USB interface of your computer. As the interface tool for this connection use e.g. the JTAG dongle “J-Link” and “ARM-USB-TINY”.

If using J-Link or ARM-USB-TINY in ICE, the following explanation are common for them. After this start the “OpenOCD”. OpenOCD runs as a daemon, which means, that a program runs in the background waiting for commands to be submitted to it.

Click on *OpenOCD* and the external tool will be started as shown below.



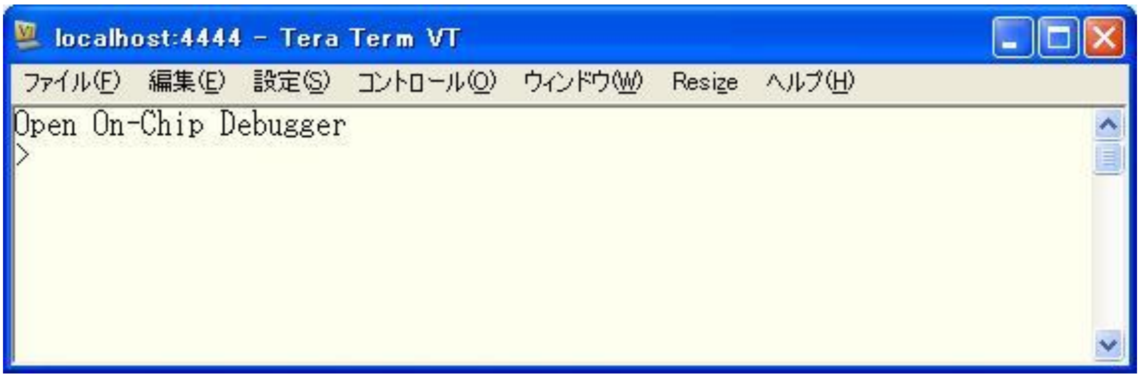
In the console view at the bottom, check that the daemon server has been started.



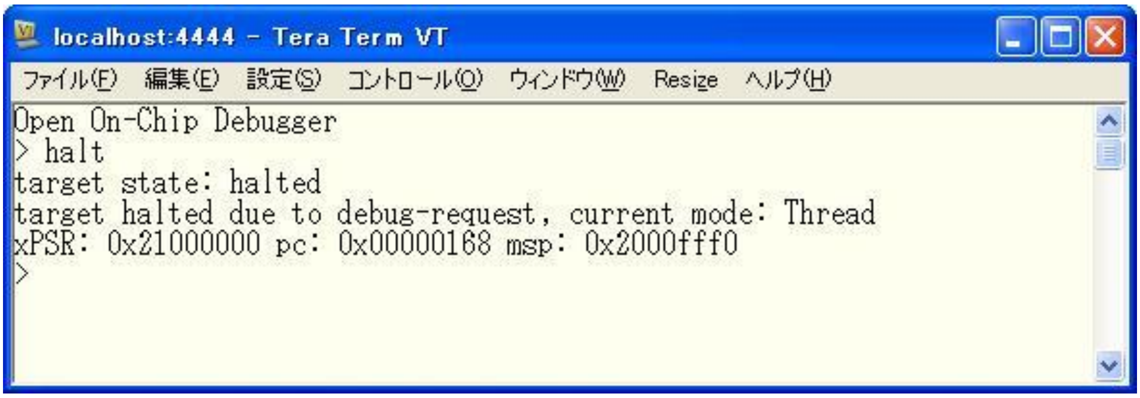
Then, the MCU must be changed to halt state. Because if it is run state, an error may occur between GDB server and OpenOCD.

Please connect to OpenOCD with the terminal emulator(using Tera Term in this documentation).

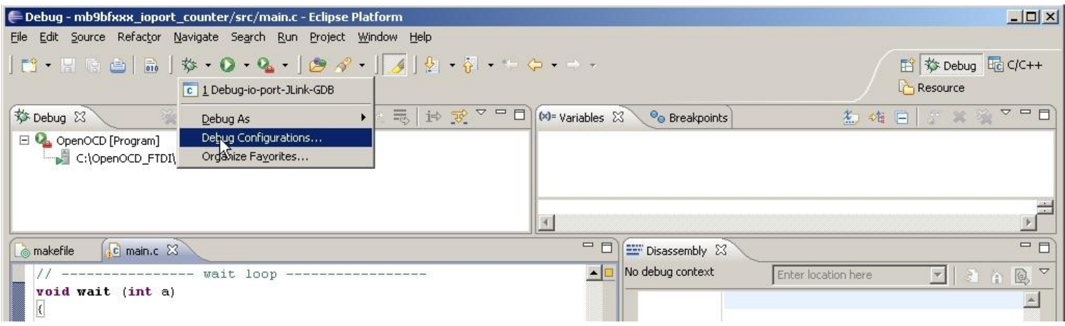
If displayed with “Open On-Chip Debugger”, connection is success.



By *halt* command, confirm that the target is halt state.



Now create a new “Debug Configuration”. For this, click on the *Debug Configurations...* as shown below.



The first debug configuration with "J-Link GDB Server" was saved, but also a special configuration for debugging with OpenOCD is needed.

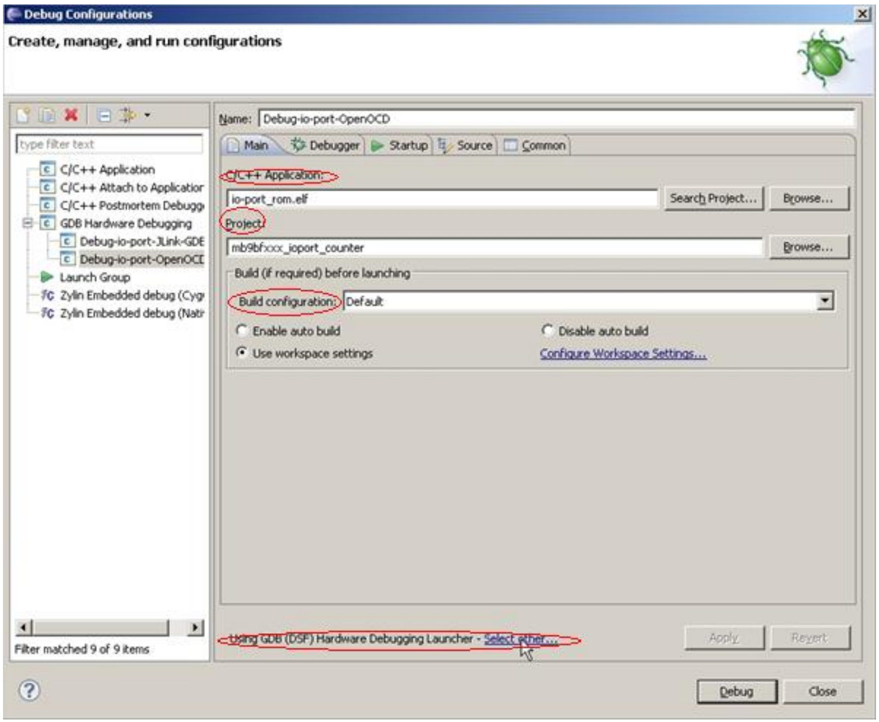
To create a new debug configuration select "GDB Hardware Debugging" and click on *New*.

Rename the debug configuration. To avoid confusion with other debug configurations (using J-Link GDB Server), it is recommended that the selected name a reference to the project name (*io-port*) and to the used external tool (*OpenOCD*).

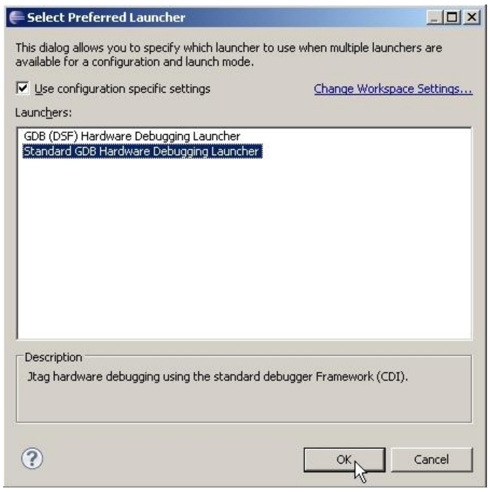
In the "Project" text box, use the *Browse* button to find the project *ioport_sk-fm3-*****.

In the "C/C++ Application" text box, use the *Search Project…* button to locate the application debugger file *io-port_rom.elf*.

Set the "Build configuration" text box to "Use Active".

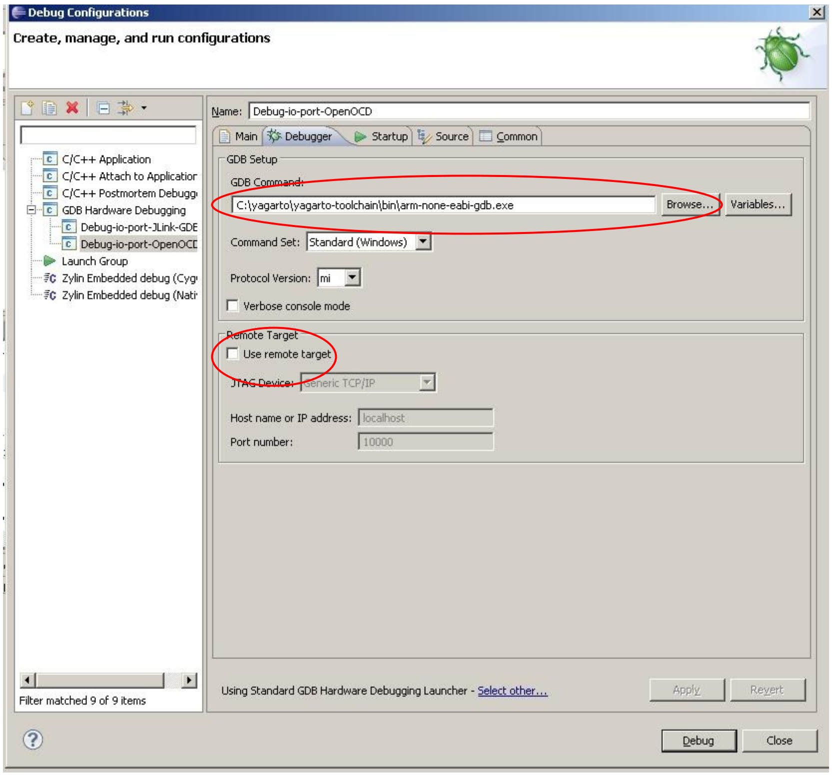


Click on *Select other....* by “Using GDB (DSF) Hardware Debugging launcher” as shown below and select “Standard GDB Hardware Debugging launcher”. Click on *OK*.



Now select the “Debugger” tab as shown below. In the dialog labeled “Debugger Options”, use the *Browse* button to locate the GDB Debugger *arm-none-eabi-gdb.exe* file. It can be found e.g. in: *C:¥yagarto¥yagarto-toolchain¥bin*.

Uncheck *Use remote target*.



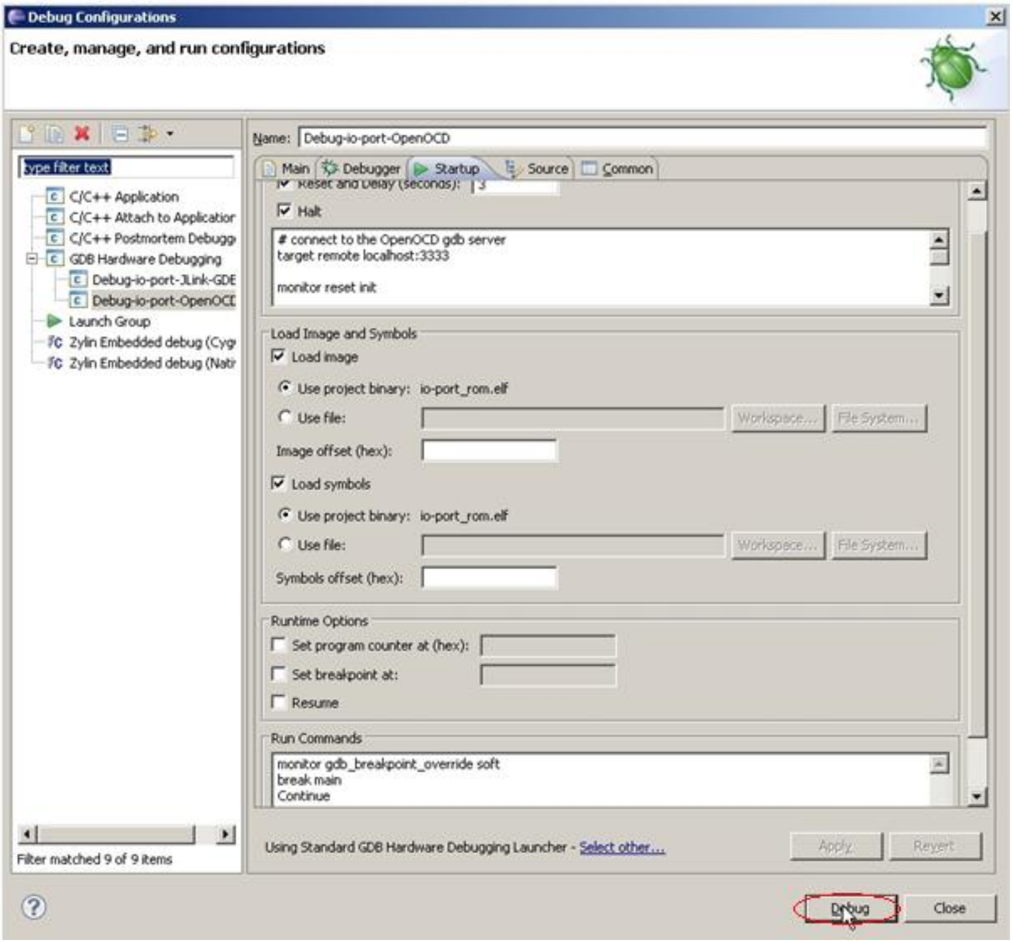
Now select the “Startup” tab as shown below.

On the “Initialization Commands” panel copy or type the following lines:

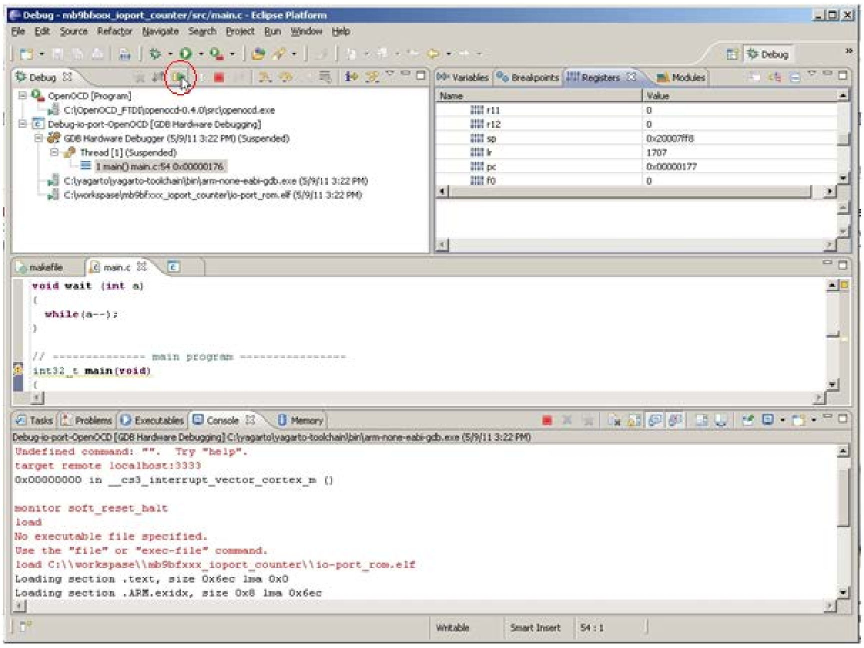
```
# connect to the OpenOCD gdb server
target remote localhost:3333

monitor reset init

monitor soft_reset_halt

load
```

On the “Run Commands” panel add the following lines:

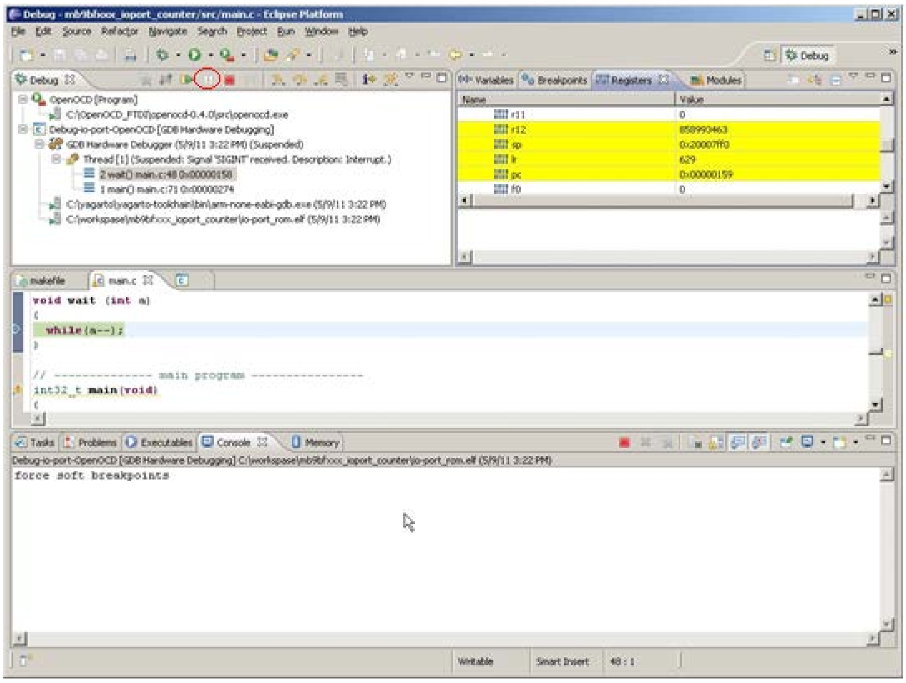
```
monitor gdb_breakpoint_override soft
break main
Continue
```



The rest of the configuration window can be left in its default setting. Click on *Debug* button to start the debug process.



The following figure shows a successful debug start. To resume, simply click on the *Resume* button.

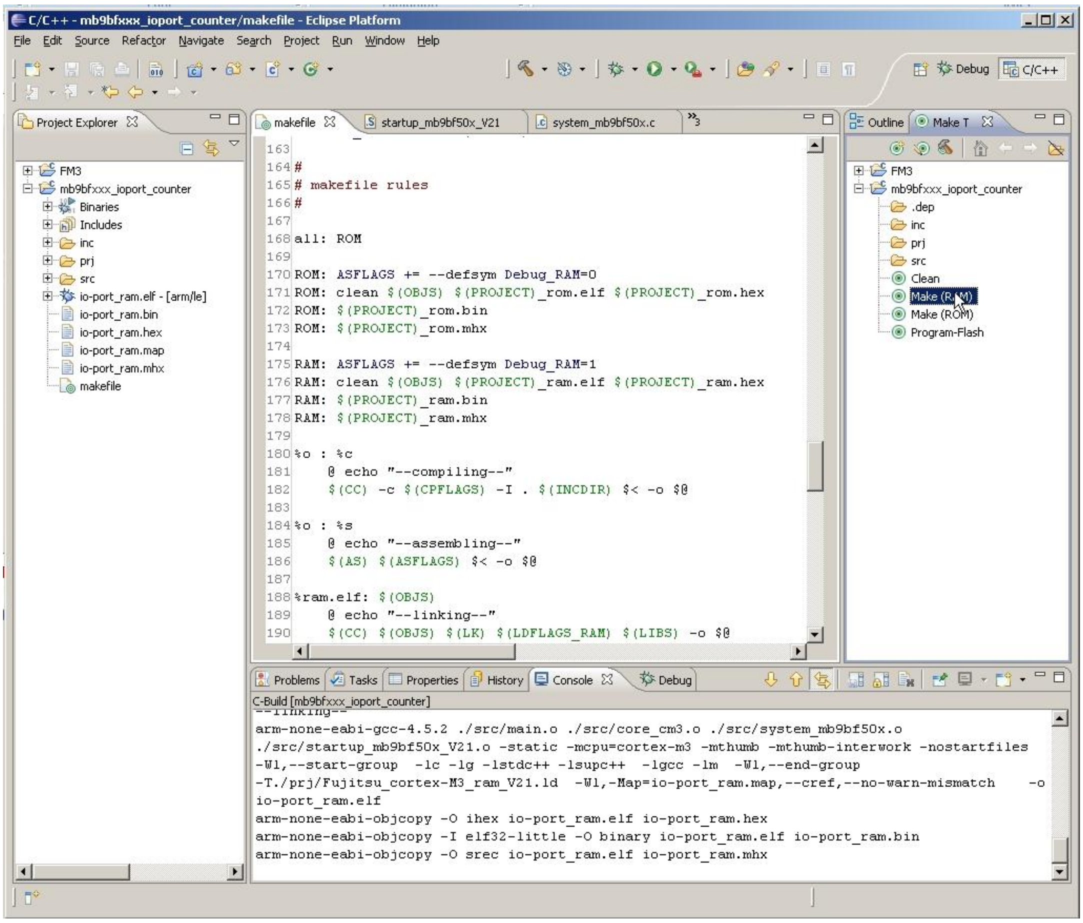


After starting the debug procedure, the debug process can be terminated at any time by clicking on the “*Suspend*” button.

12.2 Debug on the RAM

In the paragraph before the Flash debug was explained from the chapter 12.1. It is also possible to link and download an application for and to the RAM memory of the device. For this the needed RAM application must be created first. To do this, return to the “C/C++ Perspective”.

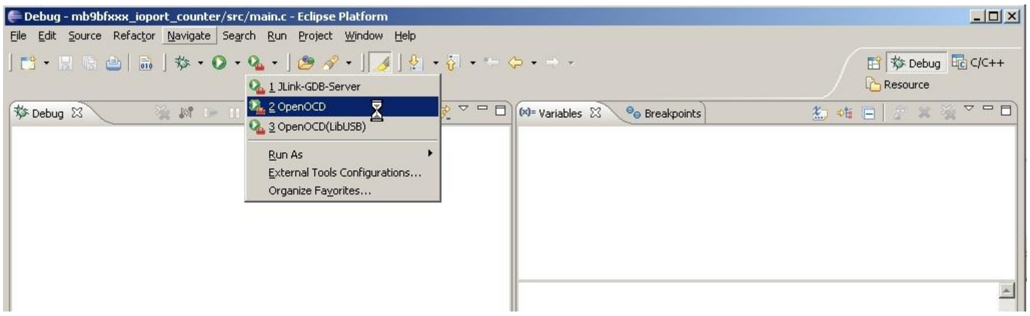
Double click on *C/C++* and the IDE will change be to C/C++ development perspective. Click on *Make (RAM)* to build the RAM make target. The RAM debug application will be generated then (Note, that the application code and the data must not exceed the RAM memory size).



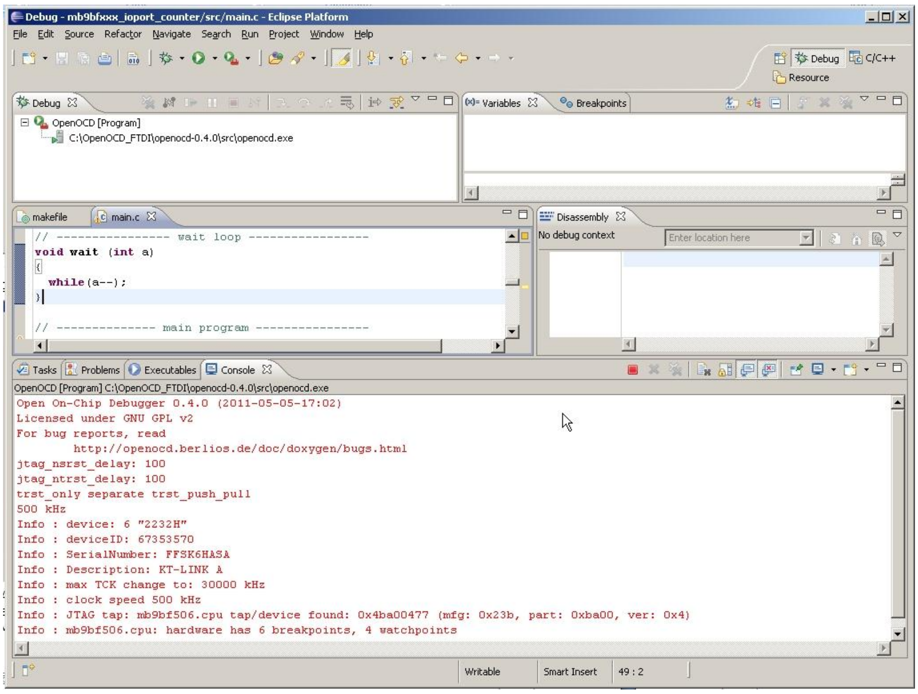
Now switch back to the *Debug perspective* to initiate the RAM debug process.

Reconnect the SK-FM3-176PMC-ETHERNET board via the JTAG interface to the USB interface of your computer.

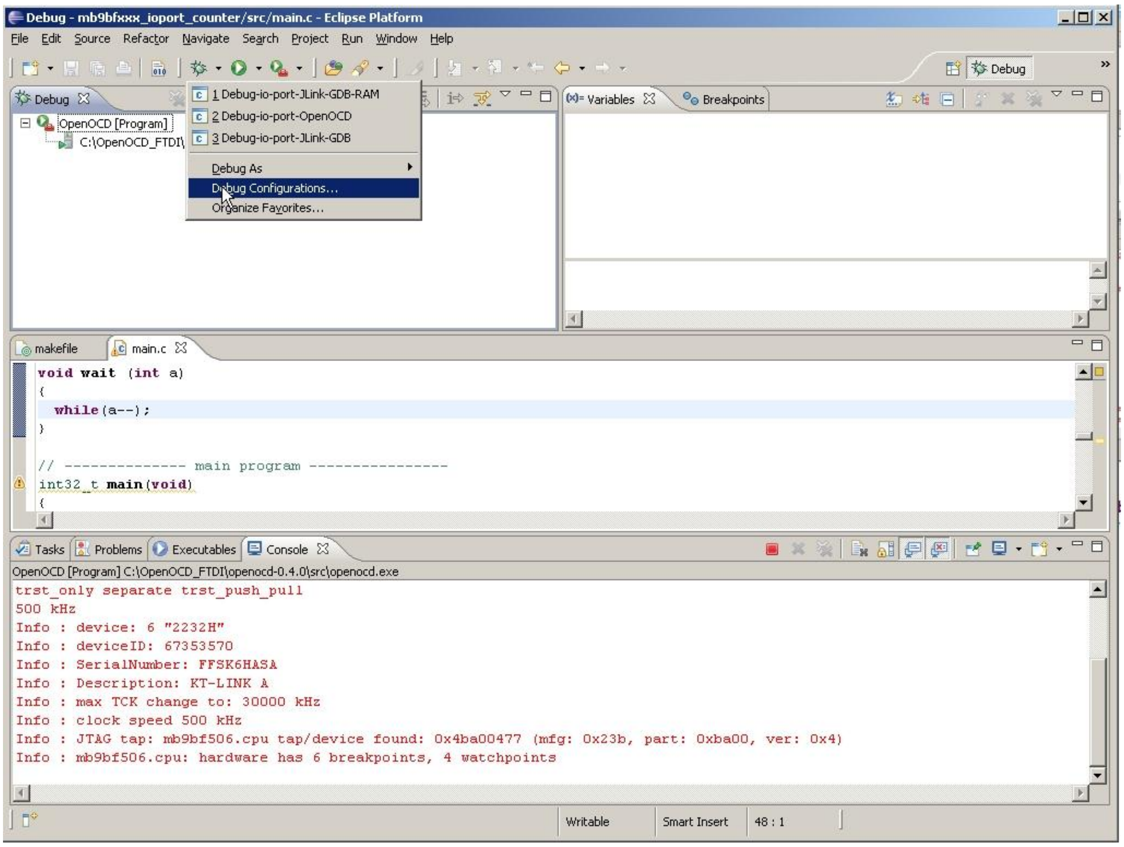
After reconnecting, please start OpenOCD. As follows, click on OpenOCD to start the external tool.



In the console view at the bottom, confirm that the server was started



To create e new debug configuration, choose *Debug Configurations...* as shown below.



Then select “GDB Hardware Debugging” and click on *New*.

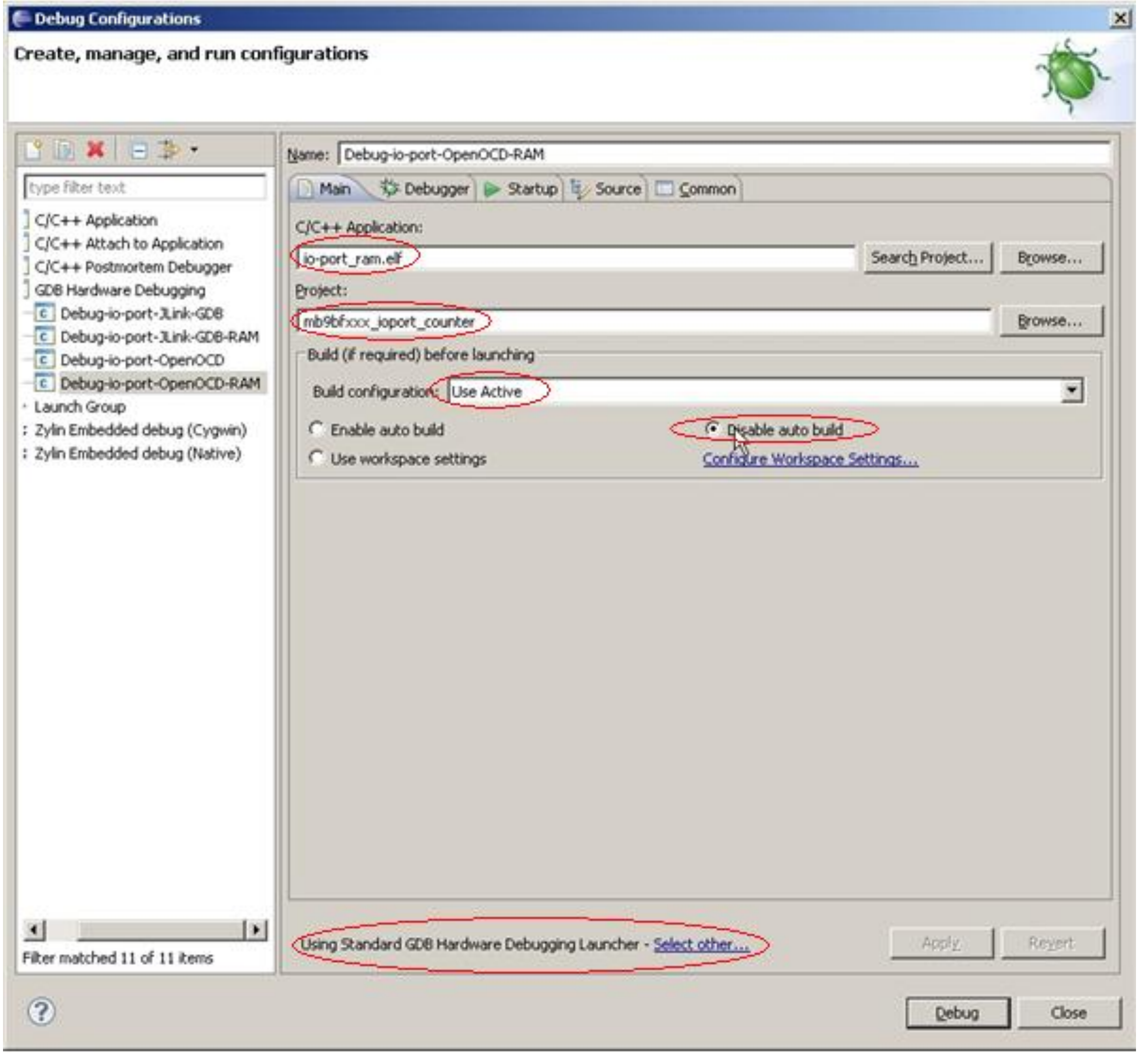
Rename the debug configuration. For differencing the RAM debug from the Flash debug, give the name also a suffix “_RAM” to avoid confusions with the configurations already saved.

In the “Project” text box, use the *Browse* button to find the project *mb9bfxxx_ioport_counter*.

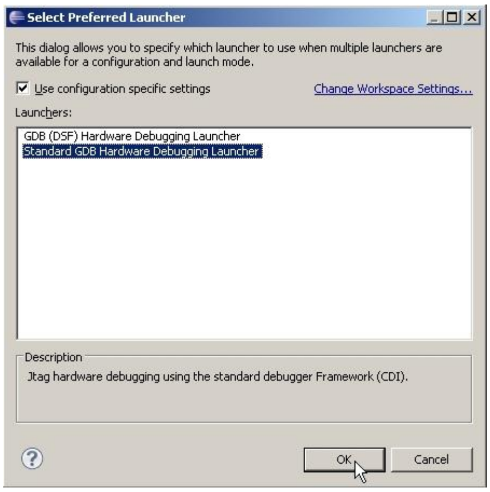
In the “C/C++ Application” text box, use the *Search Project…* button to find the application file *io-port_ram.elf*.

Set the “Build configuration” text box to “Use Active”, and check the box “disable auto build”.

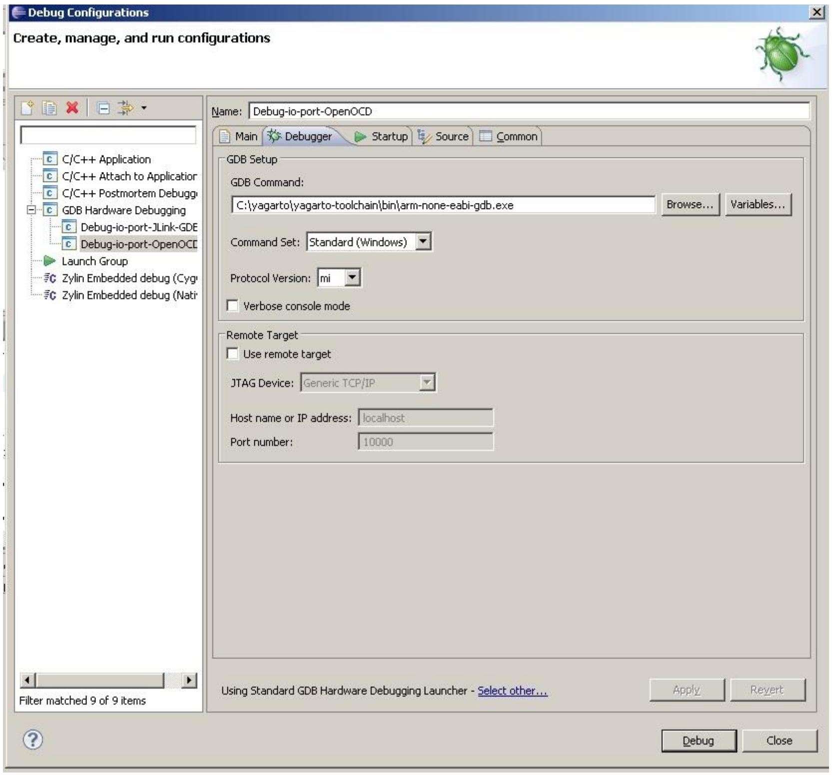
Click on *Select other....* by “Using GDB (DSF) Hardware Debugging launcher” as shown below and select “Standart GDB Hardware Debugging launcher”. Click on *OK*.



Click on *Select other*, please change "GDB(DSF) Hardware Debugging launcher" to "Standard GDB Hardware Debugging launcher". After changing, click on *OK*.



The “Debugger” configuration tab is the same by all configurations.



In the “Startup” tab copy into the “Initialization Commands” panel the following command lines:

```
# connect to the OpenOCD gdb server
target remote localhost:3333

monitor reset init
monitor reset halt
monitor soft_reset_halt

# Vector table placed in RAM
monitor mww 0xE000ED08 0x1fff0000

load
```

Use RAM start (Vector table start) for address! (→ `monitor mww 0xE000ED08 0x1fff0000`)

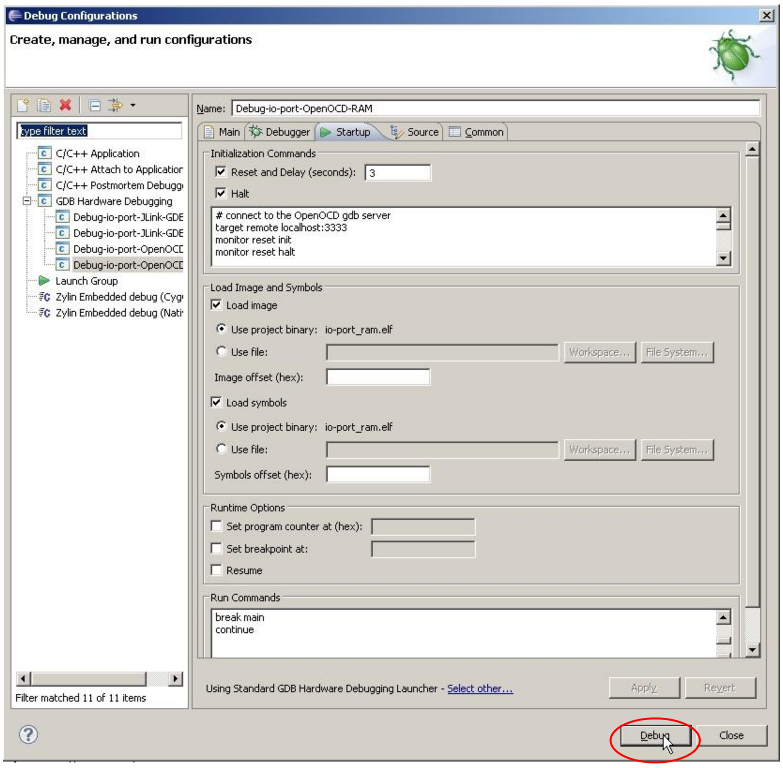
In the “Startup” tab copy into the “Run Commands” panel the following command lines:

```
break main
set $r13 = *(int*)0x1fffE000
set $pc = *(int*)0x1fff0004
continue
```

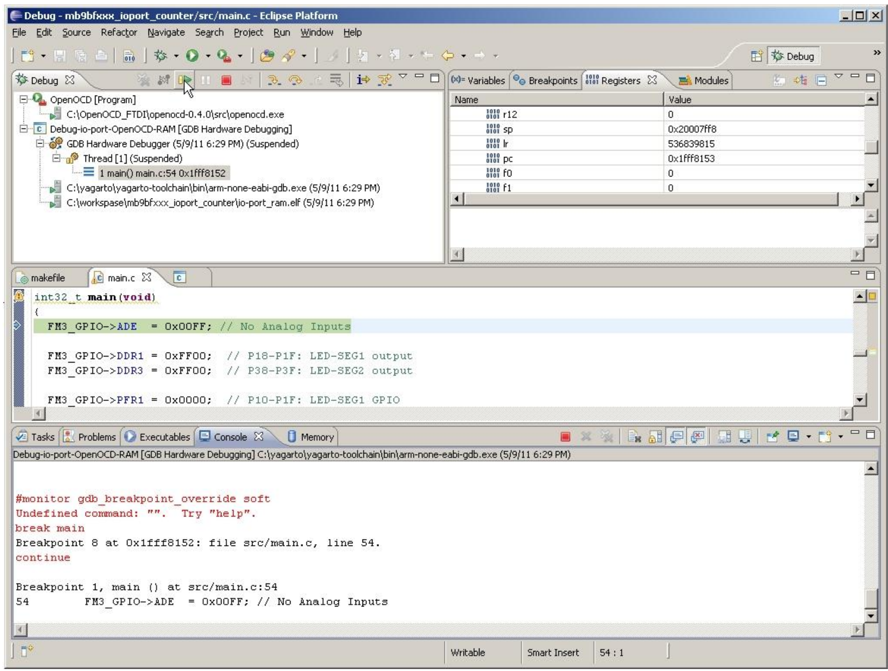
Stack pointer for address! (→ `set $r13 = *(int*)0x1fffE000`)

Use RAM start (Vector table start) + 4 Bytes for address! (→ `set $pc = *(int*)0x1fff0004`)

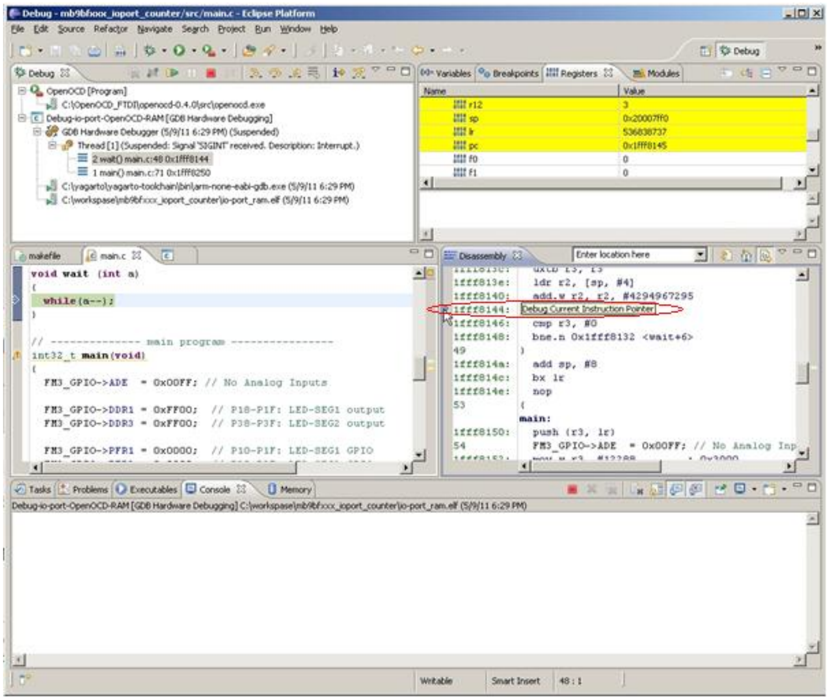
The rest of the configuration window can be left in its default settings. Click on *Debug* button to start the debug process.



The screenshot below shows a successful RAM debug process start. To resume, simply click on the *Resume* button.



On the “Disassembly” view, the current instruction can be observed for example. This view can be selected from the eclipse menu *Window* under *Show View*.

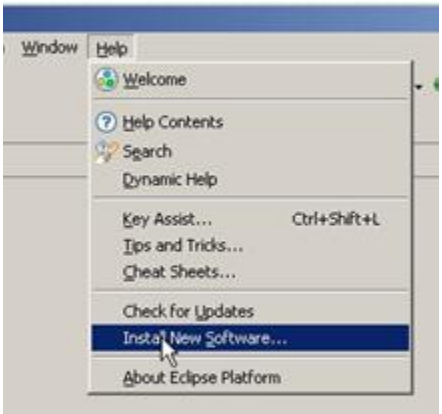


13 Eclipse Embedded Systems Register View Plug-In

The Eclipse plug-in "EmbSysRegView" is useful to get an adequate Eclipse I/O register view allowing a structured display and modification ability of the peripheral register values of all FM3 MCU resources.

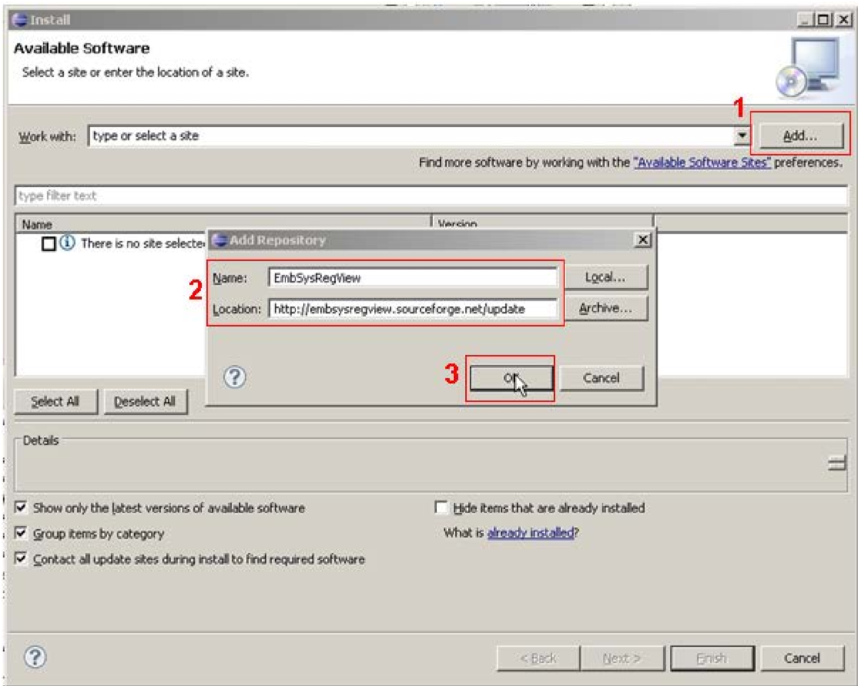
13.1 Plug-In installation

To install the Eclipse Embedded Systems Register View plug-in "EmbSysRegView", open the Eclipse menu *help* and select *Install New Software*.

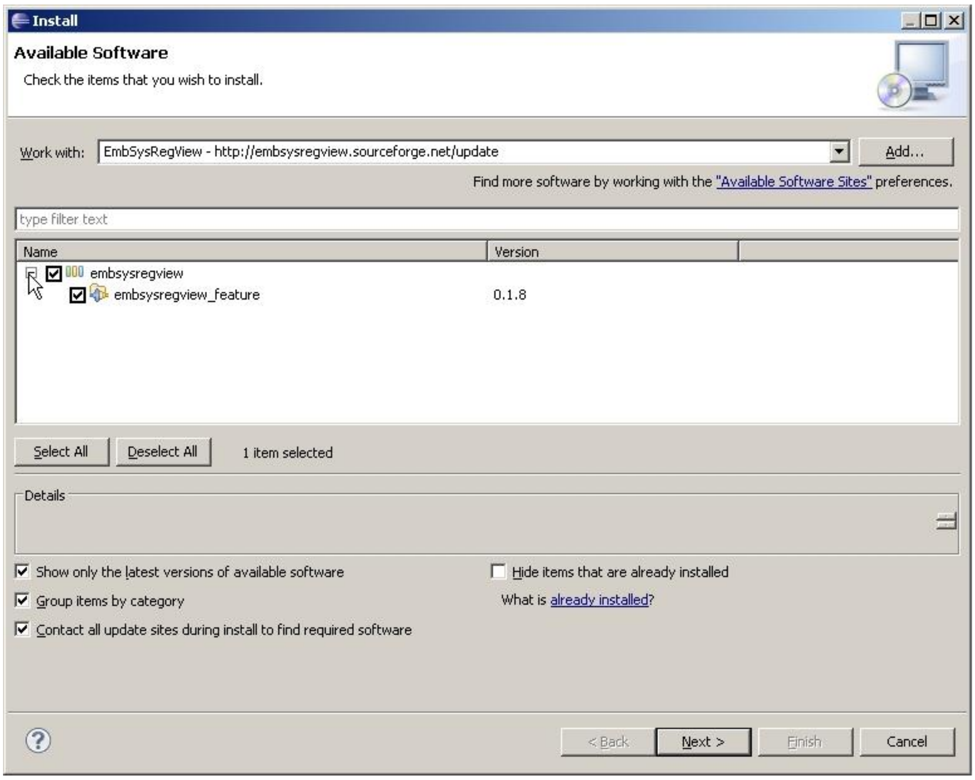


Click on the *Add* button. Enter, e.g. "EmbSysRegView" as name and in the location text box the following link: http://embsysregview.sourceforge.net/update

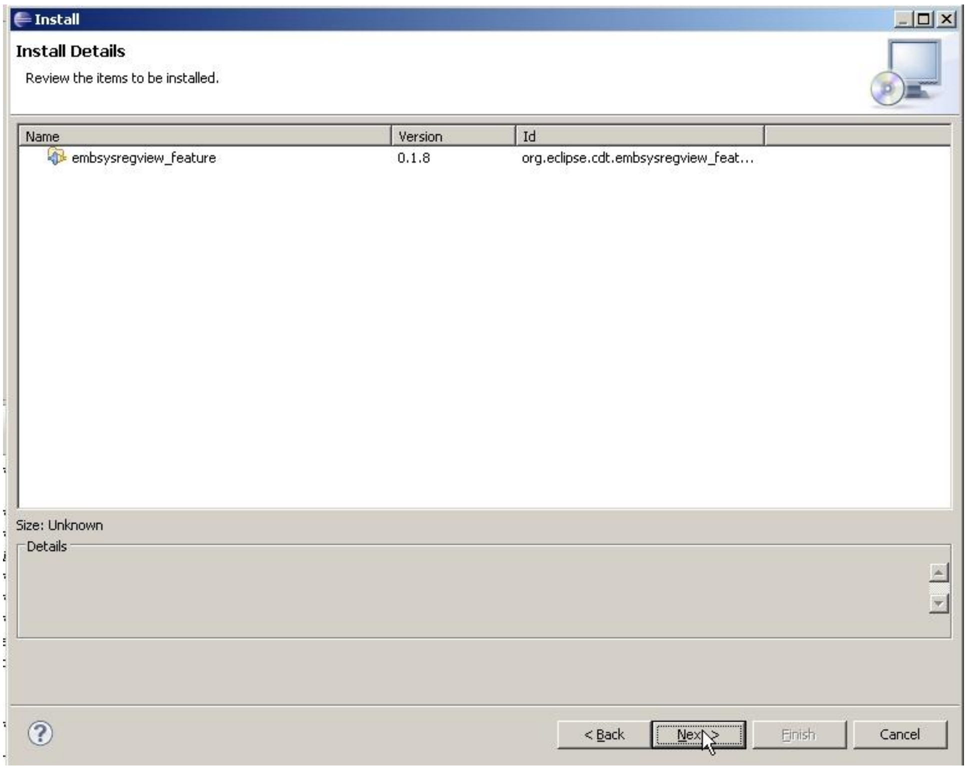
Confirm the repository with *OK*.



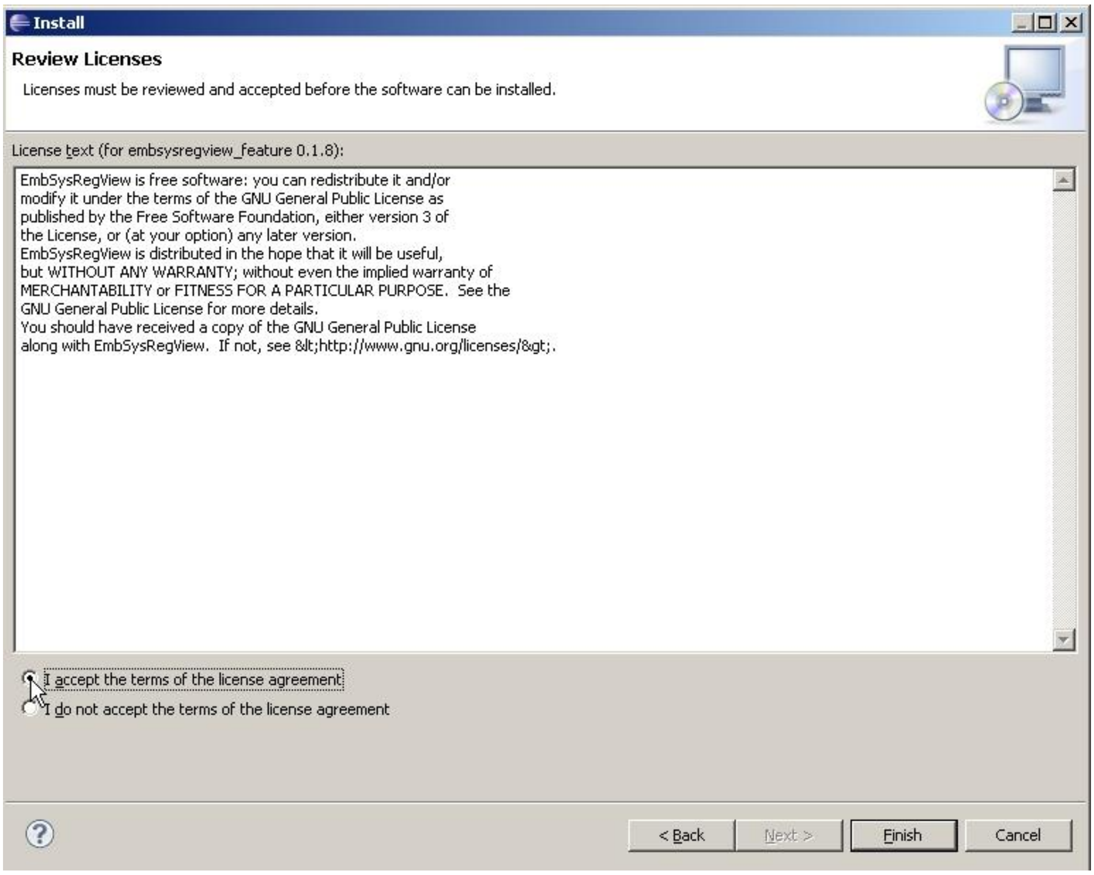
After the confirmation select all plug-in feature and click on *Next*.



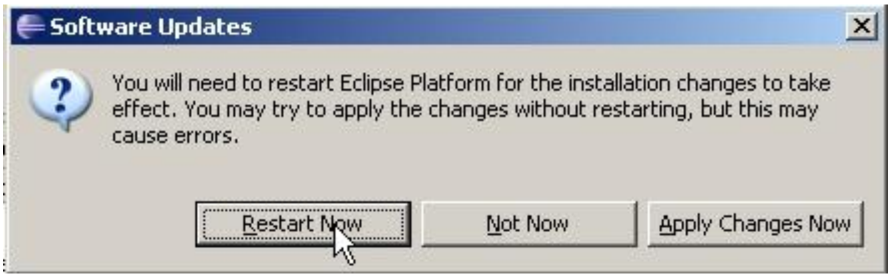
Click on *Next* to confirm the installation detail.



Read the license text thoroughly, check the radio button for “I accept the terms of the license agreement” (or skip the usage in terms of doubts) and close with *Finish*.



Eclipse will ask for IDE restart. Click on *Restart Now*.

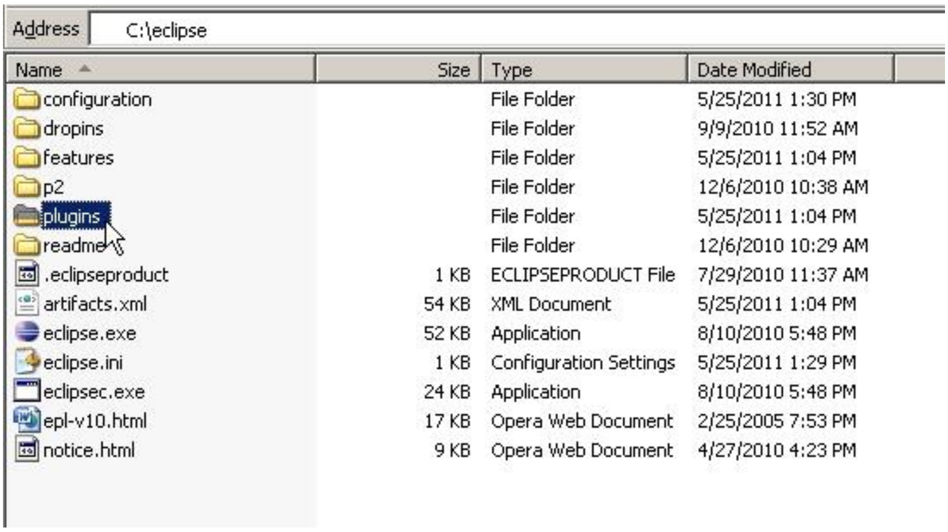


The Eclipse software is now up-to-date and the “EmbSysRegView” is also installed.

13.2 Using the Eclipse Register View

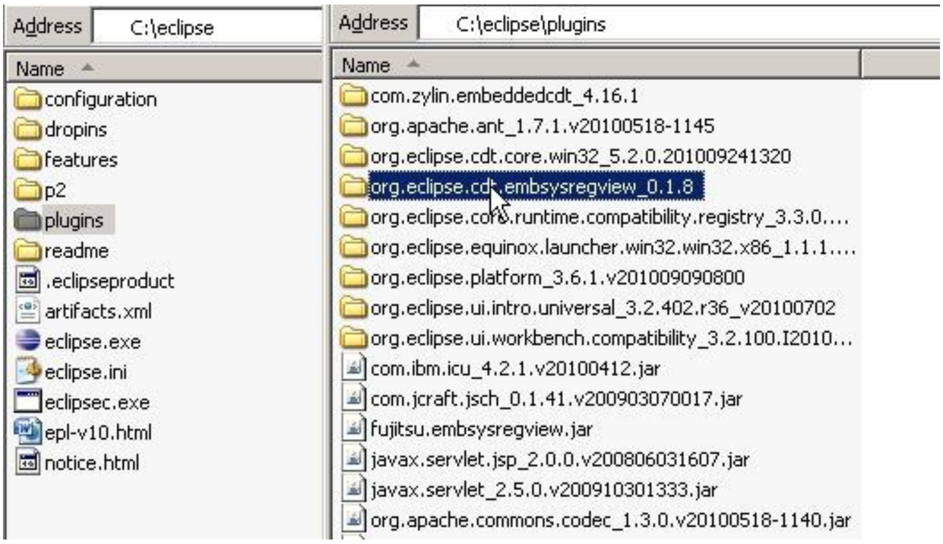
The plug-in “EmbSysRegView” is now installed. To support the peripherals Register viewing for the FM3 MCU, it is needed to use the two FM3 xml description files from Spansion, which comes along with the application note’s software package archive, and copy these files to Eclipse plug-ins directory.

The Eclipse installation directory should have the following structure:

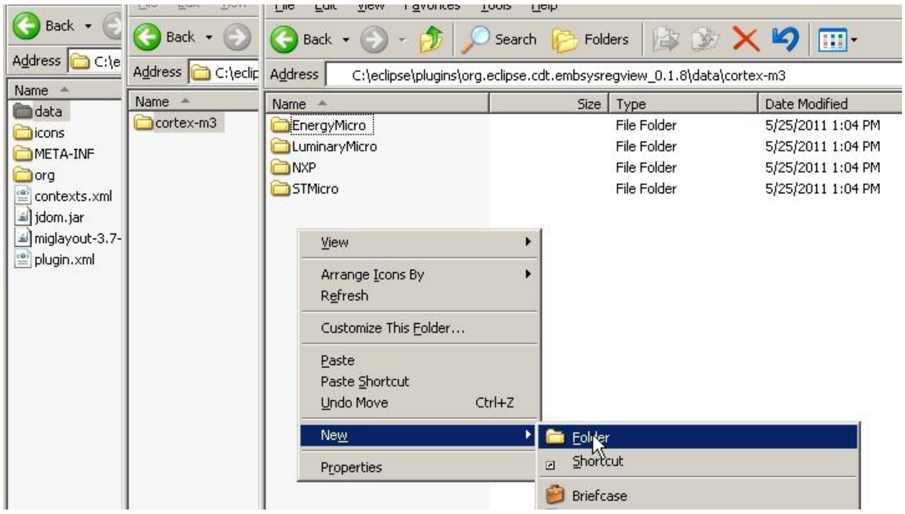

Address C:\eclipse

Name	Size	Type	Date Modified
configuration		File Folder	5/25/2011 1:30 PM
dropins		File Folder	9/9/2010 11:52 AM
features		File Folder	5/25/2011 1:04 PM
p2		File Folder	12/6/2010 10:38 AM
plugins		File Folder	5/25/2011 1:04 PM
readme		File Folder	12/6/2010 10:29 AM
.eclipseproduct	1 KB	ECLIPSEPRODUCT File	7/29/2010 11:37 AM
artifacts.xml	54 KB	XML Document	5/25/2011 1:04 PM
eclipse.exe	52 KB	Application	8/10/2010 5:48 PM
eclipse.ini	1 KB	Configuration Settings	5/25/2011 1:29 PM
eclipsec.exe	24 KB	Application	8/10/2010 5:48 PM
epl-v10.html	17 KB	Opera Web Document	2/25/2005 7:53 PM
notice.html	9 KB	Opera Web Document	4/27/2010 4:23 PM

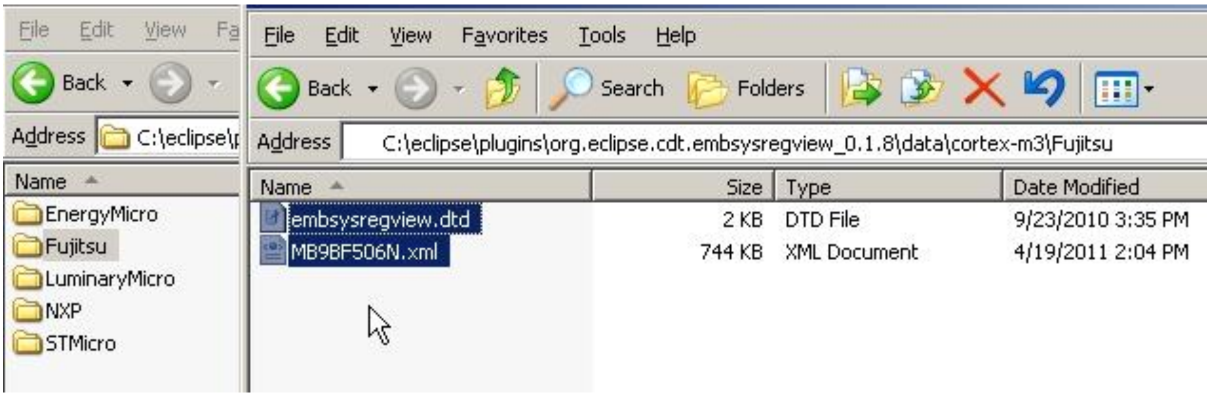
Open the directory ¥*plugins* and look for the installation directory for the installed plug-in “EmbSysRegView”.



Open the selected directory and create a new folder with the name e.g. *Fujitsu* to directory: *¥data¥cortex-m3*

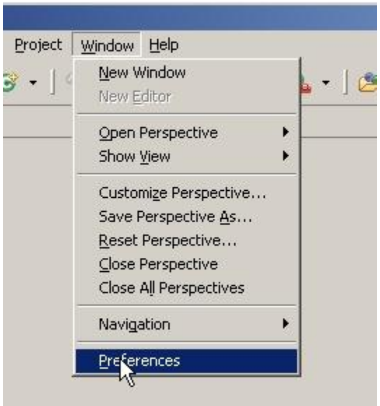


When the folder *Fujitsu* is created, add both description files *embsysregview.dtd* and *MB9BF506N.xml* to it.

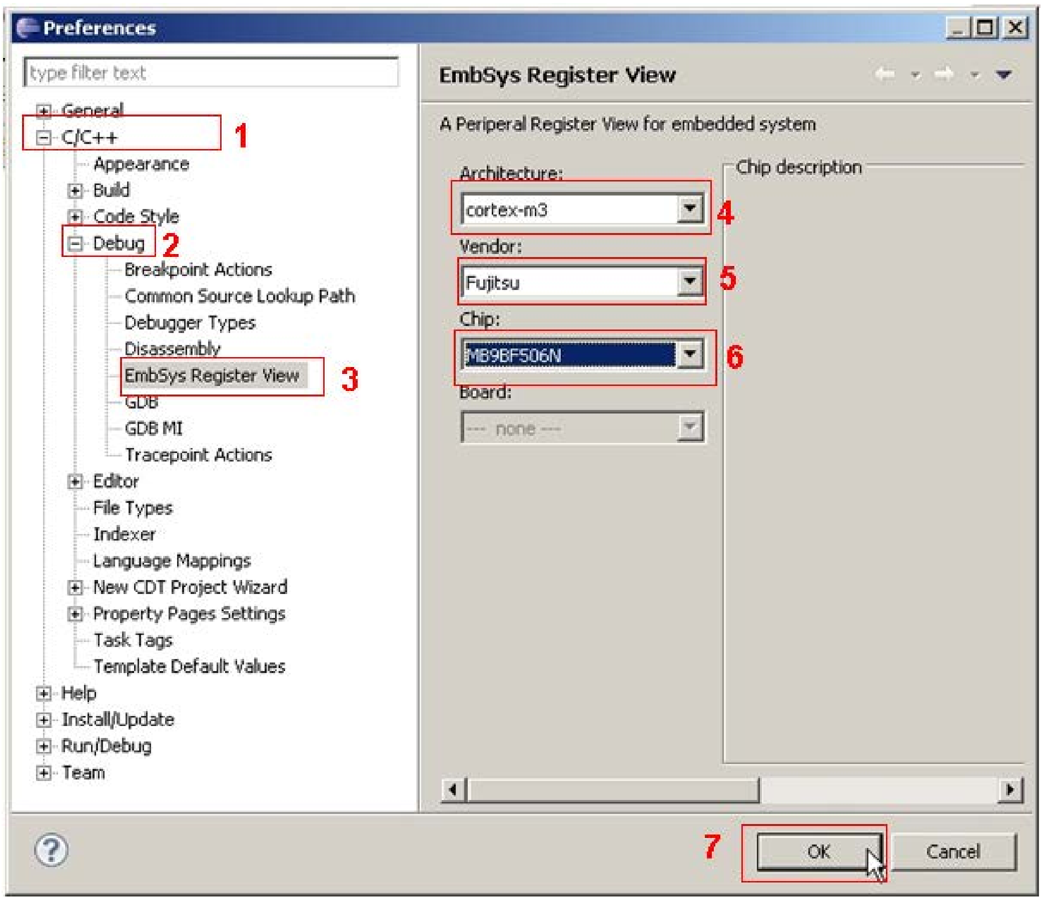


Now go back to Eclipse IDE and use the installed Register view.

For this, open *Preferences* in the Eclipse's *Window* pull-down menu.

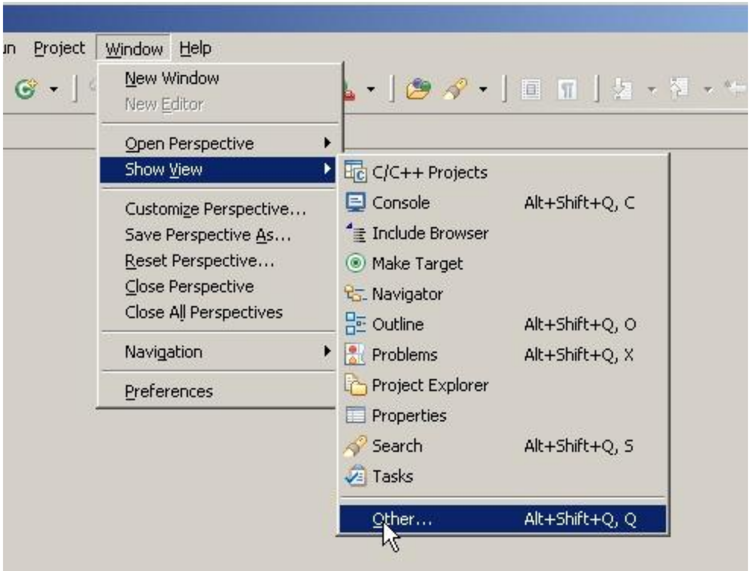


Select the correct device as shown in the figure below.

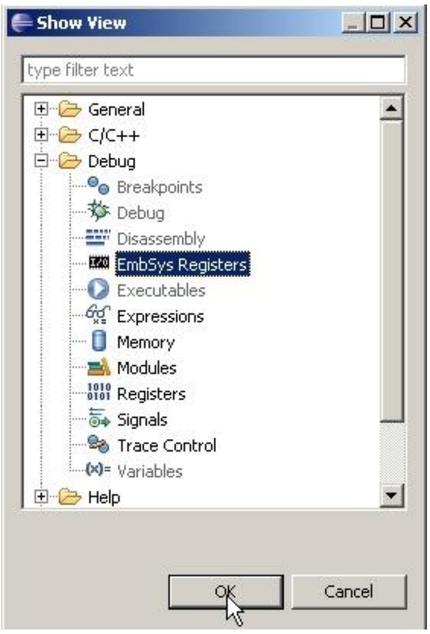


After Confirming the Register view configuration, the tool can be now used.

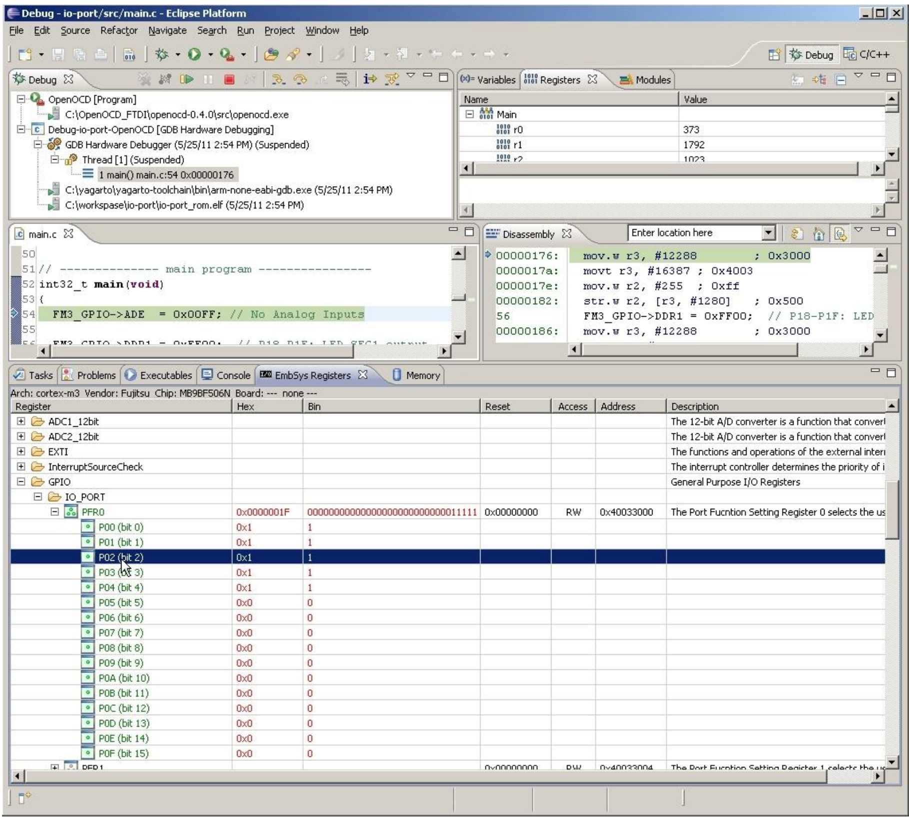
To open a register viewer in the CDT debug perspective (see chapter 12 for detailed information), select *Show View→Other…* in the Eclipse's *Window* pull-down menu.



Then expand the "Debug" node and select "EmbSys Registers". Confirm with *OK*.



During debugging on the RAM or ROM (Flash), the debug process must be stopped in a breakpoint to get content (and refresh) of a certain register. Double click on this register to start viewing its content. Registers which are selected get a green font. Changes in register contents are shown with red values. When hovering over a register's description column you see a short description for that register.



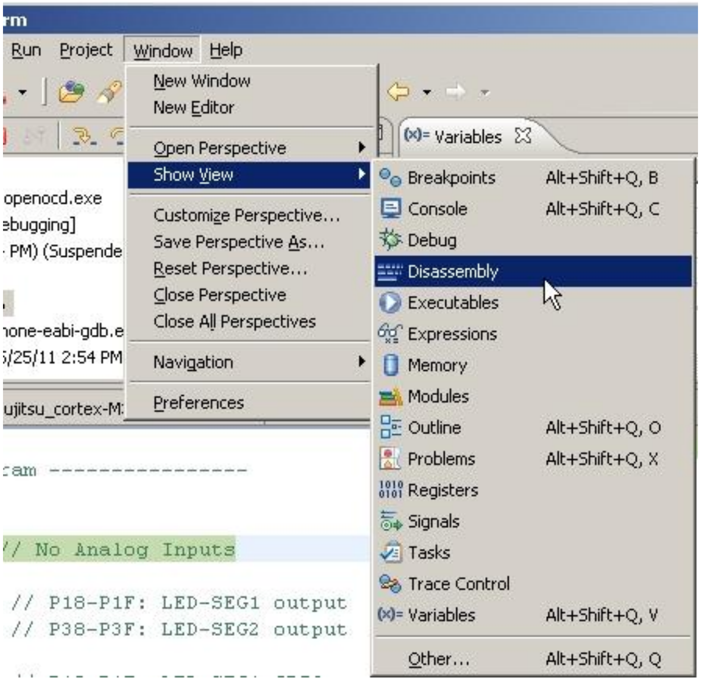
14 Eclipse Features

14.1 Overview

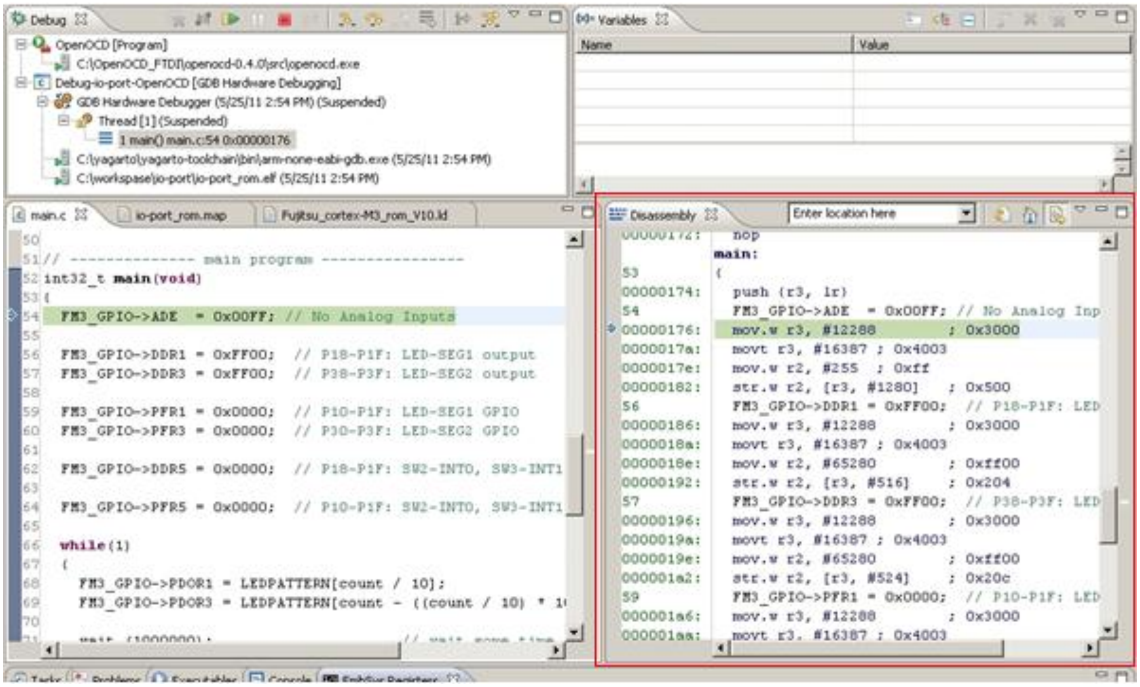
The Eclipse CDT provides many tools and features, which can help the user for the embedded software development for a FM3 MCU.
In the next paragraphs some of these features of the debug perspective are discussed.

14.2 Disassembly View

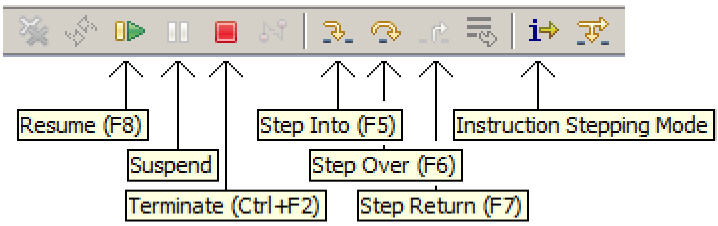
To display the “Disassembly” view in the CDT debug perspective (see chapter12 for details), select *Show View→Disassembly* in Eclipse's *Window* pull-down menu.



The view will be then displayed as shown below.



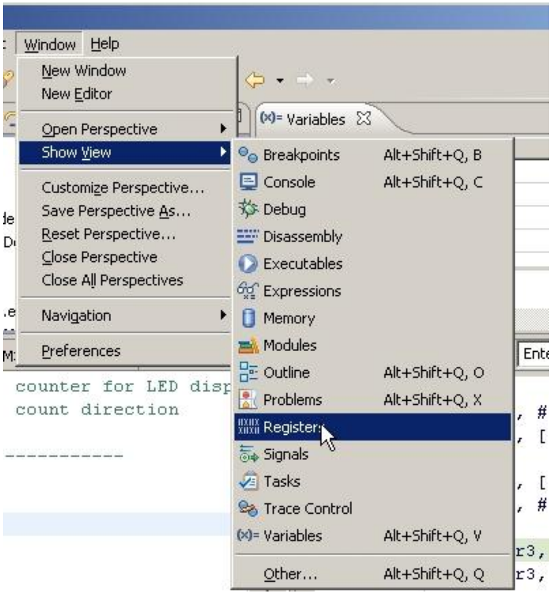
On this view a pointer to the current instruction will be set, so that the user can break the debugging process any time by clicking on the button *Suspend*. Do not mix it up with *Terminate*, which will end the debug session!



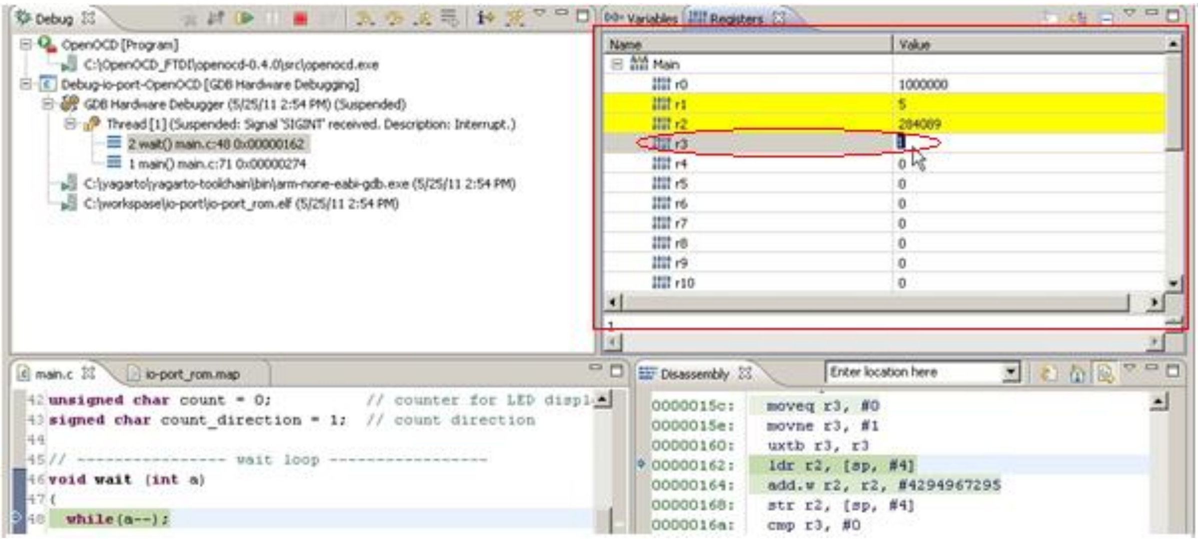
14.3 CPU Register View

The Eclipse CDT provides a register view that enables read and write access to the core registers.

To get this view, select *Show View→Register* in Eclipse's *Window* pull-down menu.



The selected view displays all core registers and their contents. Open the tree “Main” to get a CPU registers overview.

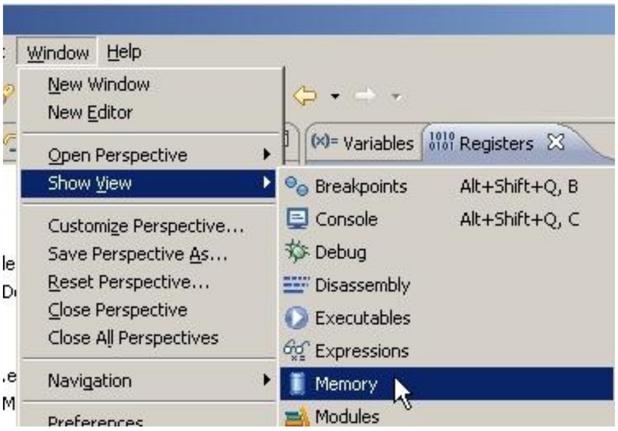


To edit the content of a register, select the register and double click on it.

14.4 Memory View

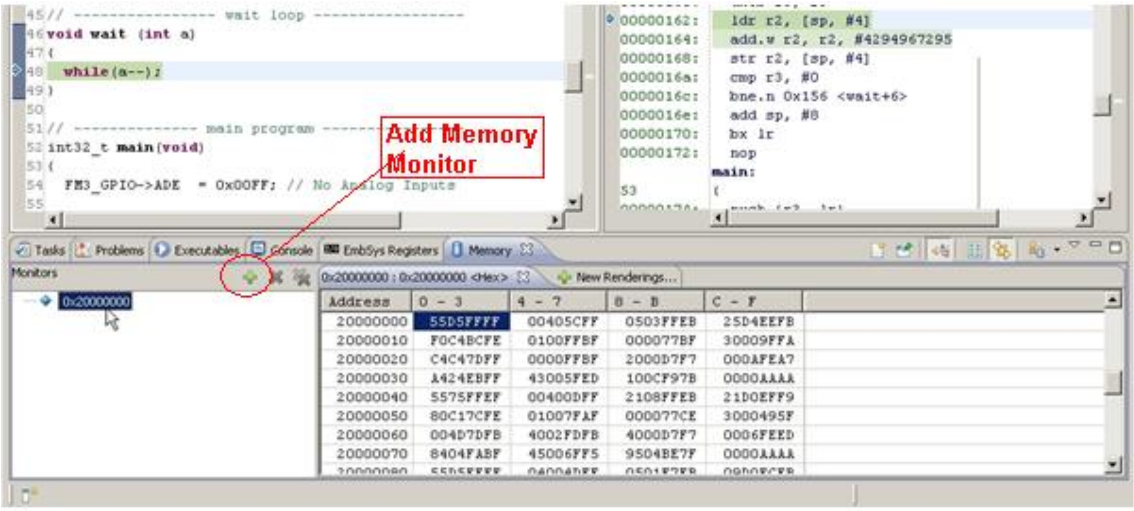
Eclipse's memory monitor view is a default part of the debug view.

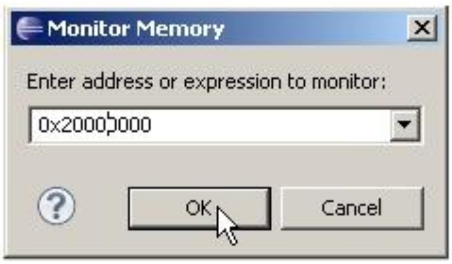
Select *Show View→Memory* in Eclipse's "*Window*" pull-down menu.



To add a new memory monitor, click to the green plus sign in the Monitor pane.

The figure below shows the active memory monitors at address 0x20000000.

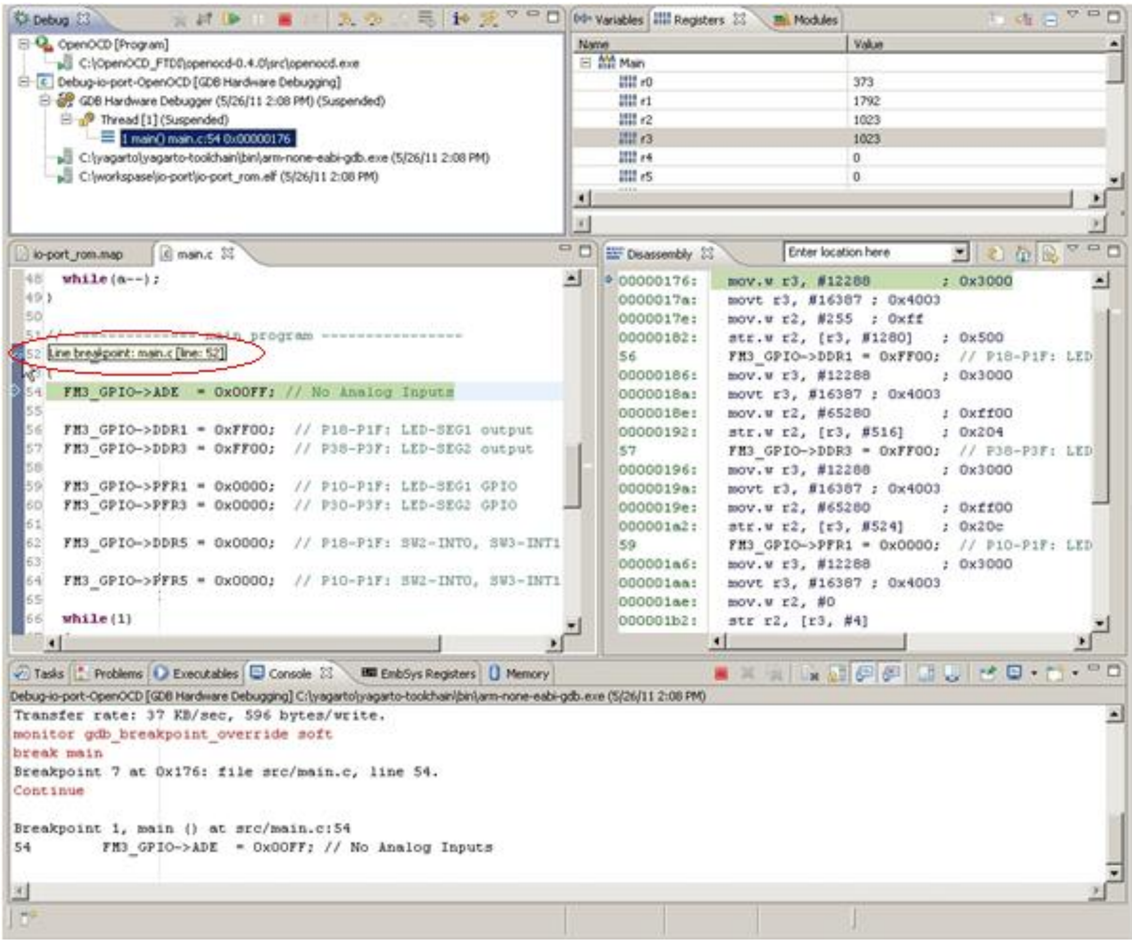




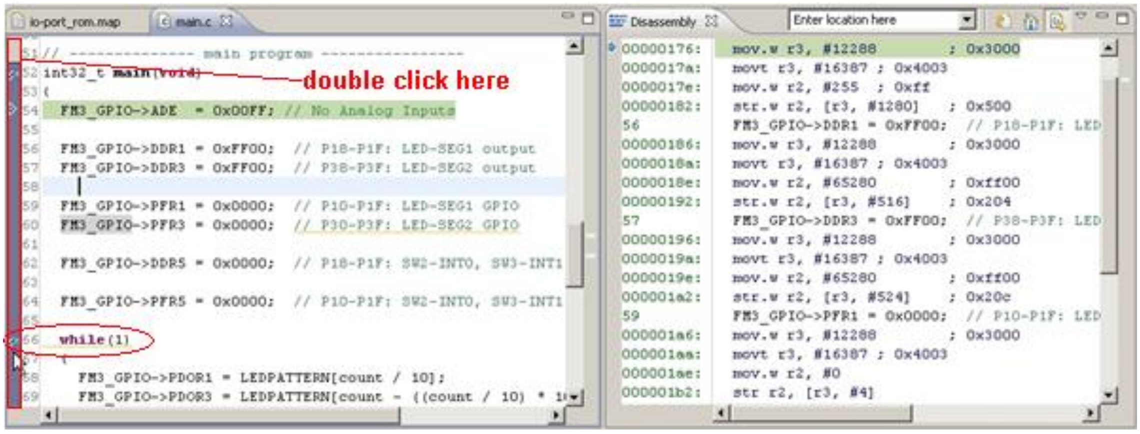
The content of a selected memory address (RAM and some I/O resources) can be edited and changed by double clicking on the respective address.

14.5 Using Breakpoints on Eclipse Debug Perspective

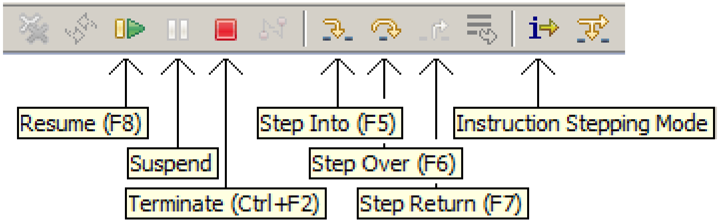
After starting a debug session, the debugger will set a breakpoint at the main function.



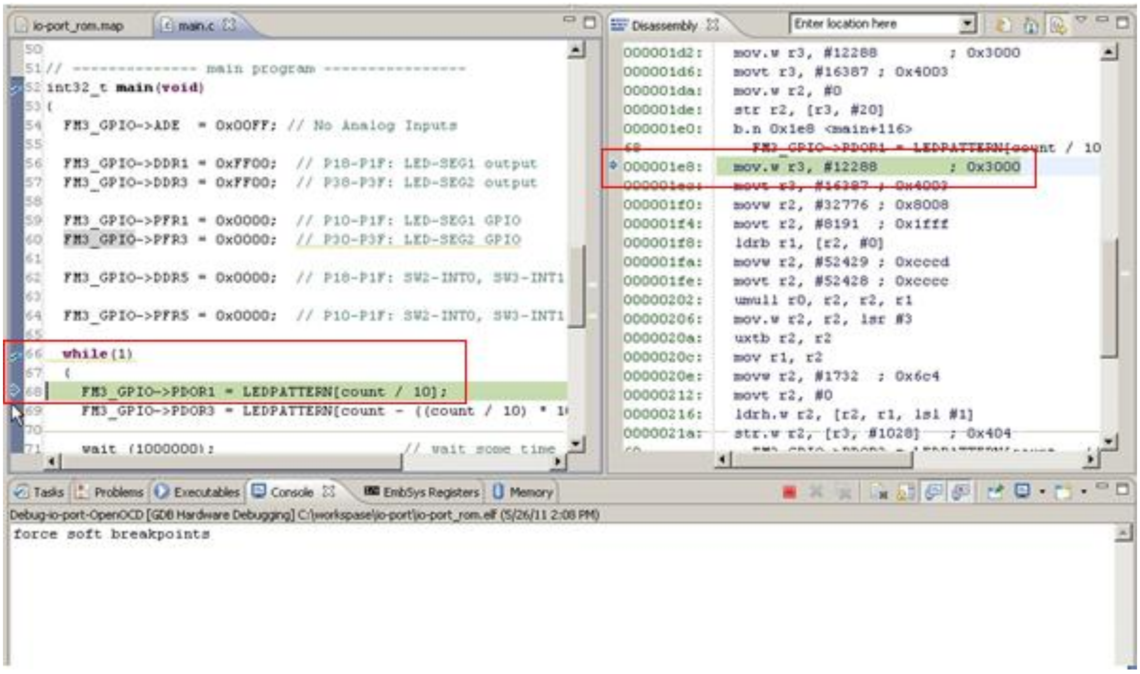
Other breakpoints can be set by double clicking in the left pane in the source code tab beside the line numbers.



Now *Resume* the debug session.



The next figure demonstrates debug process, if a breakpoint was hit.



15 Appendix

15.1 Glossary

Used abbreviations in this document

Abbr.	Meaning	Short Explanation
*.bin (file extension)	Binary Format File	A file that contains program data in raw binary form without any additional information
*.elf (file extension)	Executable and Linkable Format	Object code containing debug information (symbols, addresses, modules, etc.)
*.hex (file extension)	Hexadecimal format file (Intel)	A file that contains program data and address information (Intel format)
*.mhx (file extension)	Motorola Hexadecimal Format File	A file that contains program data and address information (Motorola S-Records format)
CDT	C/C++ Development Tooling	Tool Chain with is used by Eclipse in this configuration
EABI	Embedded-Application Binary Interface	Standard format convention interface for embedded applications (used in Linux systems → cf. None-EABI)
FTDI	Future Technology Devices International Ltd.	Company, which provides the JTAG-to-USB interface chips et al.
JTAG	Joint Test Action Group	IEEE Standard 1149.1 for testing and debugging hardware (here: MCUs)
JRE	Java Runtime Environment	Environment software for a virtual machine, which allows to run JAVA applets (e.g. Eclipse) on the PC
GDB	GNU Debugger	Debugger software for the GNU Tool Chain
GNU	"GNU's not Unix"	Development Tool Chain
LibUSB	Library for USB	Open source library for USB drivers, here the Windows compilation is used
None-EABI	None-Embedded-Application Binary Interface	Embedded application layer interface for non-Linux systems, here: Windows OS (→ cf. EABI
OCD	On-Chip Debugger/Debugging	Debugger software for on-chip debugging, here using the JTAG protocol
OpenOCD	Open Source On-Chip Debugger	Open Source Code Debugger Software
YAGARTO	"Yet another GNU ARM tool chain"	GNU tool chain ported and precompiled for Windows OS

15.2 Links

15.2.1 Software

Eclipse IDE:
http://download.eclipse.org/eclipse/downloads/

Yagarto Tool Chain:
www.yagarto.de

OpenOCD:
http://openocd.sourceforge.net/

LibUSB:
http://sourceforge.net/projects/libusb-win32/files/

Embedded System Register View Plug-In for Eclipse:
http://sourceforge.net/projects/embsysregview/

JRE:
http://java.com/

15.2.2 Hardware

J-Link from IAR
http://www.iar.com/Global/Products/Hardware-Debug-probes/DS-J-Link-ARM-09.pdf

ARM-USB-TINY from olimex
https://www.olimex.com/Products/ARM/JTAG/ARM-USB-TINY/

SK-FM3-176PMC-ETHERNET V1.1 from Spansion Semiconductor
http://www.spansion.com/products/microcontrollers/pages/tool-detail-sk-fm3-176pmc-ethernet.aspx

NOTE : These URLs are subject to change without notice.

16 Additional Information

Information about Spansion's Microcontroller can be found on the following Internet page:

http://www.spansion.com/

Revision History

Rev	Date	Remark
1.0	Jan. 07, 2013	First Edition
1.1	Jan. 31, 2014	Company name and layout design change

Colophon

The products described in this document are designed, developed and manufactured as contemplated for general use, including without limitation, ordinary industrial use, general office use, personal use, and household use, but are not designed, developed and manufactured as contemplated (1) for any use that includes fatal risks or dangers that, unless extremely high safety is secured, could have a serious effect to the public, and could lead directly to death, personal injury, severe physical damage or other loss (i.e., nuclear reaction control in nuclear facility, aircraft flight control, air traffic control, mass transport control, medical life support system, missile launch control in weapon system), or (2) for any use where chance of failure is intolerable (i.e., submersible repeater and artificial satellite). Please note that Spansion will not be liable to you and/or any third party for any claims or damages arising in connection with above-mentioned uses of the products. Any semiconductor devices have an inherent chance of failure. You must protect against injury, damage or loss from such failures by incorporating safety design measures into your facility and equipment such as redundancy, fire protection, and prevention of over-current levels and other abnormal operating conditions. If any products described in this document represent goods or technologies subject to certain restrictions on export under the Foreign Exchange and Foreign Trade Law of Japan, the US Export Administration Regulations or the applicable laws of any other country, the prior authorization by the respective government entity will be required for export of those products.

Trademarks and Notice

The contents of this document are subject to change without notice. This document may contain information on a Spansion product under development by Spansion. Spansion reserves the right to change or discontinue work on any product without notice. The information in this document is provided as is without warranty or guarantee of any kind as to its accuracy, completeness, operability, fitness for particular purpose, merchantability, non-infringement of third-party rights, or any other warranty, express, implied, or statutory. Spansion assumes no liability for any damages of any kind arising out of the use of the information in this document.