Spansion® Analog and Microcontroller Products

The following document contains information on Spansion analog and microcontroller products. Although the document is marked with the name “Fujitsu”, the company that originally developed the specification, Spansion will continue to offer these products to new and existing customers.

Continuity of Specifications

There is no change to this document as a result of offering the device as a Spansion product. Any changes that have been made are the result of normal document improvements and are noted in the document revision summary, where supported. Future routine revisions will occur when appropriate, and changes will be noted in a revision summary.

Continuity of Ordering Part Numbers

Spansion continues to support existing part numbers beginning with “MB”. To order these products, please use only the Ordering Part Numbers listed in this document.

For More Information

Please contact your local sales office for additional information about Spansion memory, analog, and microcontroller products and solutions.

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.

Fujitsu Semiconductor Europe
Application Note

MCU-AN-300403-E-V12

FM3 FAMILY

32-BIT MICROCONTROLLER

MB9BFXXX

Implementation of GNU Tool Chain for FUJITSU Cortex-M3 MCUs

APPLICATION NOTE

Revision History

Date	Issue
2011-09-27	1.0: SAh, MWi; First version
2012-03-21	1.1; MWi; Notes for different devices/configurations added
2012-04-04	1.2; MWi; Software package updated

This document contains 143 pages.

Warranty and Disclaimer

The use of the deliverables (e.g. software, application examples, target boards, evaluation boards, starter kits, schematics, engineering samples of IC's etc.) is subject to the conditions of Fujitsu Semiconductor Europe GmbH ("FSEU") as set out in (i) the terms of the License Agreement and/or the Sale and Purchase Agreement under which agreements the Product has been delivered, (ii) the technical descriptions and (iii) all accompanying written materials.

Please note that the deliverables are intended for and must only be used for reference in an evaluation laboratory environment.

The software deliverables are provided on an as-is basis without charge and are subject to alterations. It is the user's obligation to fully test the software in its environment and to ensure proper functionality, qualification and compliance with component specifications.

Regarding hardware deliverables, FSEU warrants that they will be free from defects in material and workmanship under use and service as specified in the accompanying written materials for a duration of 1 year from the date of receipt by the customer.

Should a hardware deliverable turn out to be defect, FSEU's entire liability and the customer's exclusive remedy shall be, at FSEU´s sole discretion, either return of the purchase price and the license fee, or replacement of the hardware deliverable or parts thereof, if the deliverable is returned to FSEU in original packing and without further defects resulting from the customer's use or the transport. However, this warranty is excluded if the defect has resulted from an accident not attributable to FSEU, or abuse or misapplication attributable to the customer or any other third party not relating to FSEU or to unauthorised decompiling and/or reverse engineering and/or disassembling.

FSEU does not warrant that the deliverables do not infringe any third party intellectual property right (IPR). In the event that the deliverables infringe a third party IPR it is the sole responsibility of the customer to obtain necessary licenses to continue the usage of the deliverable.

In the event the software deliverables include the use of open source components, the provisions of the governing open source license agreement shall apply with respect to such software deliverables.

To the maximum extent permitted by applicable law FSEU disclaims all other warranties, whether express or implied, in particular, but not limited to, warranties of merchantability and fitness for a particular purpose for which the deliverables are not designated.

To the maximum extent permitted by applicable law, FSEU's liability is restricted to intention and gross negligence. FSEU is not liable for consequential damages.

Should one of the above stipulations be or become invalid and/or unenforceable, the remaining stipulations shall stay in full effect.

The contents of this document are subject to change without a prior notice, thus contact FSEU about the latest one.

Contents

1 Introduction

SCOPE OF THIS DOCUMENT

1.1 Description

This document describes the implementation of Yagarto tools as a compiler collection for the Eclipse platform for the Fujitsu Cortex-M3 microcontroller family.

The Yagarto tool chain has following features:

- No need of Cygwin, Windows executable are included
- Binutils, Newlib, and the GNU compiler collection is provided

Based on the platform Eclipse, the C/C++ Development Tooling (CDT) project provides a fully functional C and C++ Integrated Development Environment IDE.

CDT has following features:

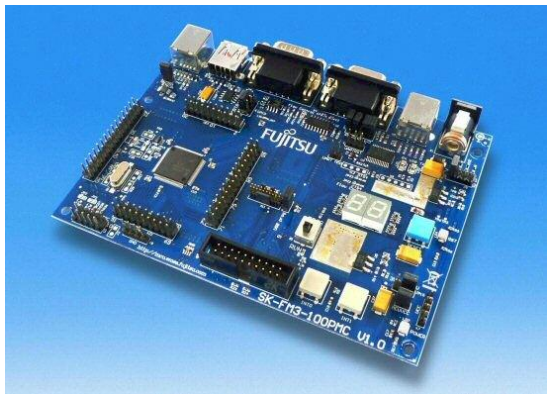
- Supports project creation and managed build for GNU tool chain.
- Code editor for the creation of makefile, assembler, C/C++ source/include, and linker script files.
- Visual debugging tools, including memory, registers, and disassembly views.
- Supports Open On-Chip Debugger and J-Link GDB Server as external debugging tools.

This document describes at first the installation procedure for all tools and software packages needed to realize a development environment based on Eclipse platform and Yagarto tool chain.
For this the first part of this document includes the installation specification of the following software packages:

- Yagarto tools
- OpenOCD
- J-Link GDB Server
- Java JRE
- Eclipse platform
- C/C++ Development Tooling CDT

In order to test the IDE installed during this description, the following board is used exemplarily:

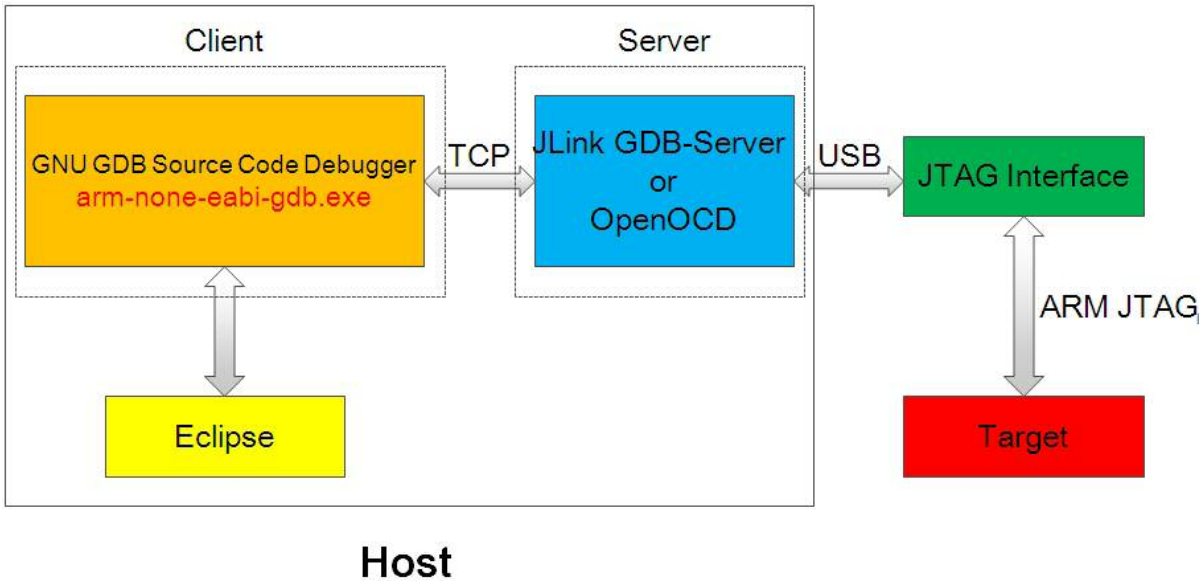
- Fujitsu Starter Kit for FM3 MCU MB9BF506N: SK-FM3-100PMC



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.

This document demonstrates two different debugging methods:



1. Debugging via OpenOCD server using the JTAG interface “KT-Link” (based on FT2232D)
2. Debugging via JLink GDB server using the JTAG interface “J-Link-ARM”

1.2.1 KT-Link

An FT2232D JTAG to USB based solution is the KT-Link from Kris-Tech

This interface has the following features:

- USB High Speed (480 MHz) connection to host PC
- Wide range of target voltage: 1.65 to 5.5V
- Hardware support for SWD and SWV
- Standard 2x10 pin ARM JTAG connector
- LEDs
- No power supply required, powered through USB
- Virtual RS232 port with all DB9 connector signals
- Serial port works in RS-232 mode or wide range of voltage (5V, 3.3V, 2.5V, 1.8V) selectable by user
- Target power supply with overload protection

The JTAG interface “KT-Link” can be purchased from the website:
www.shop.kristech.eu

1.2.2 J-Link

Another JTAG interface used with the J-Link GDB server is the “J-Link” for ARM Processors. This interface is also a product of the company Segger.

The Segger “J-Link” has the following features:

- USB powered JTAG emulator for Cortex-M devices
- License for J-Link GDB server
- License for Flash download
- License for the flash breakpoints
- Support download in RAM and Flash
- Support an unlimited number of BP in Flash
- SWD/SWV
- Voltage range: 1.2-3.3V, 5V

For more information about the “J-Link” interface:
http://www.segger.com/cms/jlink.html

1.3 Download the tutorial material

Before starting this tutorial, first download the tutorial project source and OpenOCD configuration files. These files are contained in the software package of this application note.

2 YAGARTO

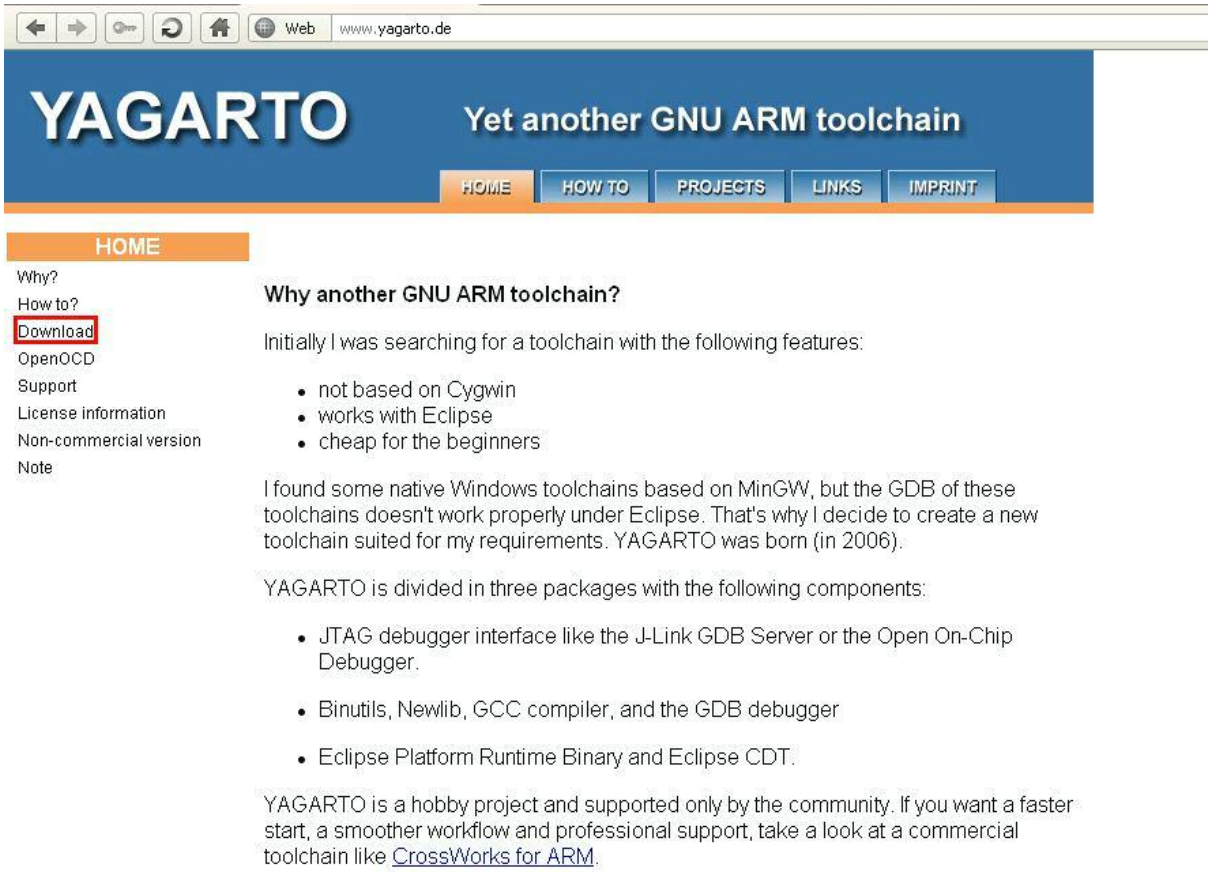
THIS CHAPTER DESCRIBES HOW TO INSTALL THE YAGARTO SOFTWARE PACKAGE

2.1 Yet another GNU ARM Tool Chain

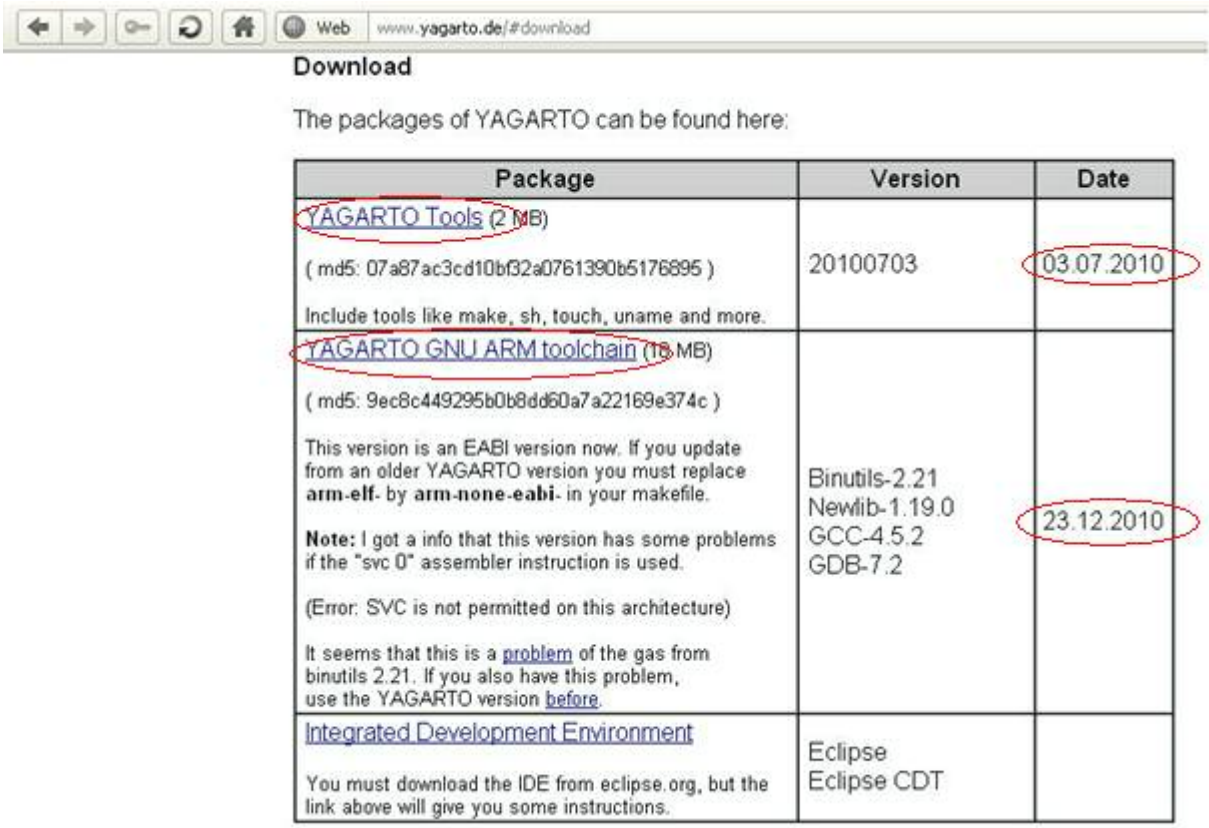
There are a number of pre-built GNU ARM compiler toolsets available on the web. This application note uses the YAGARTO pre-built ARM compiler tool suite developed by Michael Fischer. This version of the GNU compiler toolset for ARM has been natively compiled for the Intel/Windows platform.
Except the 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: www.yagarto.de



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



Download

The packages of YAGARTO can be found here:

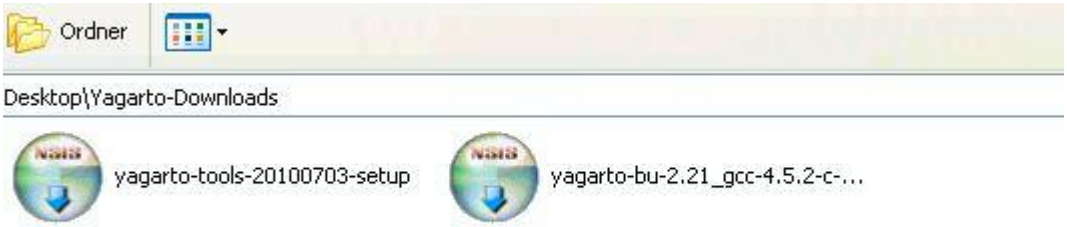
Package	Version	Date
YAGARTO Tools (2 MB) (md5: 07a87ac3cd10bf32a0761390b5176895) Include tools like make, sh, touch, uname and more.	20100703	03.07.2010
YAGARTO GNU ARM toolchain (18 MB) (md5: 9ec8c449295b0b8dd60a7a22169e374c) This version is an EABI version now. If you update from an older YAGARTO 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.

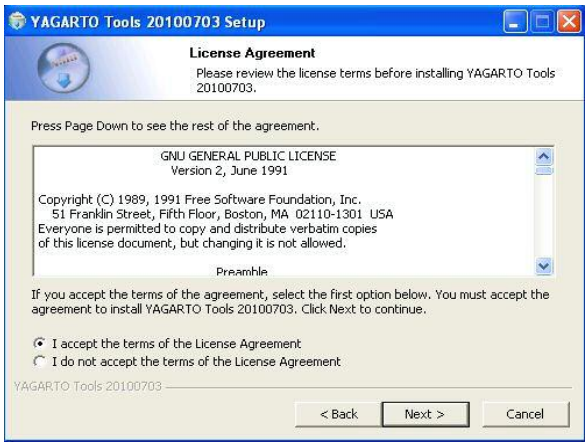
Only the first two packages are recommended at this moment, because the installation description of the third package “Eclipse IDE” and “Eclipse CDT” will be separately explained in detail in chapter 6.

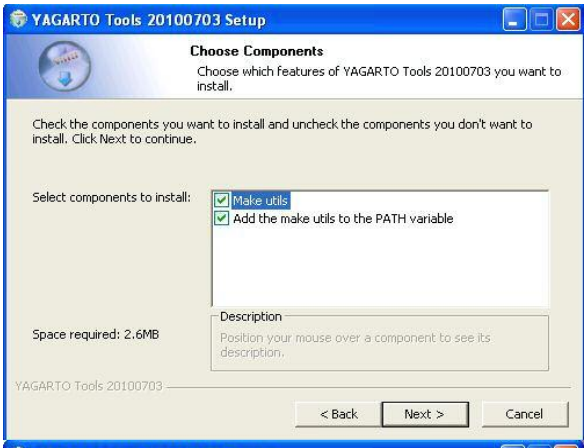
2.3 Installing Yagarto tools

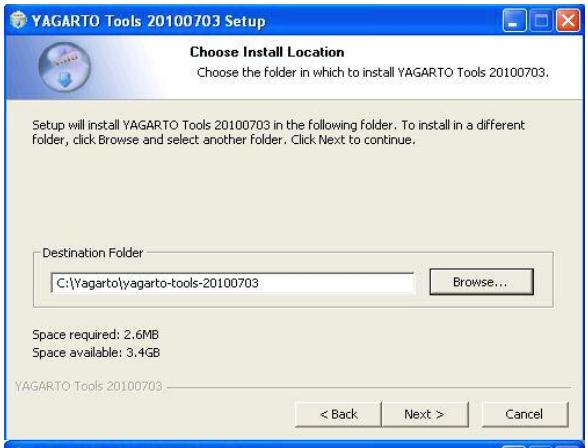
After saving the package, e.g. in the temporary folder “Yagarto-Downloads”, the installation procedure of these tools can be started.

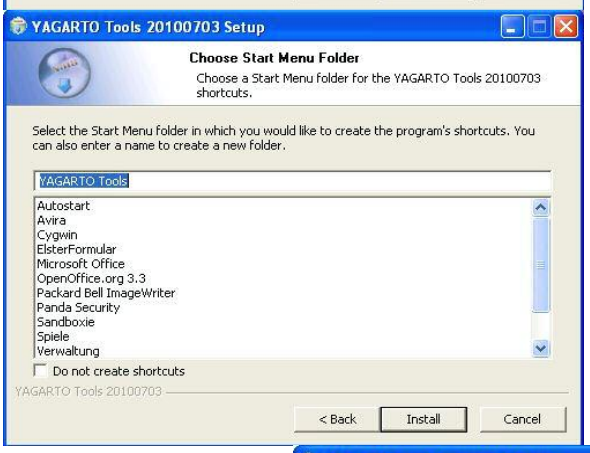


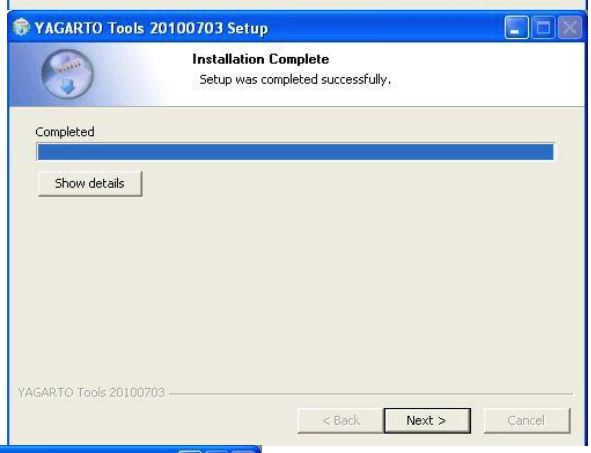
After downloading start the installation of the make utility tools “yagarto-tools-20100703-setup” or newer.





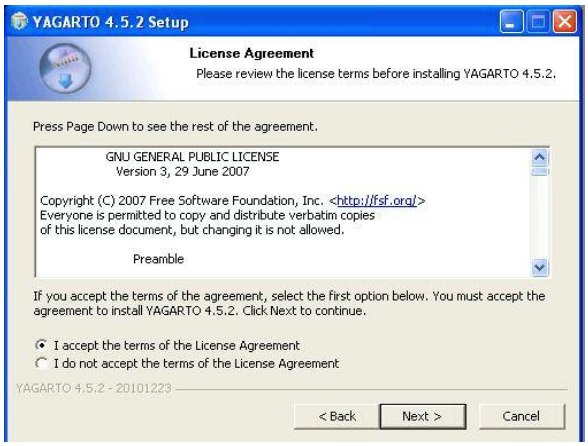


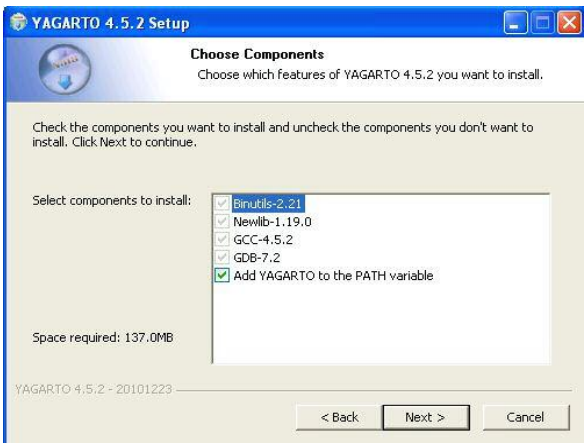


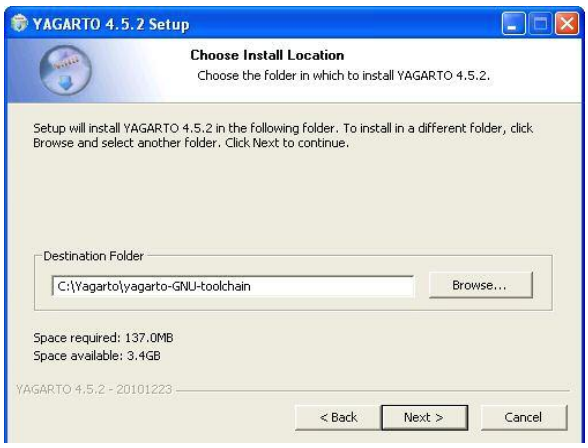


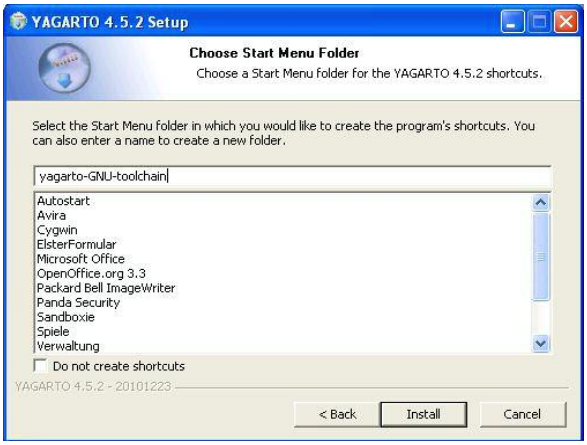
Next following the installation steps for the ARM compiler toolset “yagarto-bu-2.21_gcc-4.5.2-c-c++_nl-1.19.0_gdb-7.2_eabi_20101223” or newer.

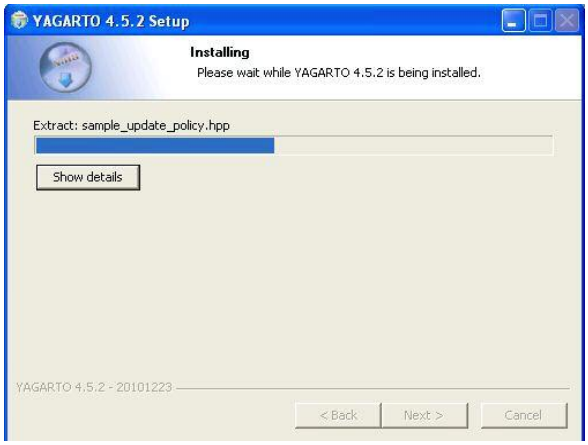

YAGARTO 4.5.2 Setup
Welcome to the YAGARTO 4.5.2 Setup Wizard
This wizard will guide you through the installation of YAGARTO 4.5.2.
It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.
Click Next to continue.
Next >
Cancel

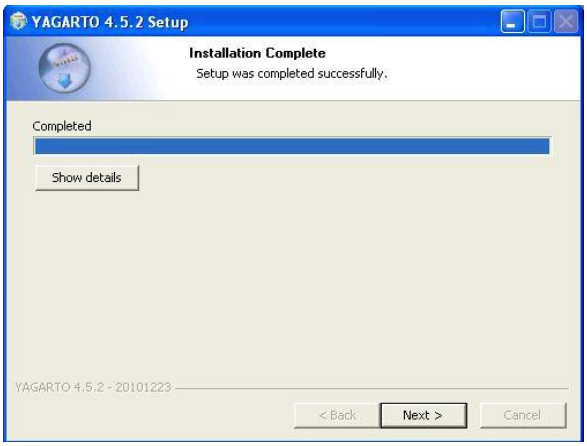

YAGARTO 4.5.2 Setup
License Agreement
Please review the license terms before installing YAGARTO 4.5.2.
Press Page Down to see the rest of the agreement.
GNU GENERAL PUBLIC LICENSE
Version 3, 29 June 2007
Copyright (C) 2007 Free Software Foundation, Inc. <http://fsf.org/>
Everyone is permitted to copy 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 4.5.2. Click Next to continue.
I accept the terms of the License Agreement
I do not accept the terms of the License Agreement
YAGARTO 4.5.2 - 20101223
< Back
Next >
Cancel


YAGARTO 4.5.2 Setup
Choose Components
Choose which features of YAGARTO 4.5.2 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:
Binutils-2.21
Newlib-1.19.0
GCC-4.5.2
GDB-7.2
Add YAGARTO to the PATH variable
Space required: 137.0MB
YAGARTO 4.5.2 - 20101223
< Back
Next >
Cancel


YAGARTO 4.5.2 Setup
Choose Install Location
Choose the folder in which to install YAGARTO 4.5.2.
Setup will install YAGARTO 4.5.2 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-GNU-toolchain
Browse...
Space required: 137.0MB
Space available: 3.4GB
YAGARTO 4.5.2 - 20101223
< Back
Next >
Cancel


YAGARTO 4.5.2 Setup
Choose Start Menu Folder
Choose a Start Menu folder for the YAGARTO 4.5.2 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-GNU-toolchain
Autostart
Avira
Cygwin
ElsterFormular
Microsoft Office
OpenOffice.org 3.3
Packard Bell ImageWriter
Panda Security
Sandboxie
Spiele
Verwaltung
Do not create shortcuts
YAGARTO 4.5.2 - 20101223
< Back
Install
Cancel


YAGARTO 4.5.2 Setup
Installing
Please wait while YAGARTO 4.5.2 is being installed.
Extract: sample_update_policy.hpp
Show details
YAGARTO 4.5.2 - 20101223
< Back
Next >
Cancel


YAGARTO 4.5.2 Setup
Installation Complete
Setup was completed successfully.
Completed
Show details
YAGARTO 4.5.2 - 20101223
< Back
Next >
Cancel


YAGARTO 4.5.2 Setup
Completing the YAGARTO 4.5.2 Setup Wizard
YAGARTO 4.5.2 has been installed on your computer.
Click Finish to close this wizard.
Visit the YAGARTO site for the latest news.
< Back
Finish
Cancel

3 OpenOCD

HOW TO INSTALL OPENOCD

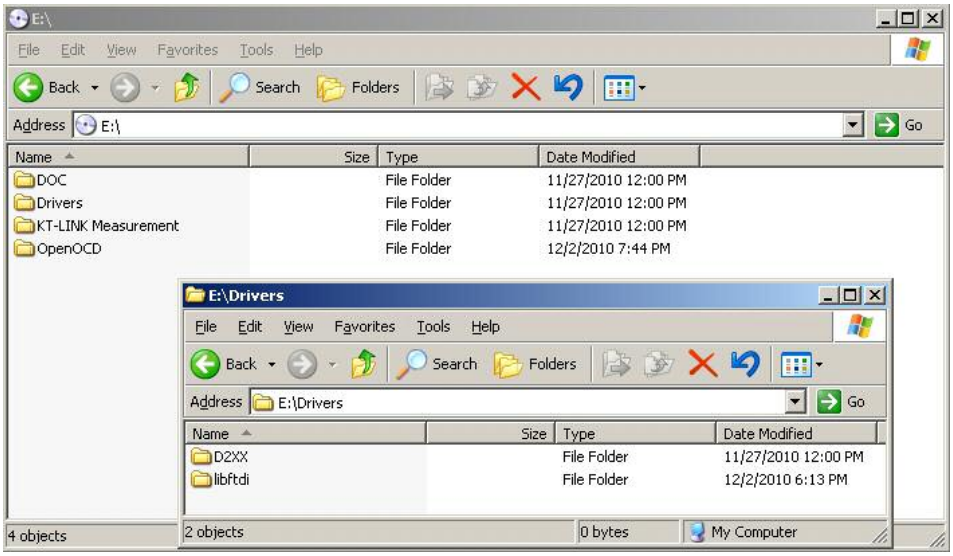
3.1 Open On-Chip Debugger

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.

In this application note the OpenOCD with the JTAG dongle ‘”KT-Link” is used exemplarily.

KT-Link can be driven with the FTDI driver “D2xx” or with the open source LibUSB driver “libftdi”. Both drivers are included on the CD delivered with KT-Link dongle.



Before installing and running OpenOCD, it must be taken care, which driver is used for the JTAG dongle.

When the LibUSB driver is used, the Windows installer program for the latest OpenOCD version can be downloaded from the website:

http://www.freddiechopin.info/index.php/en/download/category/4-openocd

Since version “0.4.0” presented on this website the LibUSB driver is included and can be installed.

If the FTDI driver is used, OpenOCD does not deliver any Windows installer, because the FTDI driver is proprietary and closed software. Therefore it is needed to configure and compile the OpenOCD manually together with this driver.

In this chapter the installation procedure of OpenOCD for JTAG dongle using LibUSB driver and the configuration procedure of OpenOCD to use dongle with FTDI driver is described.

3.2 Using Openocd with FTDI driver

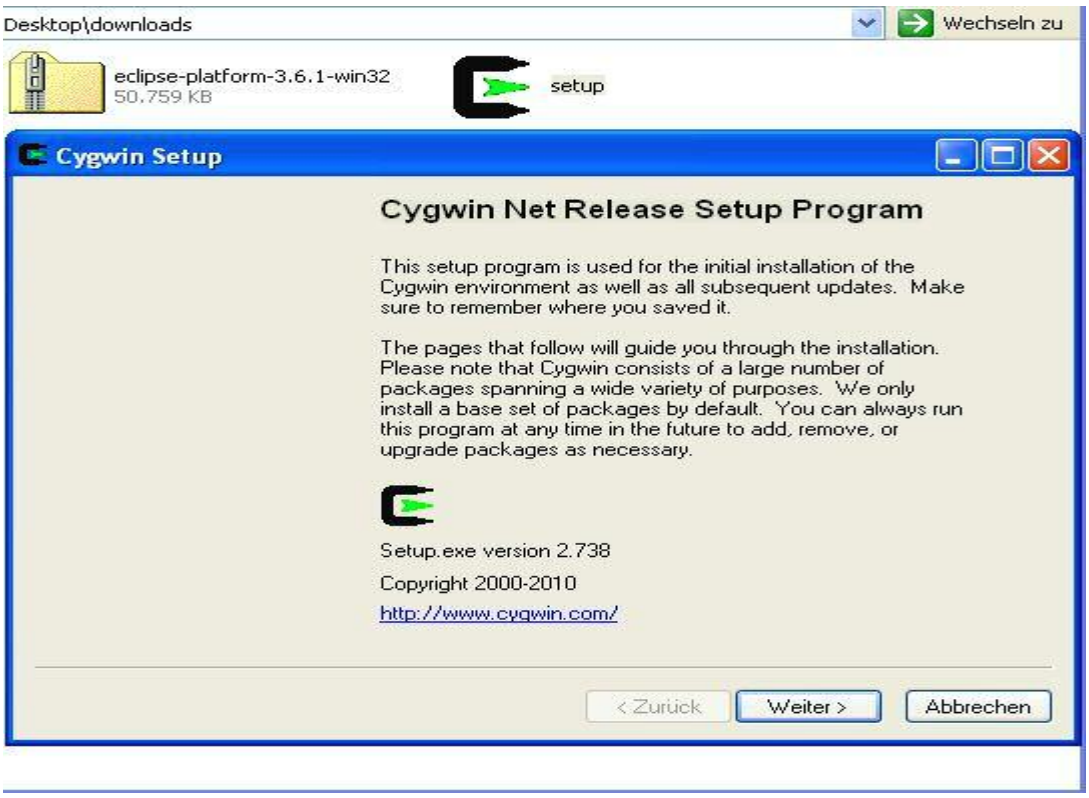
As described above the OpenOCD sources does not include the FTDI driver, so it is needed for using OpenOCD with KT-Link on the basis of the “D2xx” driver to configure and install the OpenOCD with this driver.

To use Openocd 0.4.0 (or higher) for Windows with FTDI driver based JTAG interfaces, it is needed to build OpenOCD for Windows OS using Cygwin.

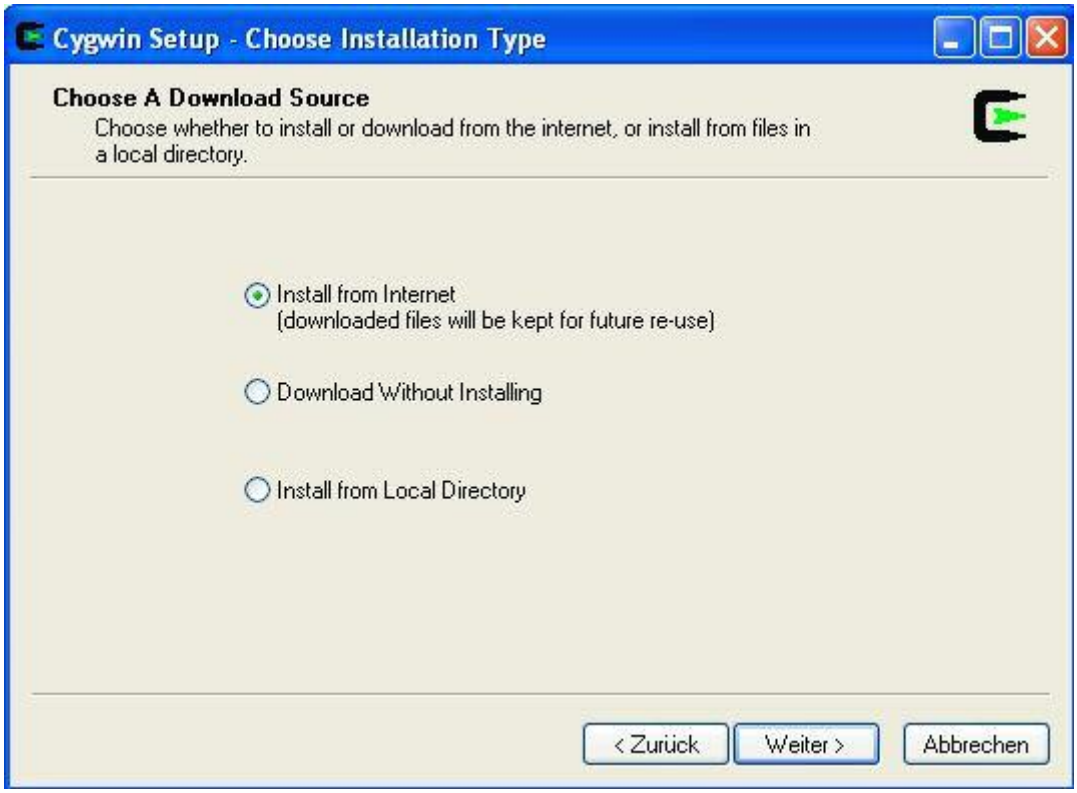
3.2.1 Installation of Cygwin

Cygwin can be installed or updated with the installer program “setup.exe”. This setup file can be downloaded from the website: http://cygwin.com/install.html

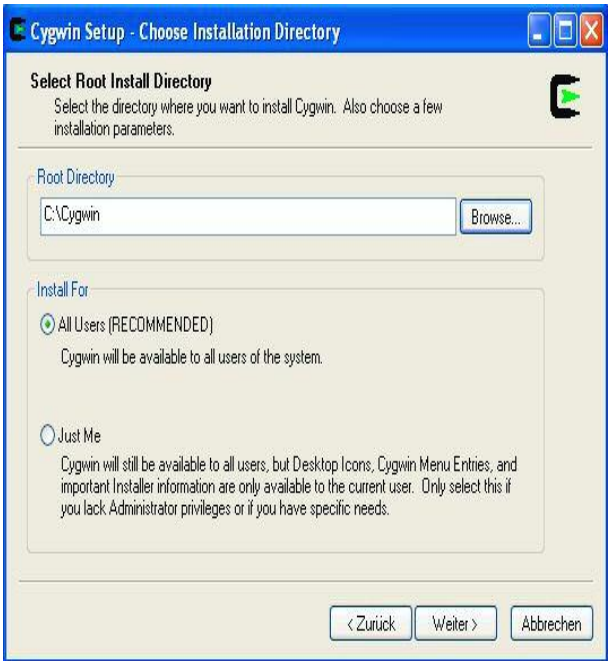
After saving the Cygwin setup file, run this program to install Cygwin.

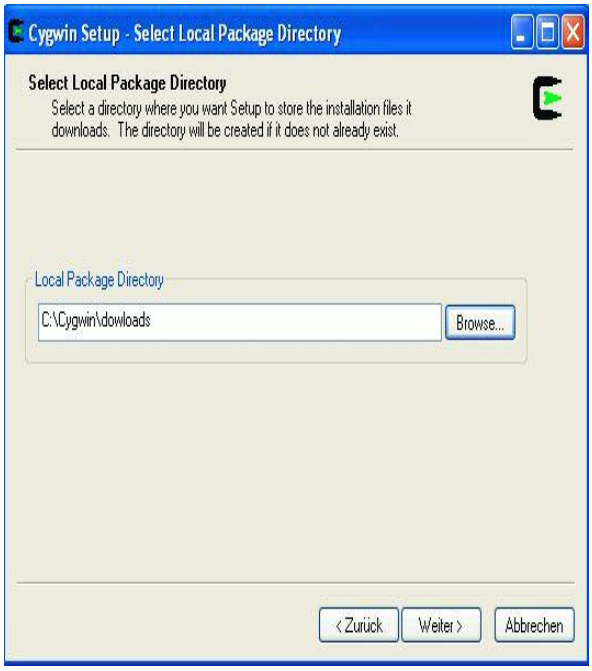


In the next step of the installation setup, choose the installation method.

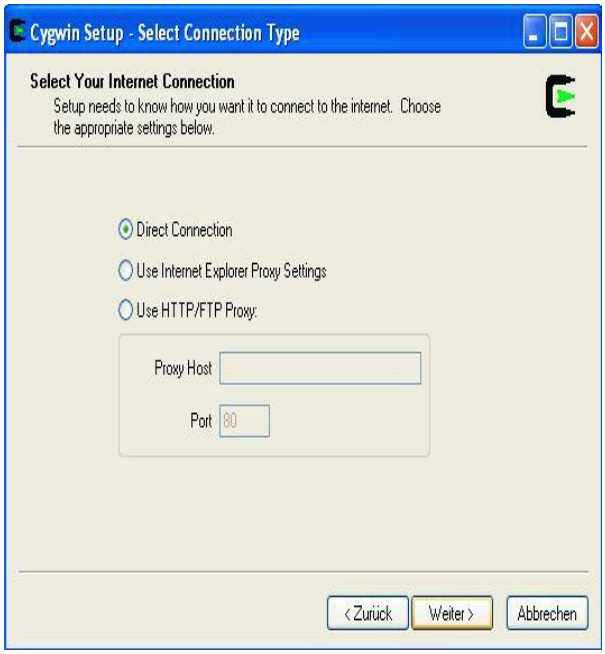


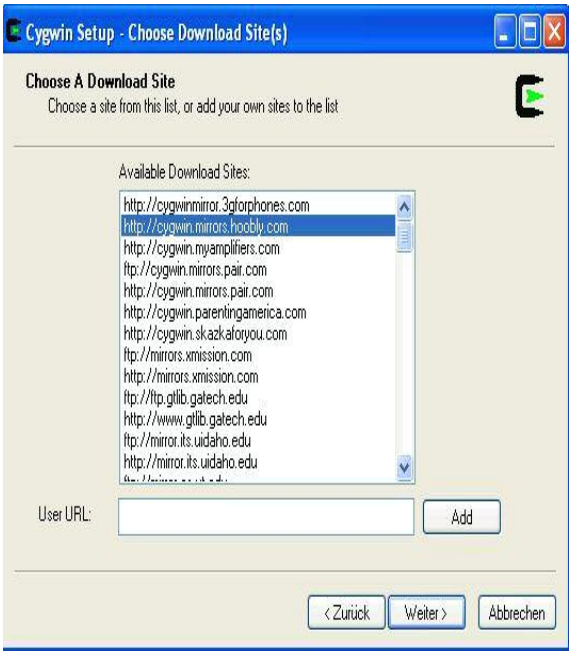
The next two steps of the installation create the root installation directory and the local directory, where the setup files will be saved.





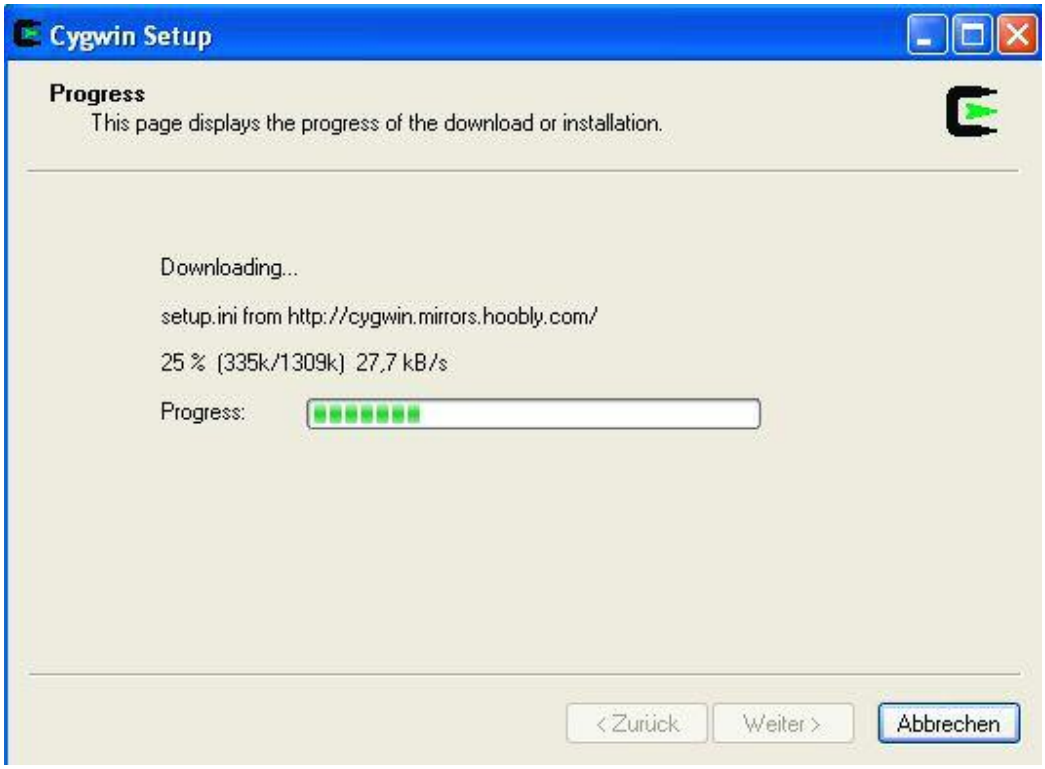
After this, first configure the network setting and choose an “http” or “ftp” mirror to get the Cygwin packages.





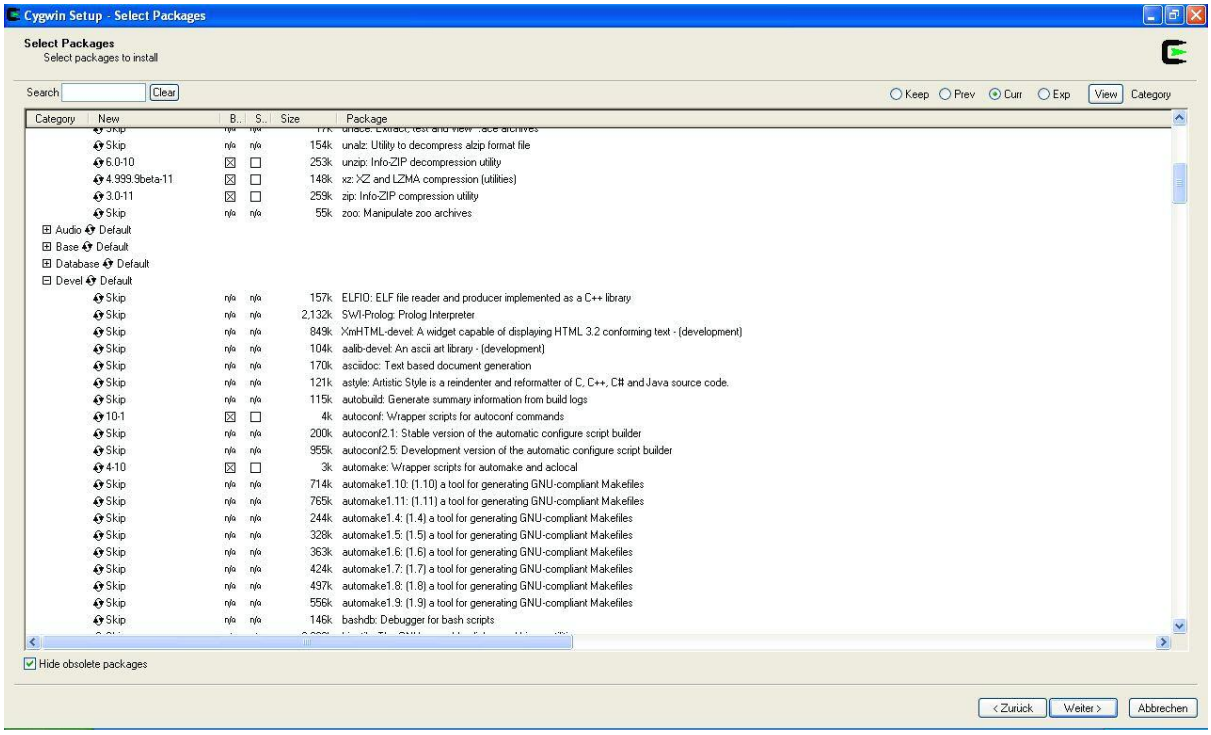
Note, if your PC is connected to a company's proxy, check *Use Internet Explorer Proxy Setting* or type the settings manually in the 3rd radio button's text boxes.

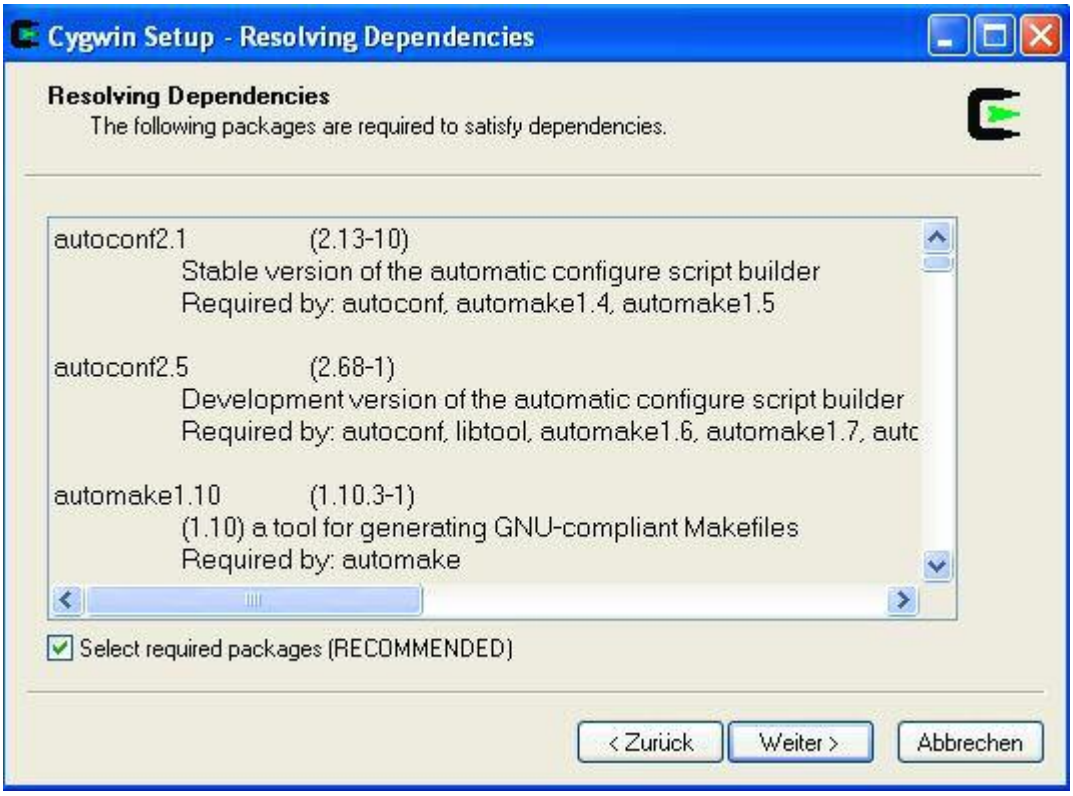
In the next step the download of the setup will start.

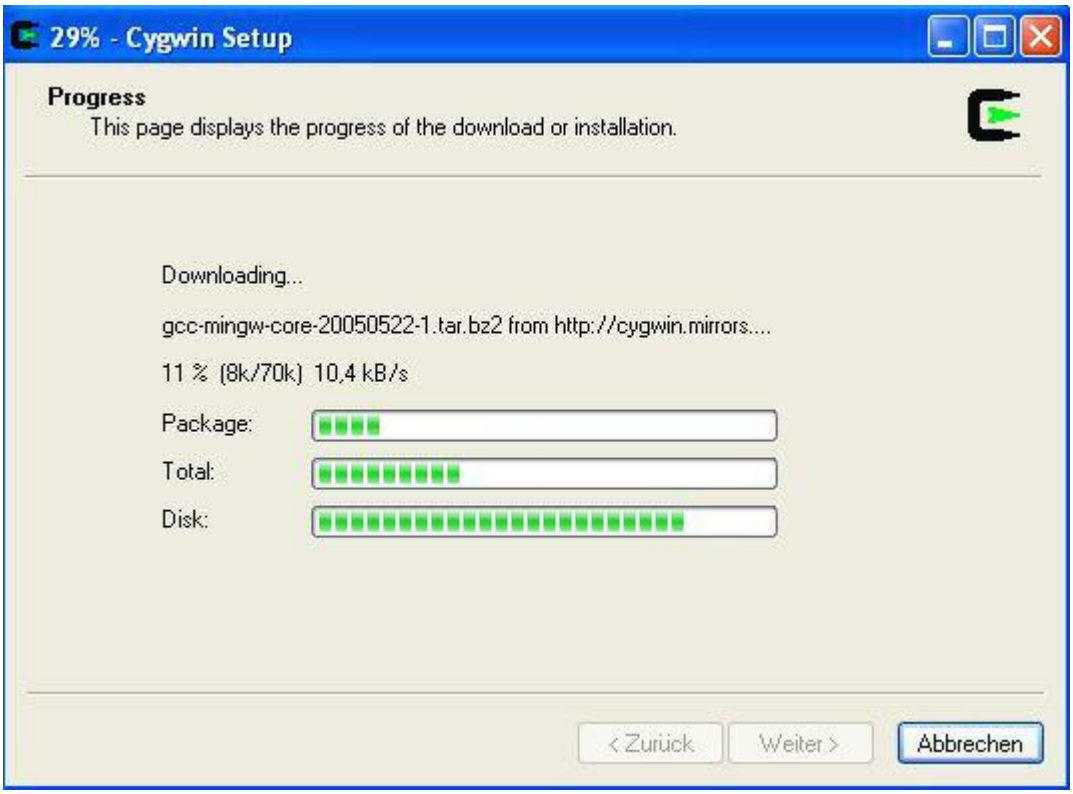


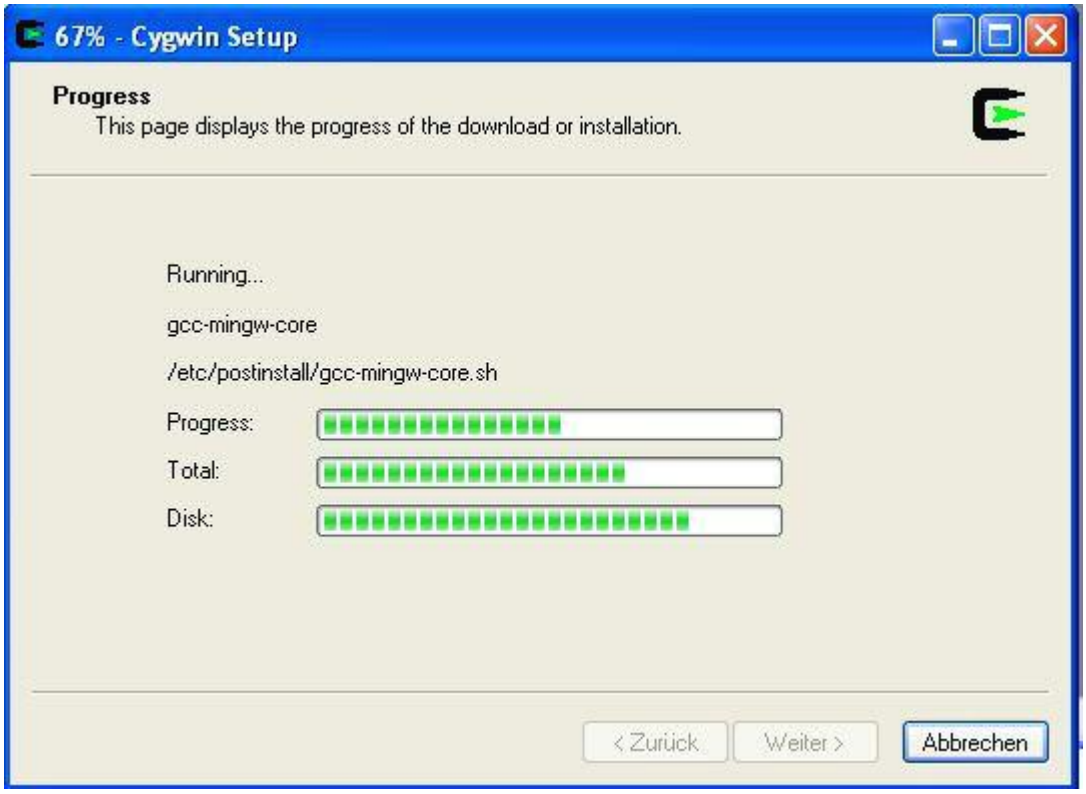
After download select the required package used to configure and compile OpenOCD. The following packages must be explicitly selected to build OpenOCD:

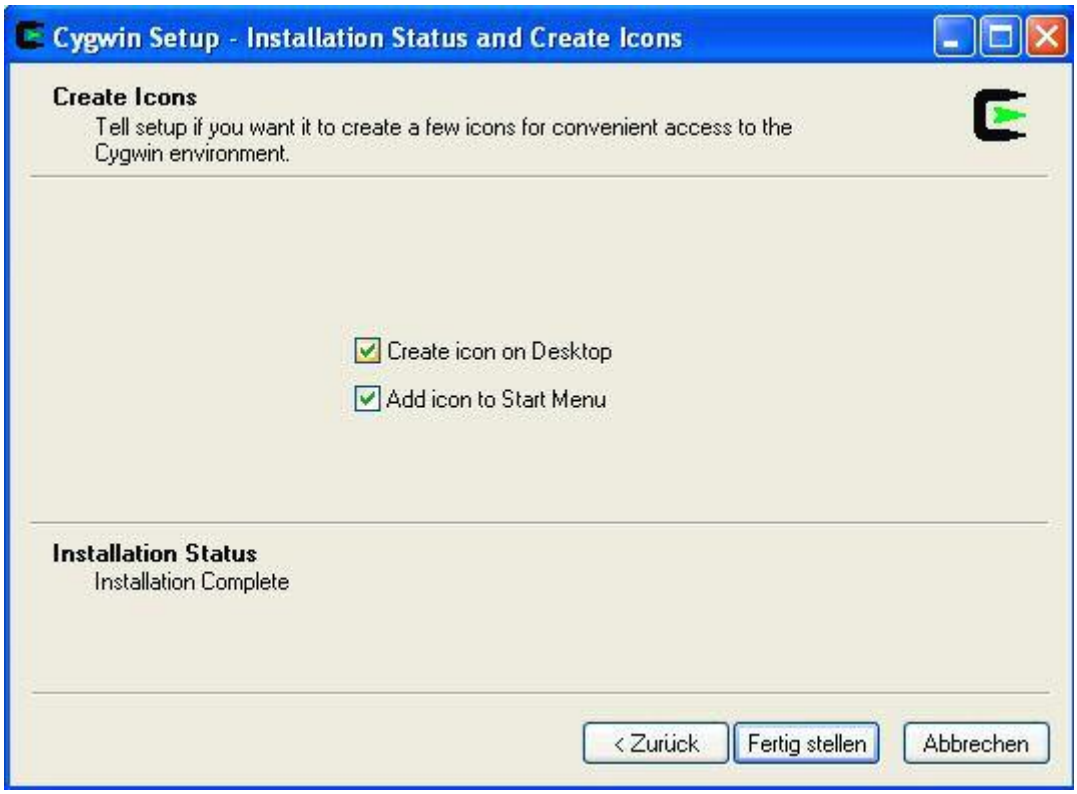
- All
  - Archive
    - *unzip*: Info-ZIP decompression utility; for FTDI driver ZIP file
- All
  - Devel
    - *autoconf*: Wrapper scripts for autoconf commands
  - Devel
    - *automake*: Wrapper scripts for automake and aclocals
  - Devel
    - *gcc*: C compiler upgrade helper
  - Devel
    - *libtool*: A shared library generation tool
  - Devel
    - *make*: The GNU version of the 'make' utility
- All
  - Publishing
    - *tetex*: The TeX text formatting system (install helper; to generate PDF documentation)
- All
  - Publishing
    - *tetex-extra*: The TeX text formatting system (extra libraries; to generate PDF documentation)




Cygwin Setup - Resolving Dependencies
Resolving Dependencies
The following packages are required to satisfy dependencies.
autoconf2.1 (2.13-10)
Stable version of the automatic configure script builder
Required by: autoconf, automake1.4, automake1.5
autoconf2.5 (2.68-1)
Development version of the automatic configure script builder
Required by: autoconf, libtool, automake1.6, automake1.7, autc
automake1.10 (1.10.3-1)
(1.10) a tool for generating GNU-compliant Makefiles
Required by: automake
Select required packages (RECOMMENDED)
< Zurück
Weiter >
Abbrechen

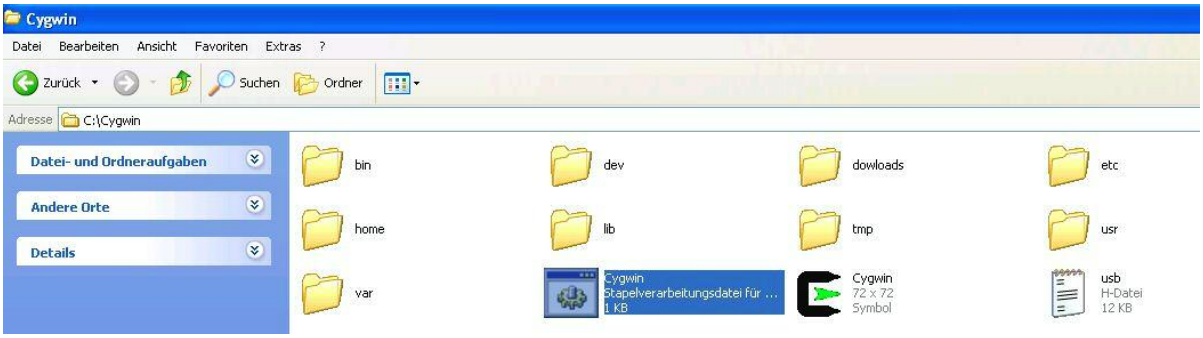

29% - Cygwin Setup
Progress
This page displays the progress of the download or installation.
Downloading...
gcc-mingw-core-20050522-1.tar.bz2 from http://cygwin.mirrors....
11 % (8k/70k) 10,4 kB/s
Package:
Total:
Disk:
< Zurück
Weiter >
Abbrechen





With this last confirmation step the installation of Cygwin is done.

In the next illustration the root installation of Cygwin can be seen.

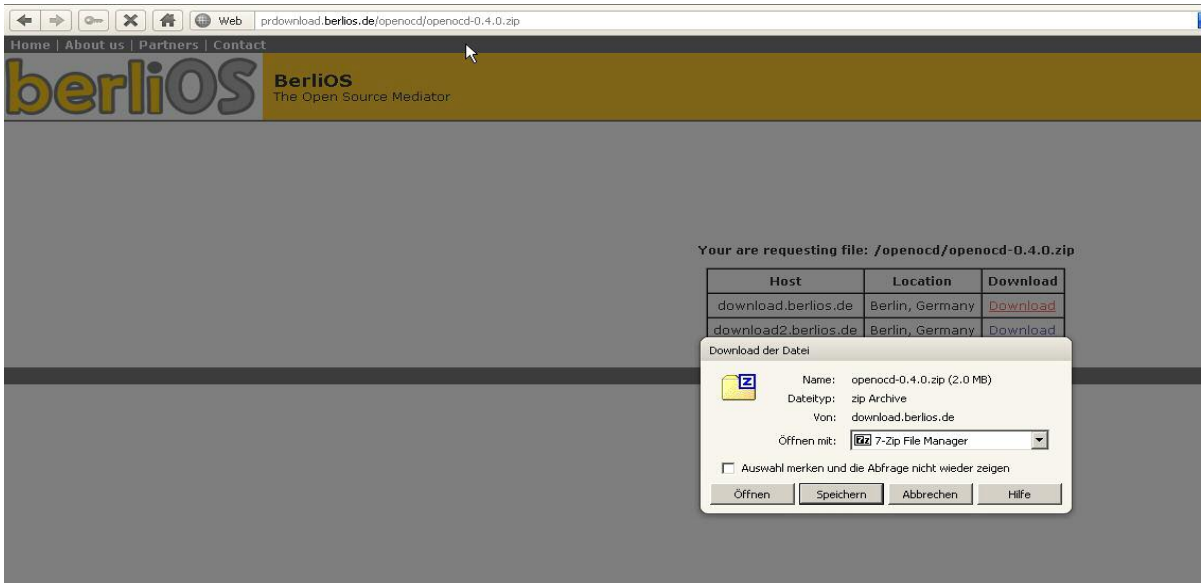


Via the batch file *cygwin.bat* the Cygwin environment can be started to configure and compile OpenOCD.

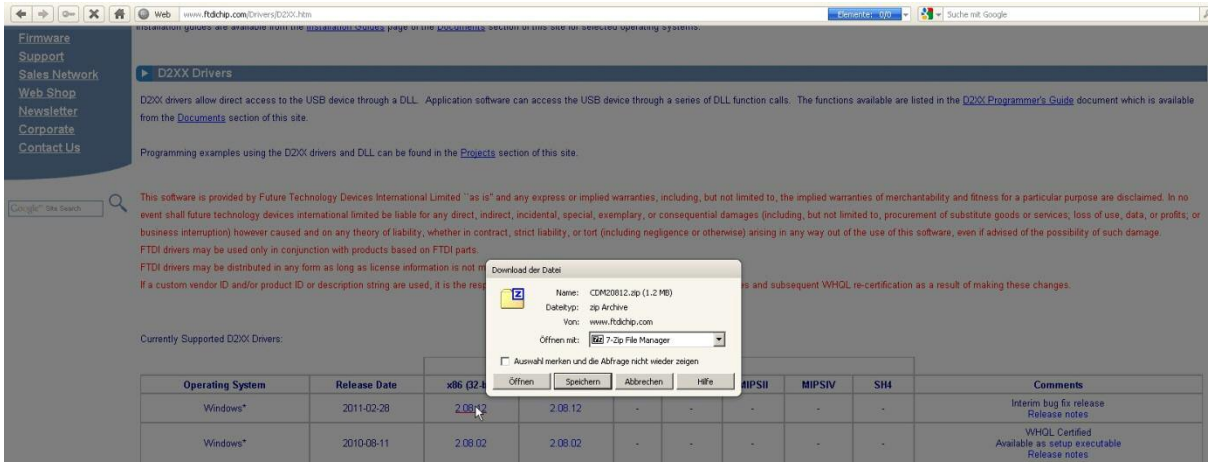
3.2.2 Download of OpenOCD source and FTDI driver

The Cygwin tools are needed for configuring and compiling OpenOCD to work with the driver of the FTDI chip delivered. For this reason we need first to procure the native source of OpenOCD and the FTDI driver.

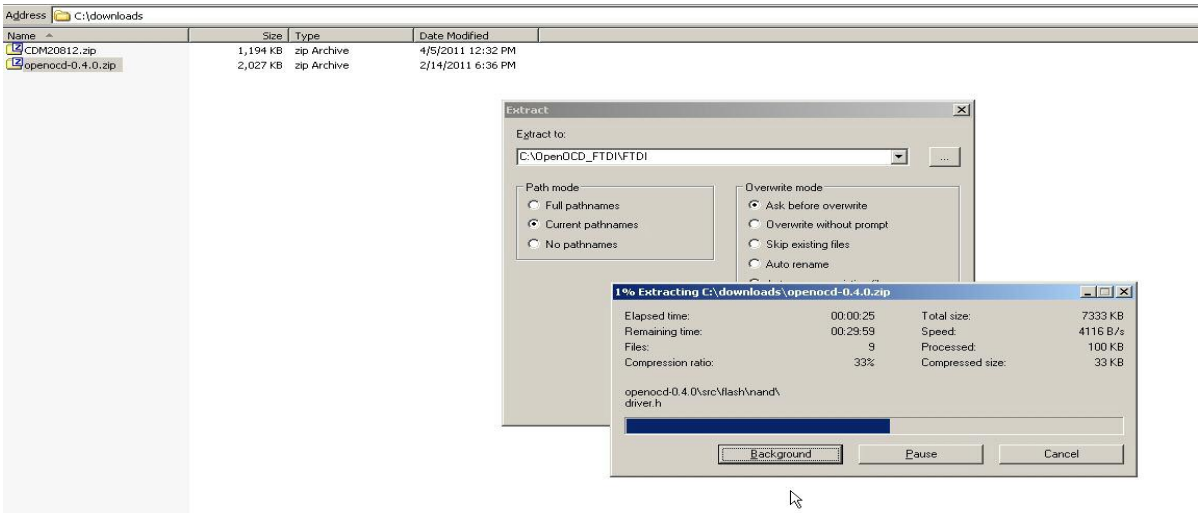
The latest version of the native sources of OpenOCD can be downloaded from the website: http://openocd.berlios.de/web/



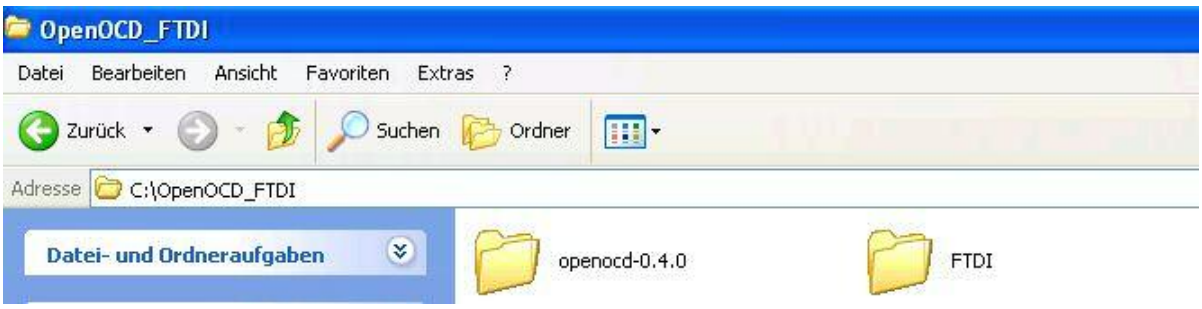
To get the driver currently available for the FTDI devices, you need only to save and later to extract the zip file available at the website: http://www.ftdichip.com/Drivers/D2XX.htm



When the OpenOCD and FTDI zip files are saved somewhere on the PC, the next step is to extract both files in a directory of your choice e.g. *C:\OpenOCD_FTDI*.



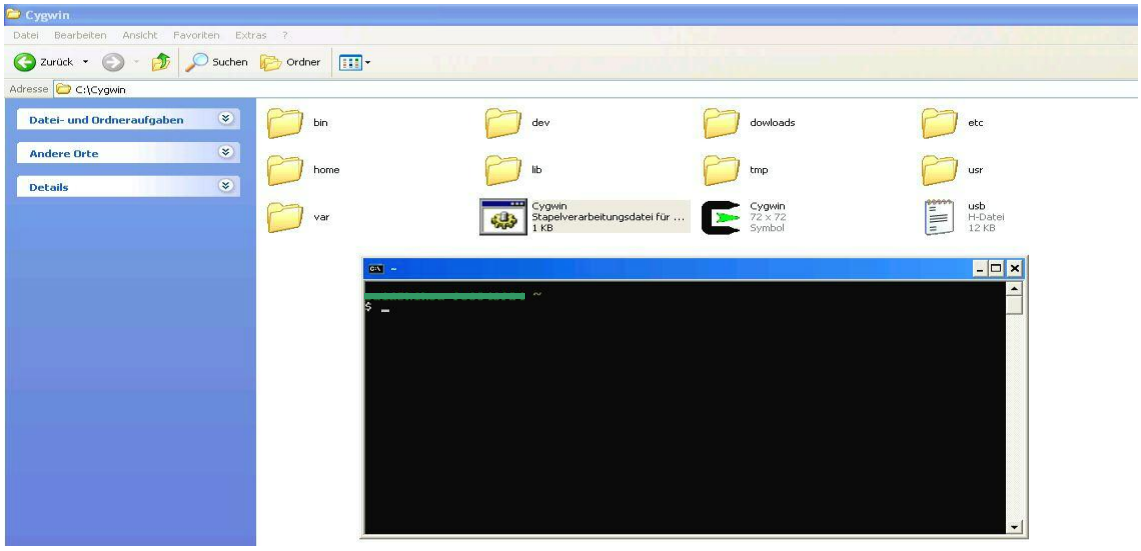
The native source files of OpenOCD and the driver for the FTDI device are now located in the directory *C:\OpenOCD_FTDI*.



This directory is needed temporarily during the next parts of this documentation.

3.2.3 Configuration and compilation of OpenOCD with FTDI driver

The configuration and compilation of OpenOCD sources will be done on the Cygwin environment. From the root installation of Cygwin run the batch file *cygwin.bat* and start this program.



Now open the folder openocd-0.4.0, where the native source of OpenOCD was extracted from the directory *C:\OpenOCD_FTDI*.

The folder openocd-0.4.0 contains the configuration script of OpenOCD. We can now start this script with the command `./configure` and configure OpenOCD to use the FTDI driver source extracted on the folder *FTDI* from the directory *C:\OpenOCD_FTDI*.

The following configuration options are recommended:

- `--enable-maintainer-mode`
- `--disable-werror`
- `--disable-shared`
- `--enable-ft2232_ftd2xx`
- `--with-ftd2xx-win32-zipdir=C:/OpenOCD_FTDI/FTDI`
- `CC="gcc-3 -mno-cygwin -L/usr/lib/mingw -L/usr/lib/w32api -I/usr/include/mingw -I/usr/include/w32api"`

You also can create a bash file (e.g. *myconfig*) containing these arguments (all in one line) like:

```
./configure --enable-maintainer-mode --disable-werror --disable-shared -
-enable-ft2232_ftd2xx --with-ftd2xx-win32-zipdir= C:/OpenOCD_FTDI/FTDI
CC="gcc-3 -mno-cygwin -L/usr/lib/mingw -L/usr/lib/w32api -
I/usr/include/mingw -I/usr/include/w32api"
```

Then execute it in the Cygwin shell with `bash myconfig`.

Note, that *gcc-3.exe* in the Cygwin's *bin/* directory has to be used for compiling. If the configuration fails, try to add the whole path to *gcc-3* in the `CC` variable above.


```
/cygdrive/c/OpenOCD_FTDI/openocd-0.4.0
                    /cygdrive/c/OpenOCD_FTDI/openocd-0.4.0
$ ./configure --enable-maintainer-mode --disable-werror --disable-shared --enab
le-ft2232_ftd2xx --with-ftd2xx-win32-zipdir=C:/OpenOCD_FTDI/FTDI CC="gcc -mno-c
ygwin" CFLAGS="-O2 -Wall"
checking for a BSD-compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
checking for a thread-safe mkdir -p... /usr/bin/mkdir -p
checking for gawk... gawk
checking whether make sets $(MAKE)... yes
checking whether to enable maintainer-specific portions of Makefiles... yes
checking for gcc... gcc -mno-cygwin
checking for C compiler default output file name... a.exe
checking whether the C compiler works... yes
checking whether we are cross compiling... no
checking for suffix of executables... .exe
checking for suffix of object files... o
checking whether we are using the GNU C compiler... yes
checking whether gcc -mno-cygwin accepts -g... yes
checking for gcc -mno-cygwin option to accept ISO C89... none needed
checking for style of include used by make... GNU
checking dependency style of gcc -mno-cygwin... gcc3
checking for gcc -mno-cygwin option to accept ISO C99... -std=gnu99
checking whether gcc -mno-cygwin -std=gnu99 and cc understand -c and -o together
...
```

When the configuration is done, start the compilation of OpenOCD with `make`.

If the make process fails, remove all files (**not** the directories) from the OpenOCD root and use the files from the application note's software package. Reconfigure as described above and try again to `make`.

```
                    /cygdrive/c/OpenOCD_FTDI/openocd-0.4.0
$ make
make  all-recursive
make[1]: Entering directory `/cygdrive/c/OpenOCD_FTDI/openocd-0.4.0'
Making all in src
make[2]: Entering directory `/cygdrive/c/OpenOCD_FTDI/openocd-0.4.0/src'
make  all-recursive
make[3]: Entering directory `/cygdrive/c/OpenOCD_FTDI/openocd-0.4.0/src'
Making all in helper
make[4]: Entering directory `/cygdrive/c/OpenOCD_FTDI/openocd-0.4.0/src/helper'
make  all-am
make[5]: Entering directory `/cygdrive/c/OpenOCD_FTDI/openocd-0.4.0/src/helper'
```

After that the compilation procedure is done, use the command `strip` can be used to remove unnecessary information from the generated executable binary program *OpenOCD.exe* located in the folder *src* of the directory *openocd-0.4.0*.

```
/cygdrive/c/OpenOCD_FTDI/openocd-0.4.0
mv -f .deps/main.Tpo .deps/main.Po
/bin/sh ../libtool --tag=CC   --mode=link gcc -mno-cygwin -std=gnu99  -O2 -Wall
-I/cygdrive/c/OpenOCD_FTDI/FTDI -Wall -Wstrict-prototypes -Wformat-security -Wex
tra -Wno-unused-parameter -Wbad-function-cast -Wcast-align -Wredundant-decls  -L
/cygdrive/c/OpenOCD_FTDI/FTDI/i386 -o openocd.exe main.o libopenocd.la -lftd2xx
libtool: link: gcc -mno-cygwin -std=gnu99 -O2 -Wall -I/cygdrive/c/OpenOCD_FTDI/F
TDI -Wall -Wstrict-prototypes -Wformat-security -Wextra -Wno-unused-parameter -W
bad-function-cast -Wcast-align -Wredundant-decls -o openocd.exe main.o  -L/cygdr
ive/c/OpenOCD_FTDI/FTDI/i386 ./.libs/libopenocd.a -lws2_32 -lftd2xx
make[4]: Leaving directory `/cygdrive/c/OpenOCD_FTDI/openocd-0.4.0/src'
make[3]: Leaving directory `/cygdrive/c/OpenOCD_FTDI/openocd-0.4.0/src'
make[2]: Leaving directory `/cygdrive/c/OpenOCD_FTDI/openocd-0.4.0/src'
Making all in doc
make[2]: Entering directory `/cygdrive/c/OpenOCD_FTDI/openocd-0.4.0/doc'
make[2]: Nothing to be done for `all'.
make[2]: Leaving directory `/cygdrive/c/OpenOCD_FTDI/openocd-0.4.0/doc'
make[2]: Entering directory `/cygdrive/c/OpenOCD_FTDI/openocd-0.4.0'
make[2]: Leaving directory `/cygdrive/c/OpenOCD_FTDI/openocd-0.4.0'
make[1]: Leaving directory `/cygdrive/c/OpenOCD_FTDI/openocd-0.4.0'

                    /cygdrive/c/OpenOCD_FTDI/openocd-0.4.0
$ strip -s src/openocd.exe

                    /cygdrive/c/OpenOCD_FTDI/openocd-0.4.0
$ exit
```

With the last command `exit` the Cygwin environment is quit.

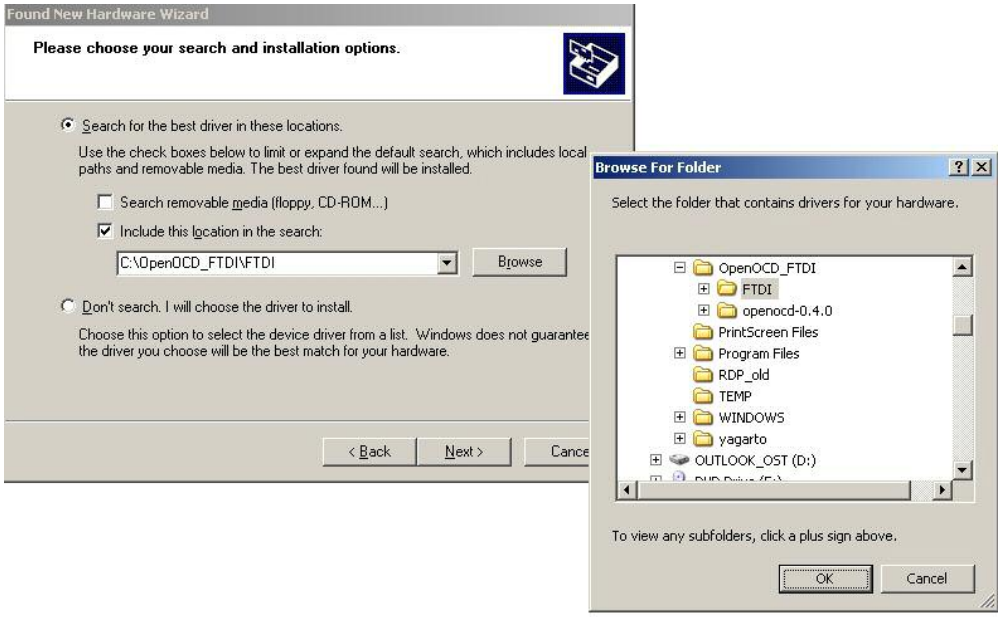
3.3 Test of OpenOCD configured for FTDI driver

After configuring and installing of OpenOCD now run it as a daemon sever to test, whether the server and the target are connected. For the connection tool use the JTAG dongle “KT-Link” and install its drivers.

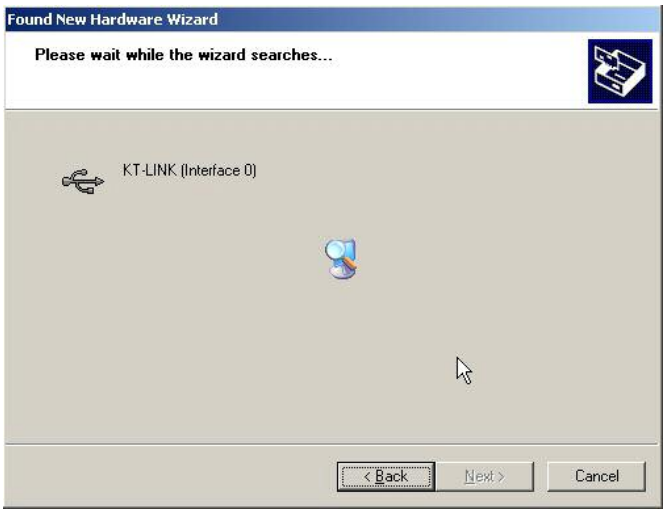
3.3.1 Installation of FTDI drivers for the JTAG dongle

After connecting the JTAG dongle over USB to the computer Windows OS reports that a new hardware was detected.

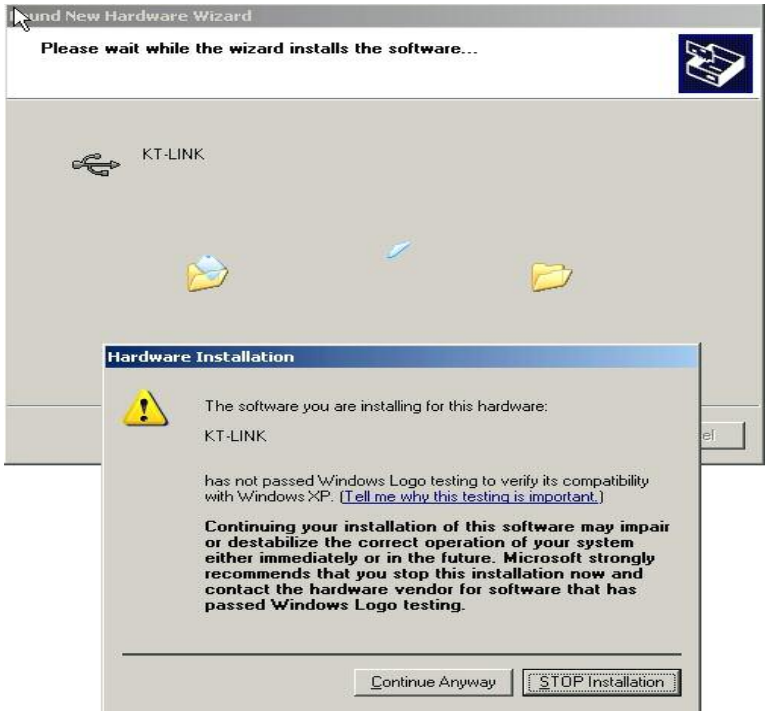
The JTAG dongle “KT-Link” is build on the basis of FTDI devices, so use the FTDI drivers delivered with the dongle or the driver that is already extracted on the folder *C:\OpenOCD_FTDI\FTDI* in the next installation step.



Windows then looks for an adequate USB driver.

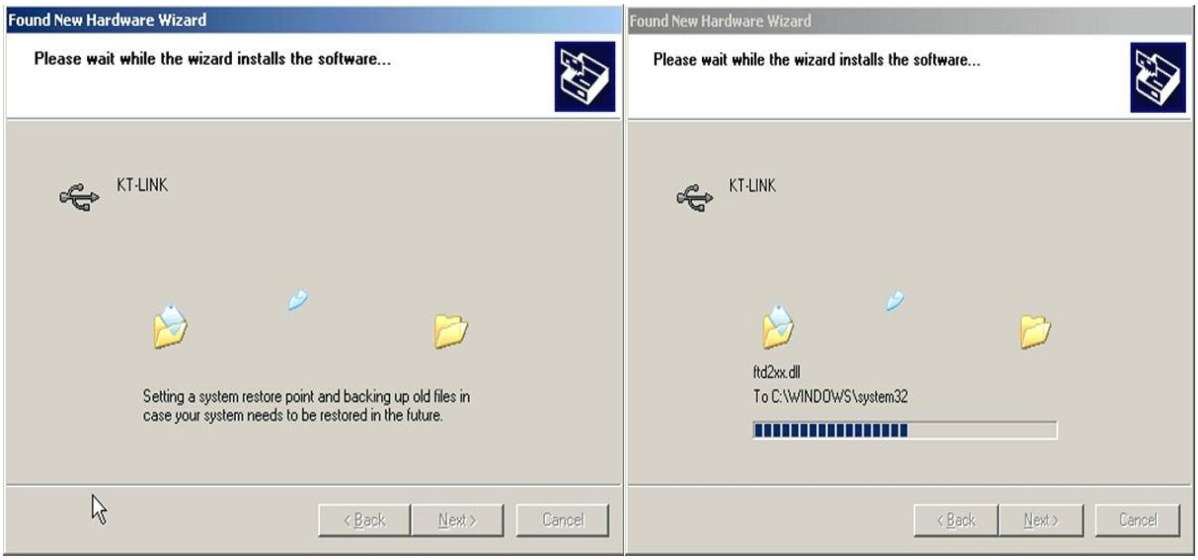


Once a USB driver was found, Windows asks for a confirmation to continue the installation.



After the confirmation the installation process will be resumed. This procedure may look different on other Windows versions.

The next figures demonstrate the procedure by using Windows XP.

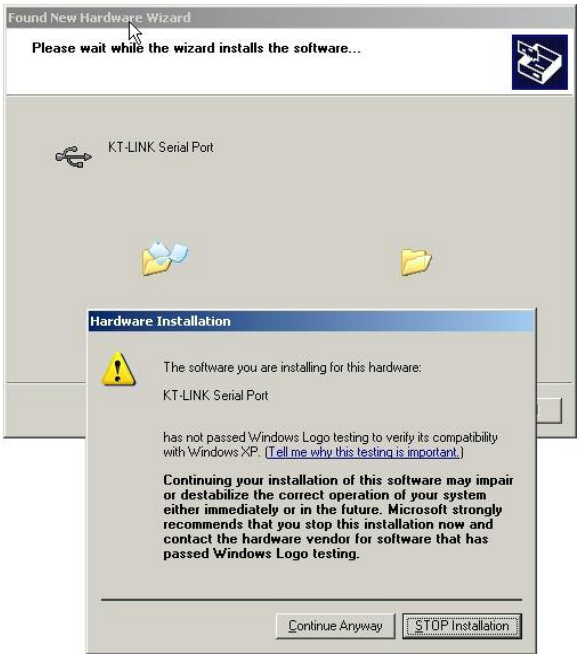


With the confirmation on the next step Windows installs the found driver for the detected “KT-Link” interface.

When the first KT-Link USB driver is successfully installed, Windows switches to the installation of the Virtual COM Port driver. The installation procedure is the same for all interfaces of this dongle.


Found New Hardware Wizard
Welcome to the Found New Hardware Wizard
This wizard helps you install software for:
KT-LINK Serial Port
If your hardware came with an installation CD or floppy disk, insert it now.
What do you want the wizard to do?
Install the software automatically (Recommended)
Install from a list or specific location (Advanced)
Click Next to continue.
< Back
Next >
Cancel


Found New Hardware Wizard
Welcome to the Found New Hardware Wizard
This wizard helps you install software for:
KT-LINK
If your hardware came with an installation CD or floppy disk, insert it now.
What do you want the wizard to do?
Install the software automatically (Recommended)
Install from a list or specific location (Advanced)
Click Next to continue.
< Back
Next >
Cancel

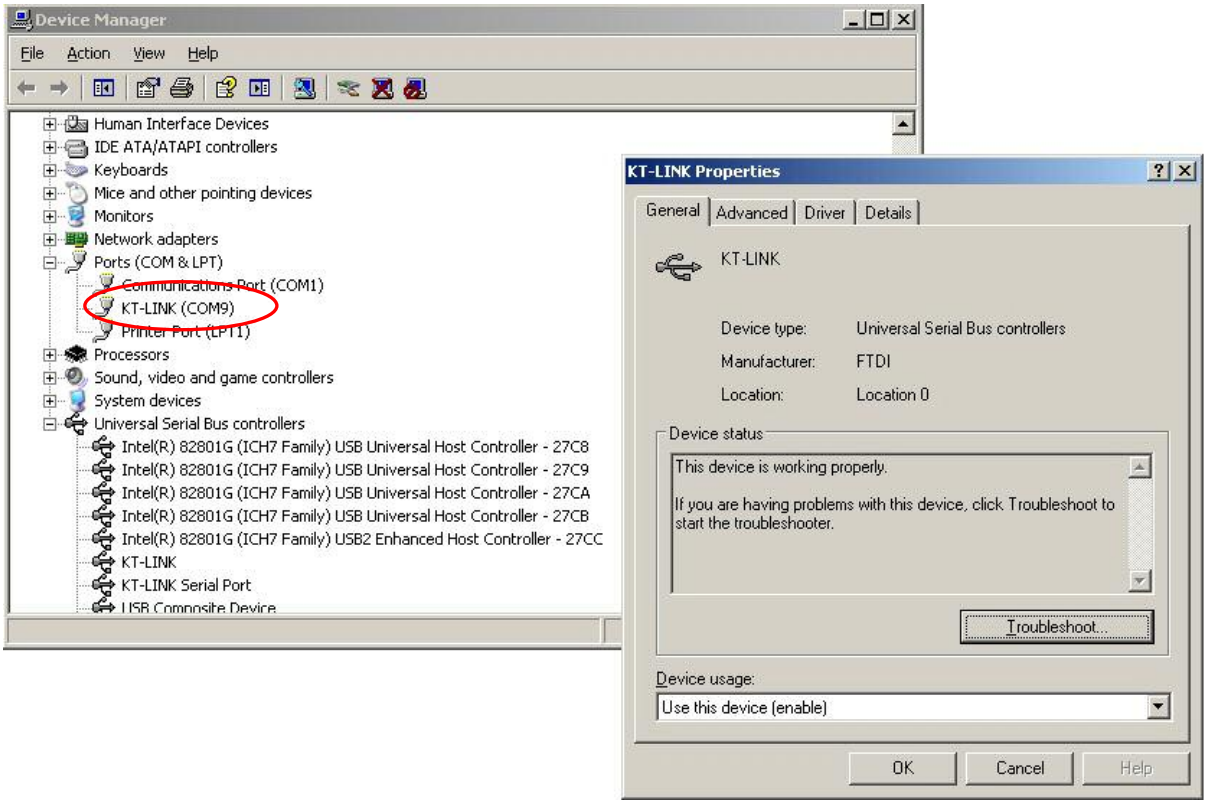

Found New Hardware Wizard
Please wait while the wizard installs the software...
KT-LINK Serial Port
Hardware Installation
The software you are installing for this hardware:
KT-LINK Serial Port
has not passed Windows Logo testing to verify its compatibility with Windows XP. (Tell me why this testing is important.)
Continuing your installation of this software may impair or destabilize the correct operation of your system either immediately or in the future. Microsoft strongly recommends that you stop this installation now and contact the hardware vendor for software that has passed Windows Logo testing.
Continue Anyway
STOP Installation


Found New Hardware Wizard
Please wait while the wizard installs the software...
KT-LINK
Hardware Installation
The software you are installing for this hardware:
KT-LINK
has not passed Windows Logo testing to verify its compatibility with Windows XP. (Tell me why this testing is important.)
Continuing your installation of this software may impair or destabilize the correct operation of your system either immediately or in the future. Microsoft strongly recommends that you stop this installation now and contact the hardware vendor for software that has passed Windows Logo testing.
Continue Anyway
STOP Installation


Found New Hardware Wizard
Completing the Found New Hardware Wizard
The wizard has finished installing the software for:
KT-LINK Serial Port
Click Finish to close the wizard.
< Back
Finish
Cancel


Found New Hardware Wizard
Completing the Found New Hardware Wizard
The wizard has finished installing the software for:
KT-LINK
Click Finish to close the wizard.
< Back
Finish
Cancel

To check the installation of the JTAG dongle “KT-Link” refer to the Window’s Device Manager.



The number of the COM port the driver used during the installation may differ for system to system.

3.3.2 Run OpenOCD

To test the OpenOCD server installed on the computer use the windows command line.

Connect the SK-FM3-100PMC board via JTAG interface to the USB interface of your computer. Use the JTAG dongle “KT-Link”.

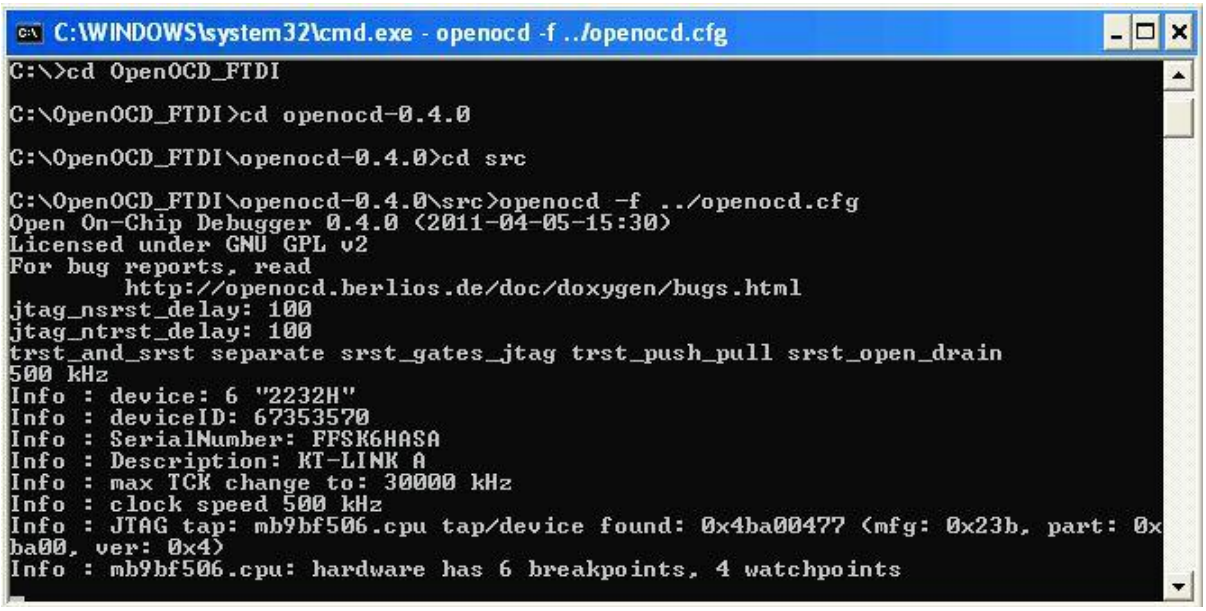
A configuration script file *openocd.cfg* for OpenOCD is also needed. This file is included in the software package of this application note (Eclipse project workspace).

The OpenOCD configuration file *openocd.cfg* for the MB9BF506N example is shown below:

```
# Interface used “KT-Link”
interface ft2232
ft2232_device_desc "KT-LINK"
ft2232_layout ktlink
ft2232_vid_pid 0x0403 0xBBE2

# Fujitsu Cortex-M3 with 512kB Flash and 64 kB RAM

if { [info exists CHIPNAME] } {
      set  _CHIPNAME $CHIPNAME
} else {
      set  _CHIPNAME mb9bf506
}

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

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

# MB9BF506 has 64kB of RAM on its main system bus
$_TARGETNAME configure -work-area-phys 0x1FFF8000 -work-area-size 0x10000
-work-area-backup 0


# MB9BF506 has 512kB 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 -f <Your path to the Eclipse workspace project>/openocd.cfg
```



The screen shot demonstrates that the OpenOCD server was started and a Connection between host and target over JTAG is available.

If the Kt-Link is not recognized (e.g. FTDI device connection fails), download Libusb Device Filter from

http://sourceforge.net/projects/libusb-win32/files/libusb-win32-releases/1.2.4.0/libusb-win32-bin-1.2.4.0.zip/download

and run *install-filter-win.exe* and choose the connected device (e.g. KT-Link).

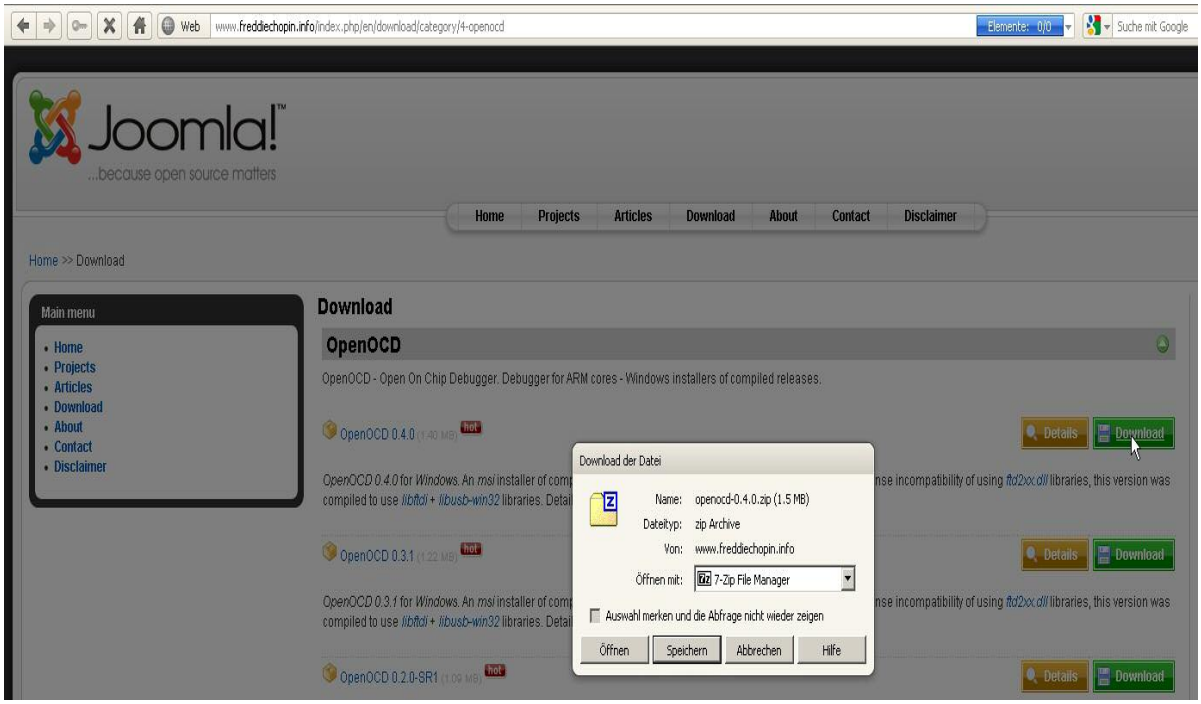
Afterwards try again to check the OpenOCD connection as described above.

3.4 Using Open OCD with LibUSB driver

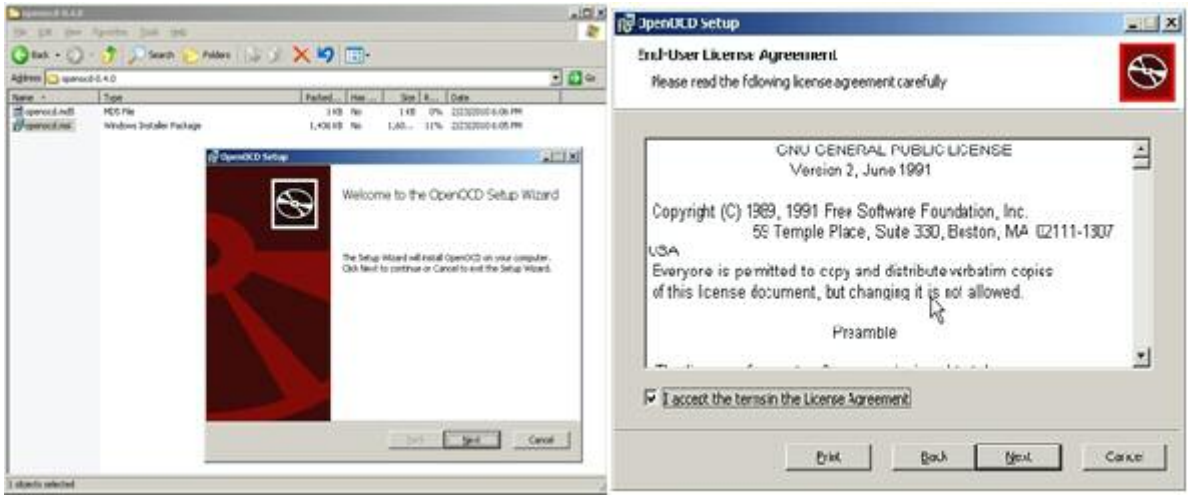
3.4.1 Installation of OpenOCD version supporting LibUSB driver

The Windows installer program for the version of OpenOCD that support LibUSB driver can be downloaded from the website:

http://www.freddiechopin.info/index.php/en/download/category/4-openocd

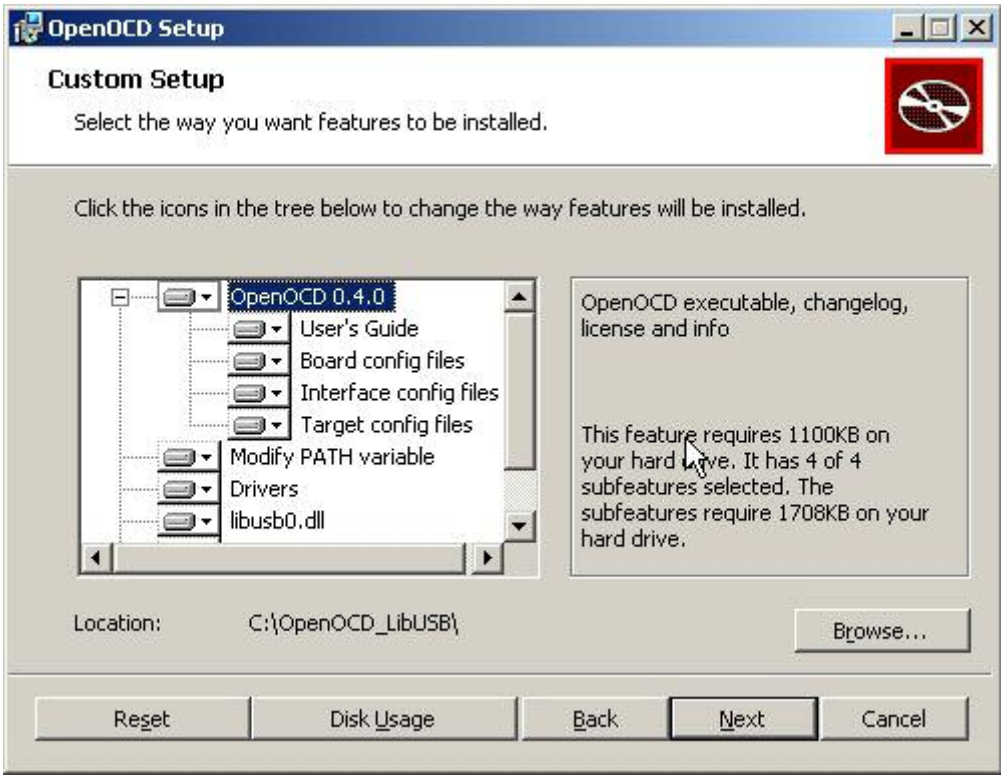


The zip file should be saved and then extracted. After this it is only needed to start the installer program and to follow the instructions.

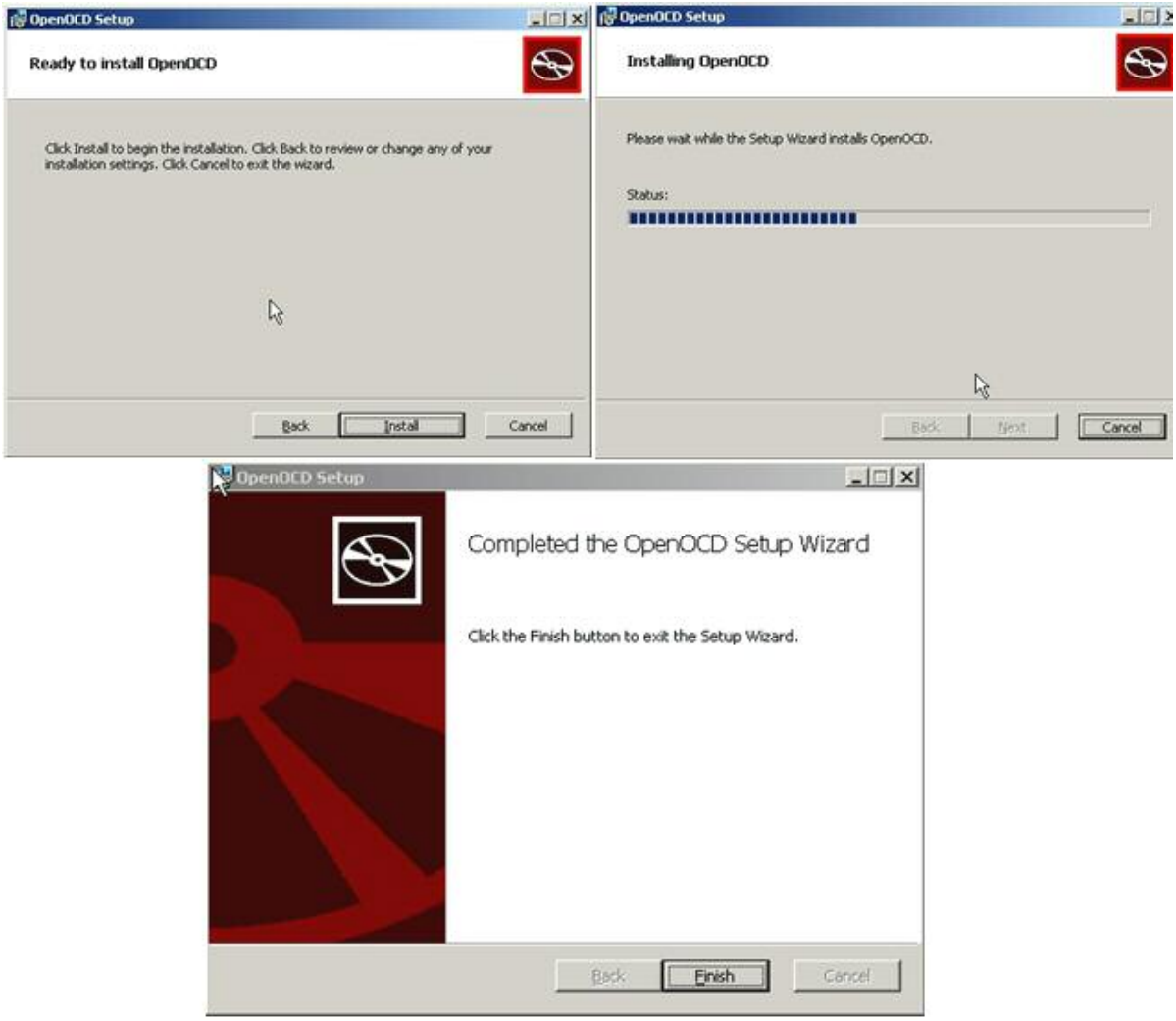


After the confirmation of the GPL licence, chose the features of OpenOCD, which will be installed. During the installation OpenOCD executable file *openocd.exe* will be added to the Windows path.

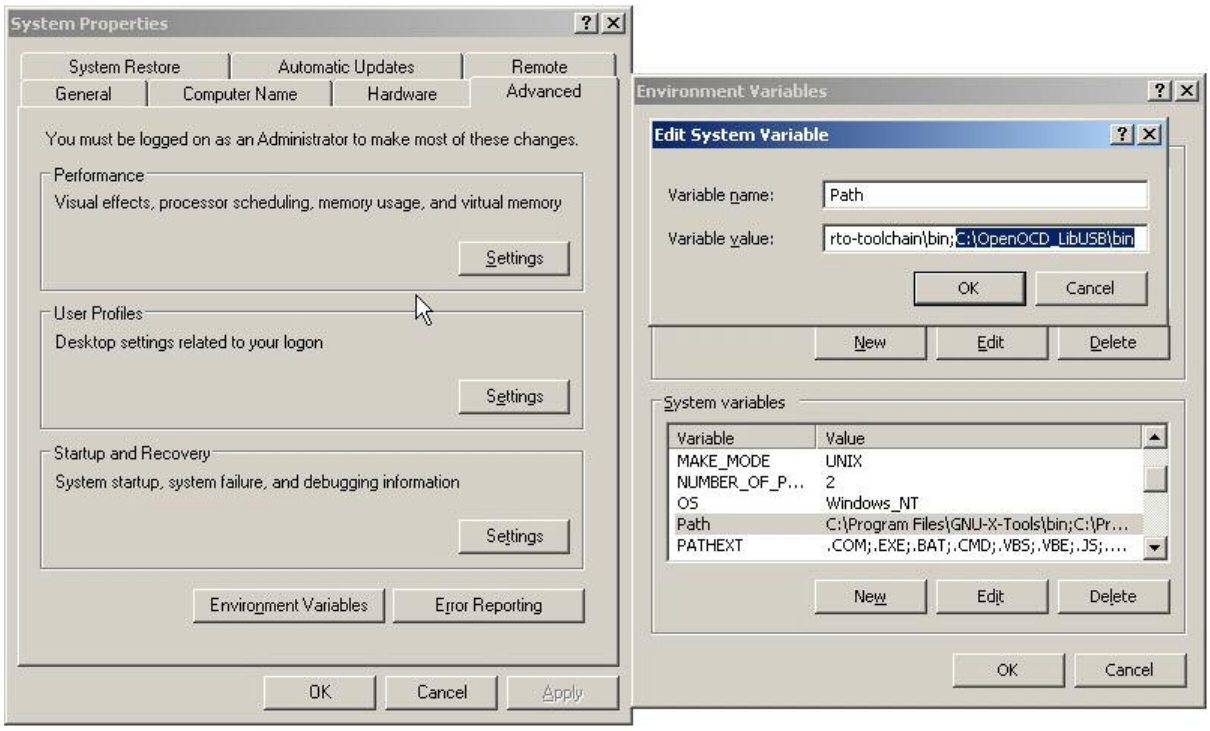
For the next steps it is needed to recall the location of the folder, where OpenOCD was installed, e.g. *C:\OpenOCD_LibUSB*.



The following steps of the OpenOCD installation procedure have to be done.



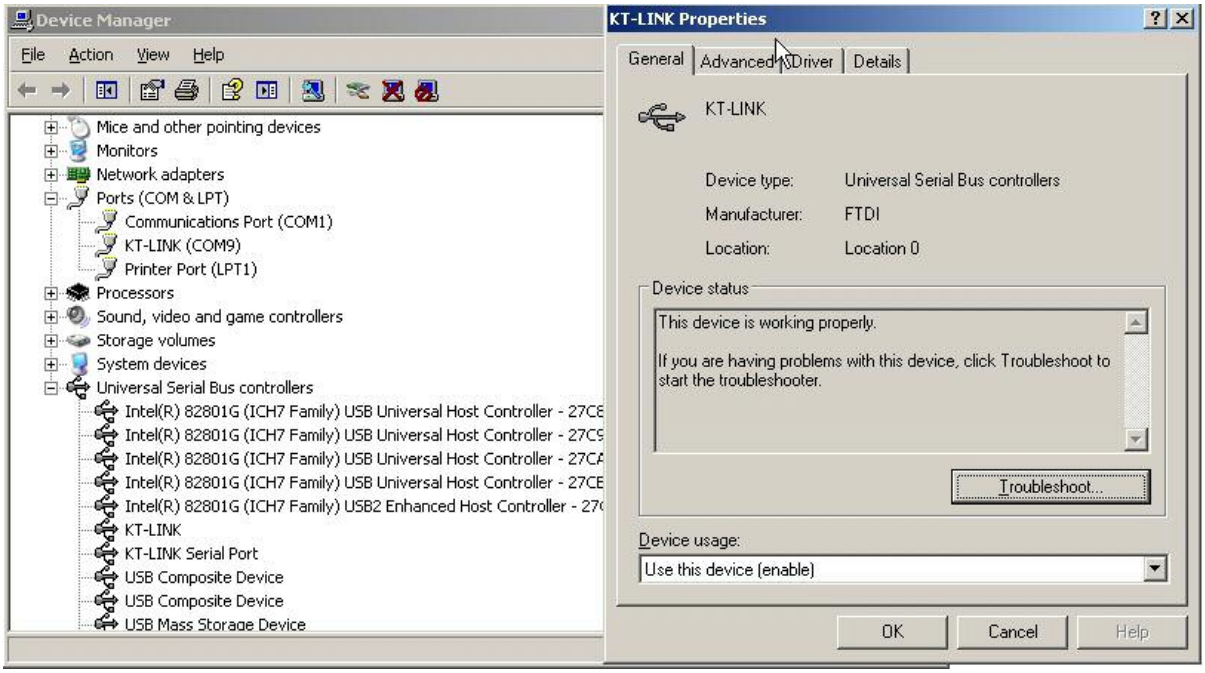
The Windows System Properties shows, if OpenOCD was successfully installed and added to the Windows path.



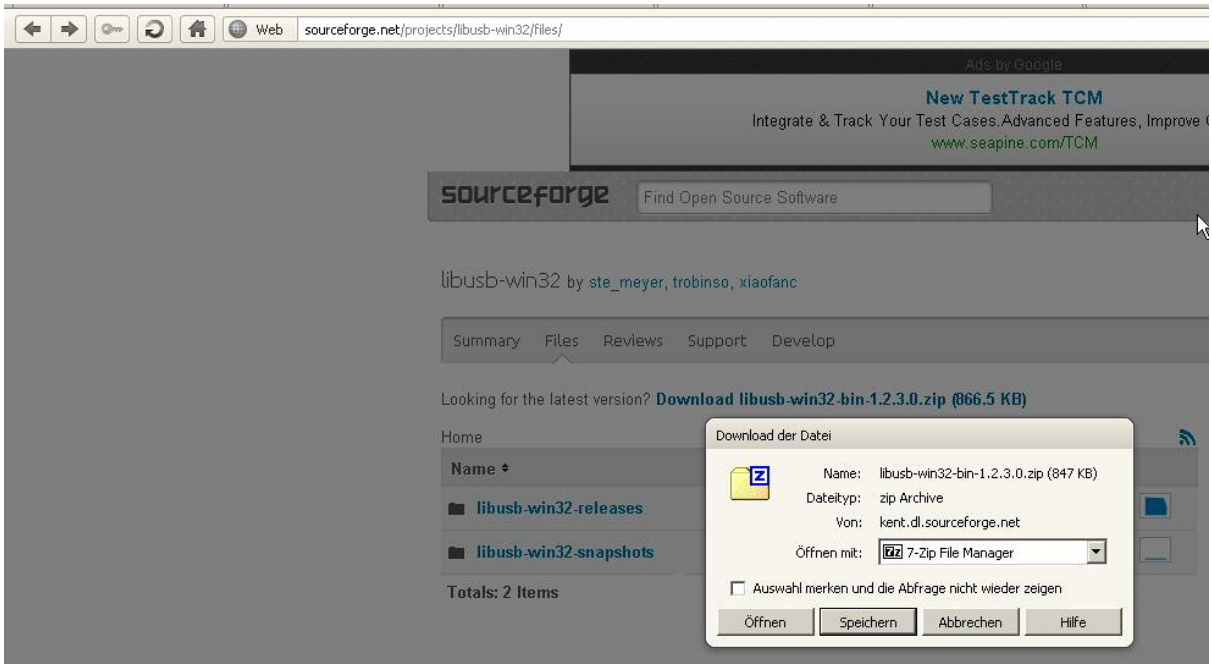
3.4.2 Installation of LibUSB driver for the JTAG dongle

The JTAG dongle “KT-Link” can be driven with the open source LibUSB driver included on the OpenOCD version that was installed (see previous chapters).

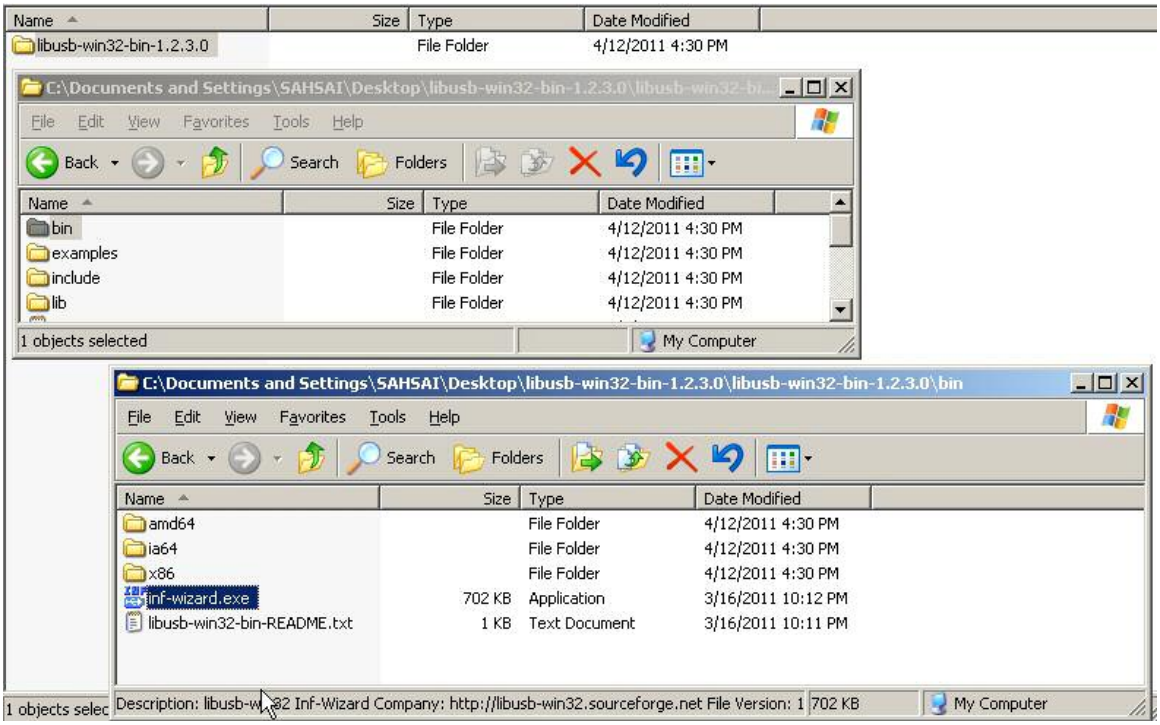
To use this installed version with KT-Link, first change the KT-Link driver from FTDI to LibUSB driver.



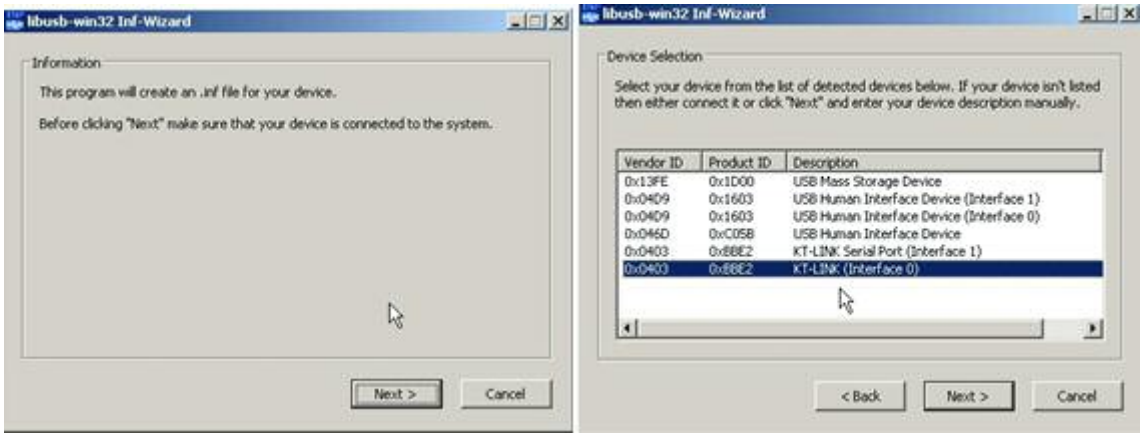
The LibUSB driver for Windows (2000, XP, Vista and 7; 98 SE and ME for versions up to 0.1.12.2) can be generated with the binary package for Windows “libusb-win32”. This package can be downloaded from: http://sourceforge.net/projects/libusb-win32/files/



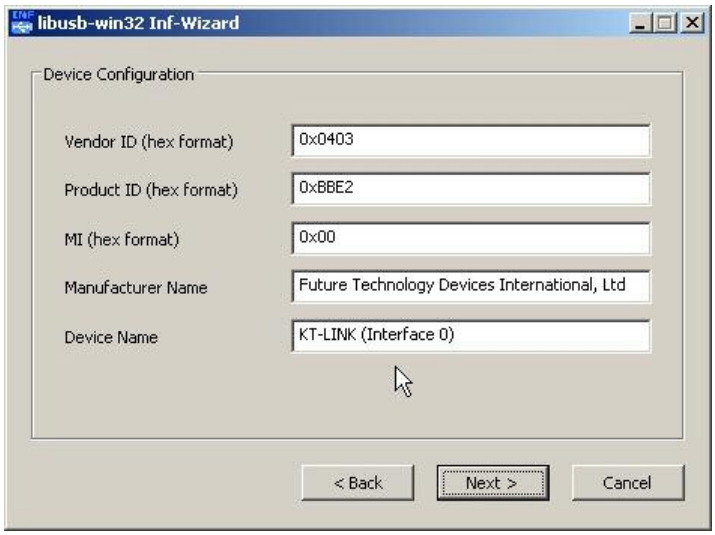
After saving and decompressing the zip file, continue with the installation of the LibUSB driver for “KT-Link”.



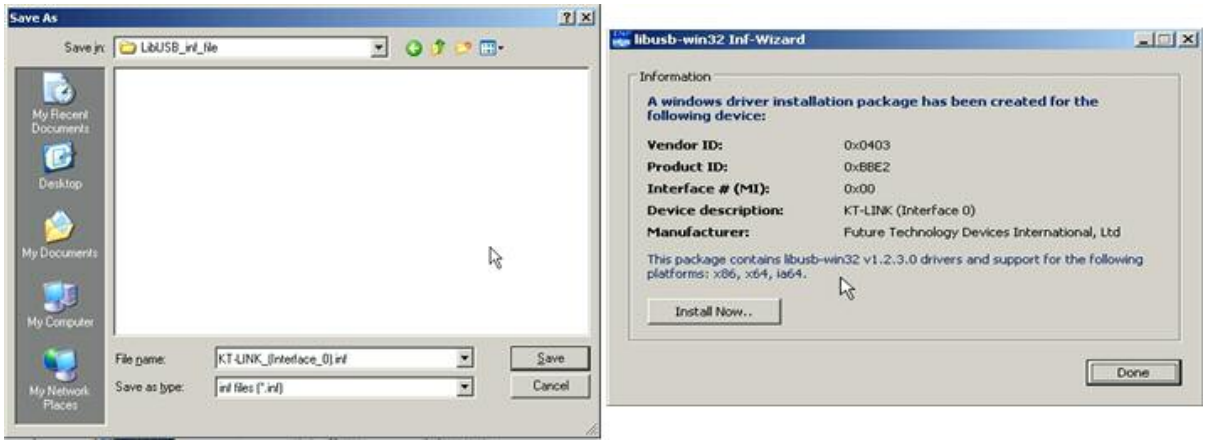
With the wizard program *inf-wizard.exe* the LibUSB driver for the “KT-Link” dongle will be generated and installed. Run this installation wizard.



The wizard program will find two interface devices from “KT-Link”. Both devices must be installed. The installation is the same for all devices. Start the installation with interface 0.

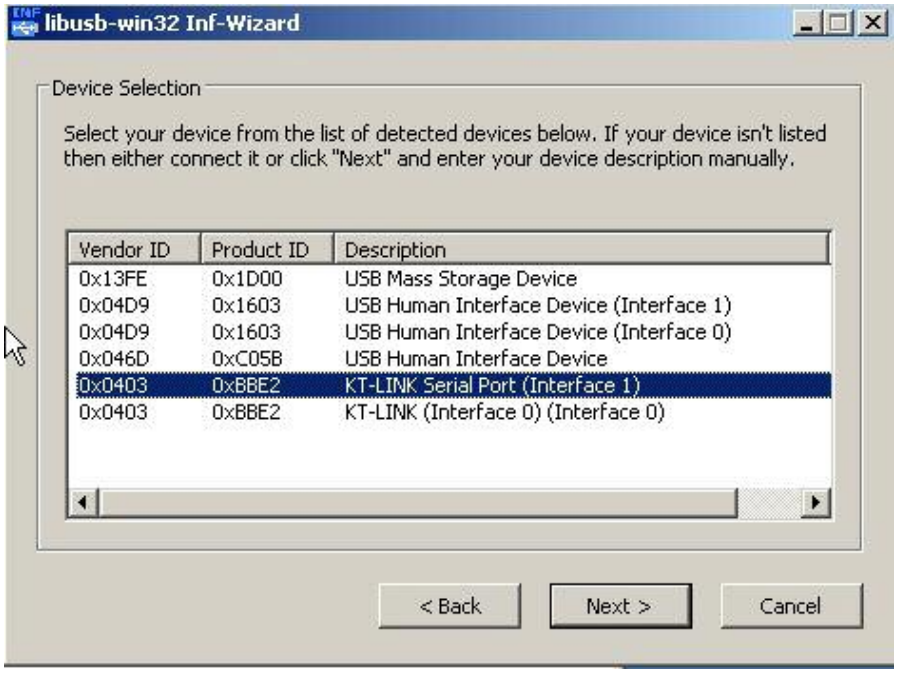


In the next step save the generated *i*nf file for the first “KT-Link” interface in a folder of your choice. After this the wizard asks for immediate installation of the LibUSB driver.

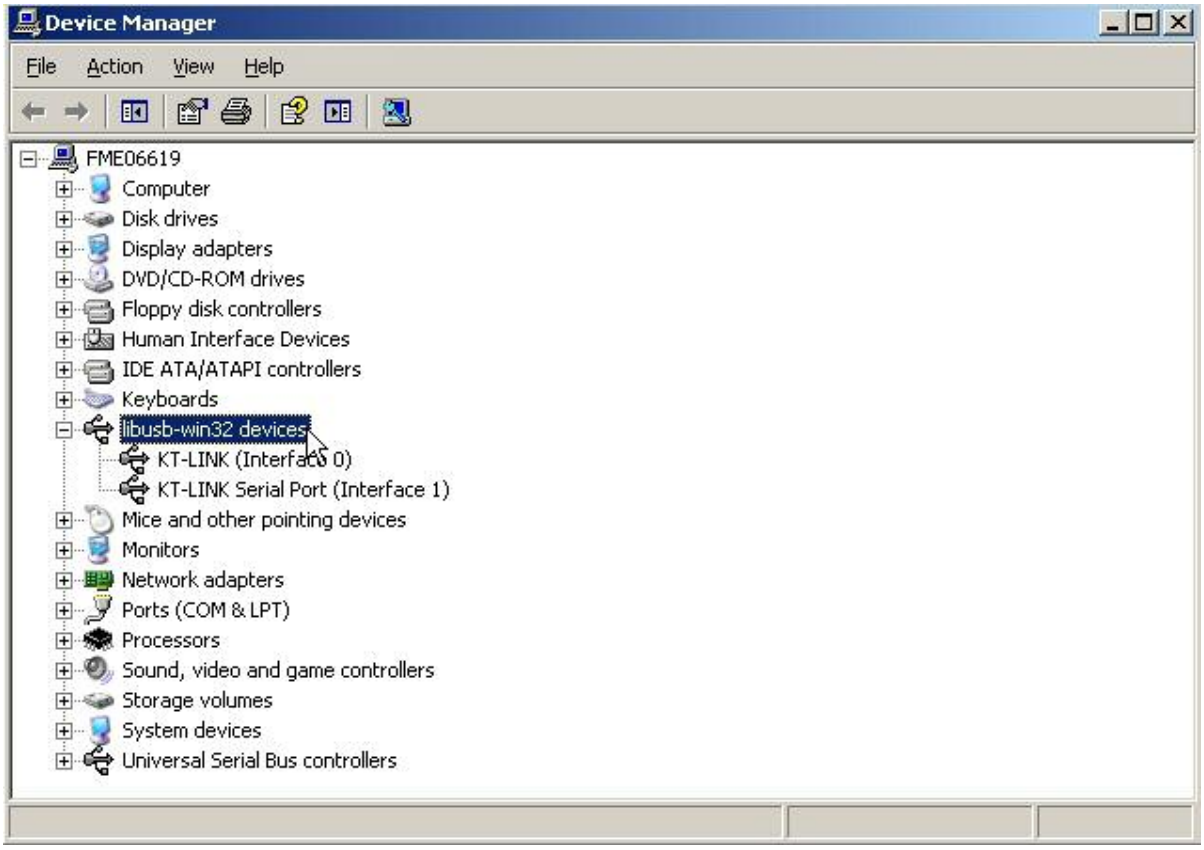


The installation procedure of the driver will be contuned and the LibUSB driver will be installed.

The same installation procedure must be done for all interfaces of the JTAG dongle. So this procedure is also needed for the second interface “Serial Port” previously found by the wizard.



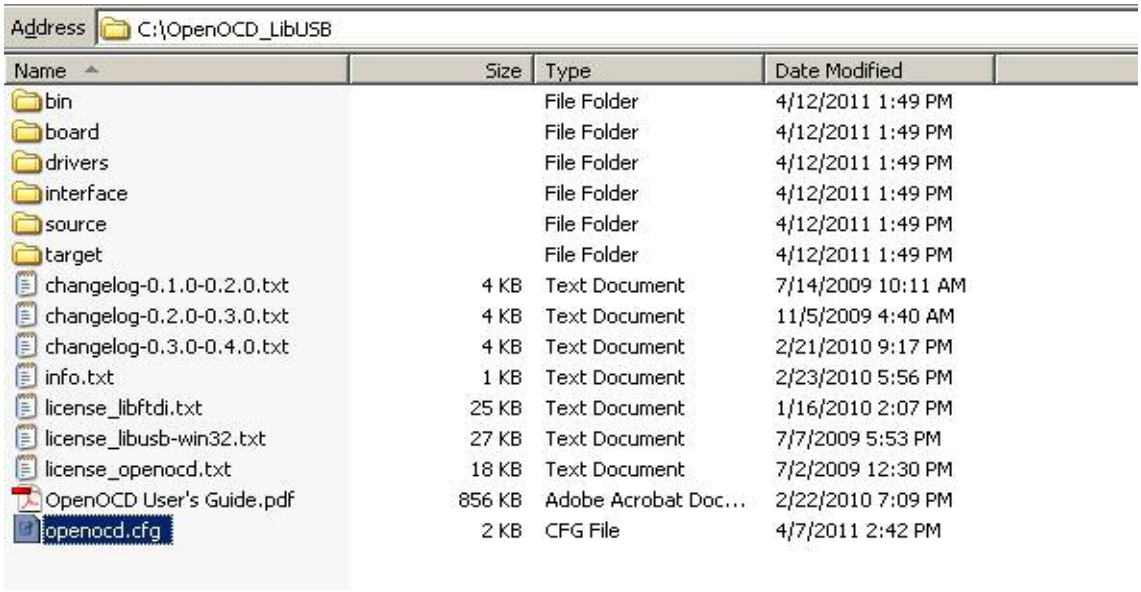
When the update of the driver for all “KT-Link” interfaces was done, we can check the result of this procedure with the Windows Device manager.



3.4.3 Test of OpenOCD Server configured on the Basis of LibUSB Driver

To run the installed version of OpenOCD server which integrates the LibUSB driver, connect the SK-FM3-100PMC board via the JTAG dongle “KT-Link” to the USB interface of the host PC like shown on the photo in chapter 3.3.2.

The OpenOCD configuration file *openocd.cfg* is also needed. OpenOCD will look for this file in its directory ‘*C:\OpwenOCD_LibUSB*. Add this file in your OpenOCD directory. See chapter 3.3.2 for details.



During the installation of OpenOCD the wizard adds the executable file *openocd.exe* from the binary folder *bin* to the windows path.

To check OpenOCD server, start the windows prompt and run the OpenOCD executable.

The OpenOCD command parameter `-f` assigns the configuration file *openocd.cfg* as argument to the executable file *openocd.exe*. You can use absolute or relative paths. The executable file is located in the *bin* folder.

The command is (assuming that the configuration file is located one folder above *openocd.exe*):

```
>openocd -f ../openocd.cfg
```

The screen shot shows that OpenOCD server has started and a connection between host and target over JTAG is available.

Note, that the Eclipse workspace projects have individual *openocd.cfg* files in their own folders!

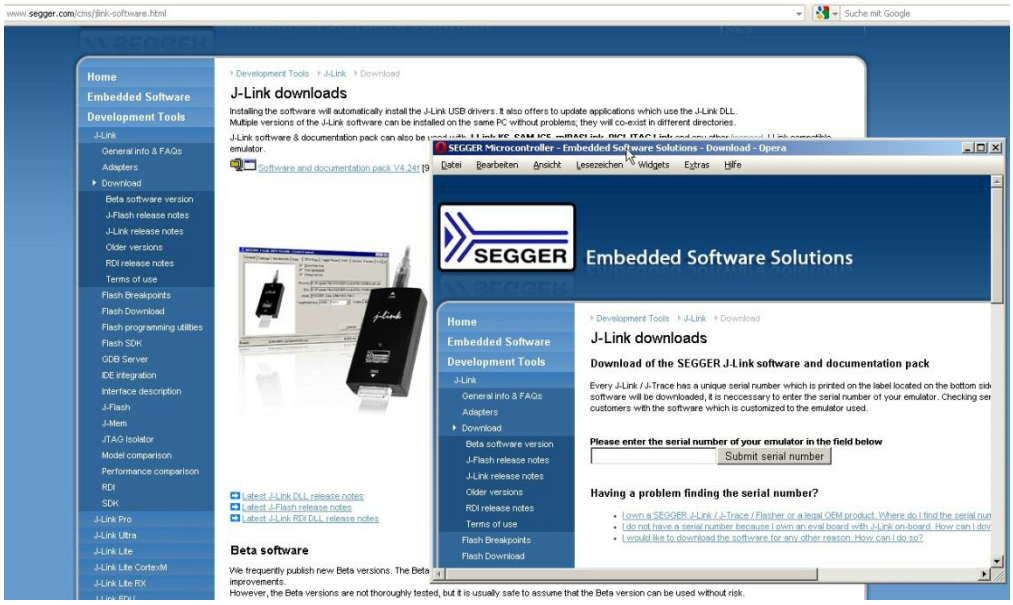
4 J-Link GDB Server

HOW TO USE THE J-LINK GDB SERVER

The J-Link GDB Server is a part of the software packet, which the user of the Segger JTAG interface “J-Link” can use. So we need first to install the J-Link software.

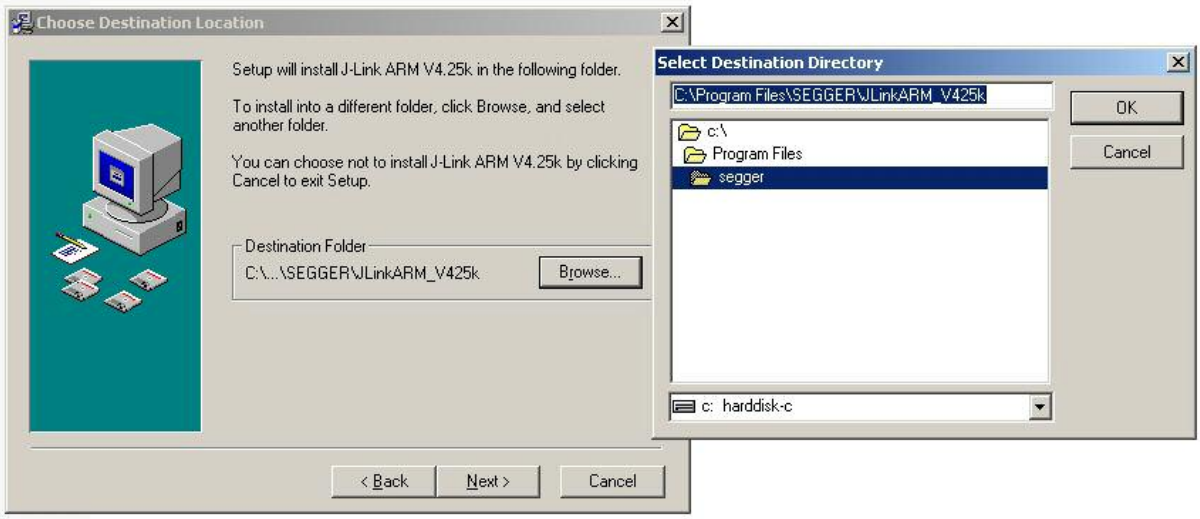
4.1 The J-Link software

The latest version of this software packet is available for downloading from the website: http://www.segger.com/cms/jlink-software.html

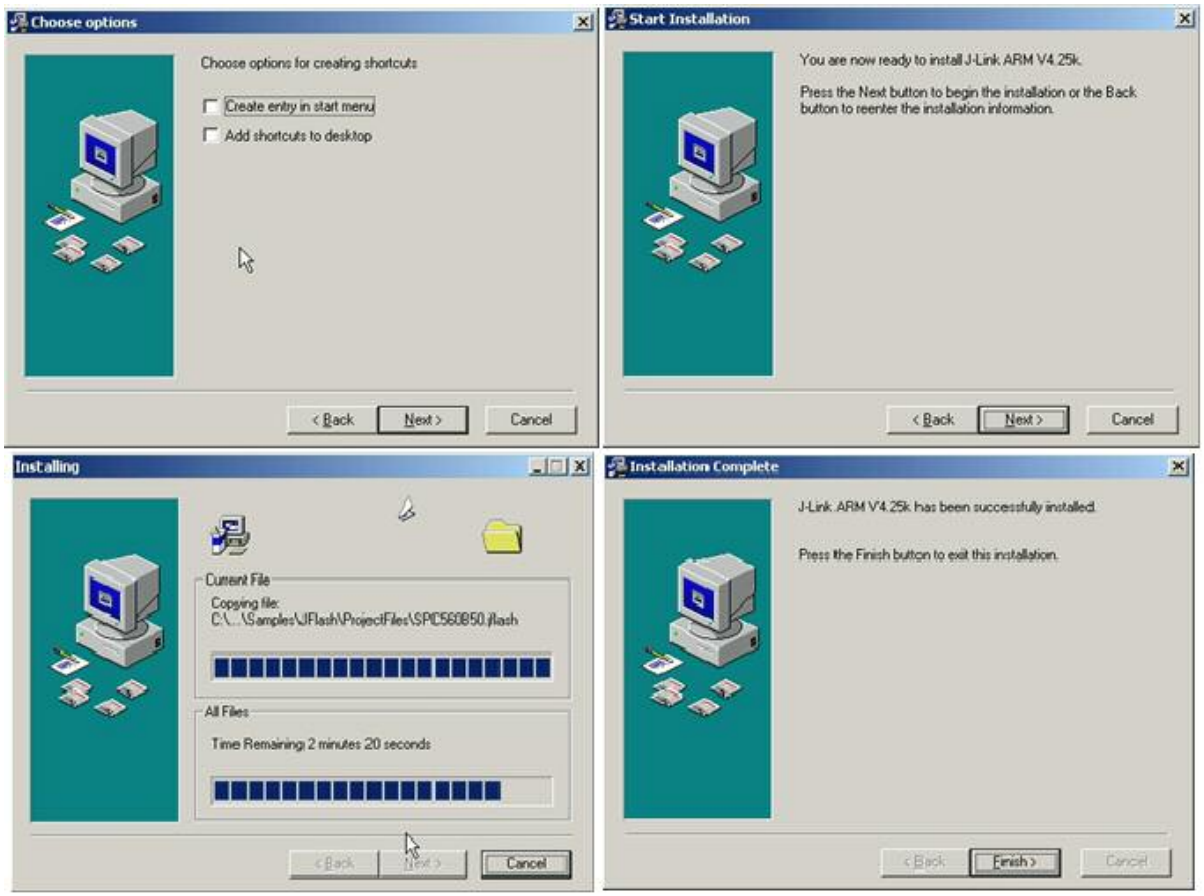


After entering the serial number of the purchased Segger “J-Link”, the software and documentation pack in zip form can be downloaded. Only unzipping this file and starting the installation wizard is needed.

Now choose the installation destination directory of the J-Link software and documentation. Please remember this destination, because it is needed often in this application note.



With the confirmation of the next steps, the installation will be finished.

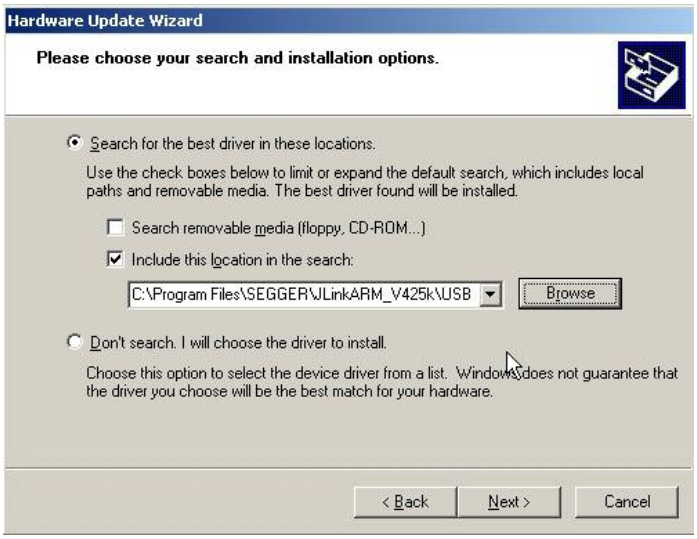


4.2 The Segger JTAG interface “J-Link”

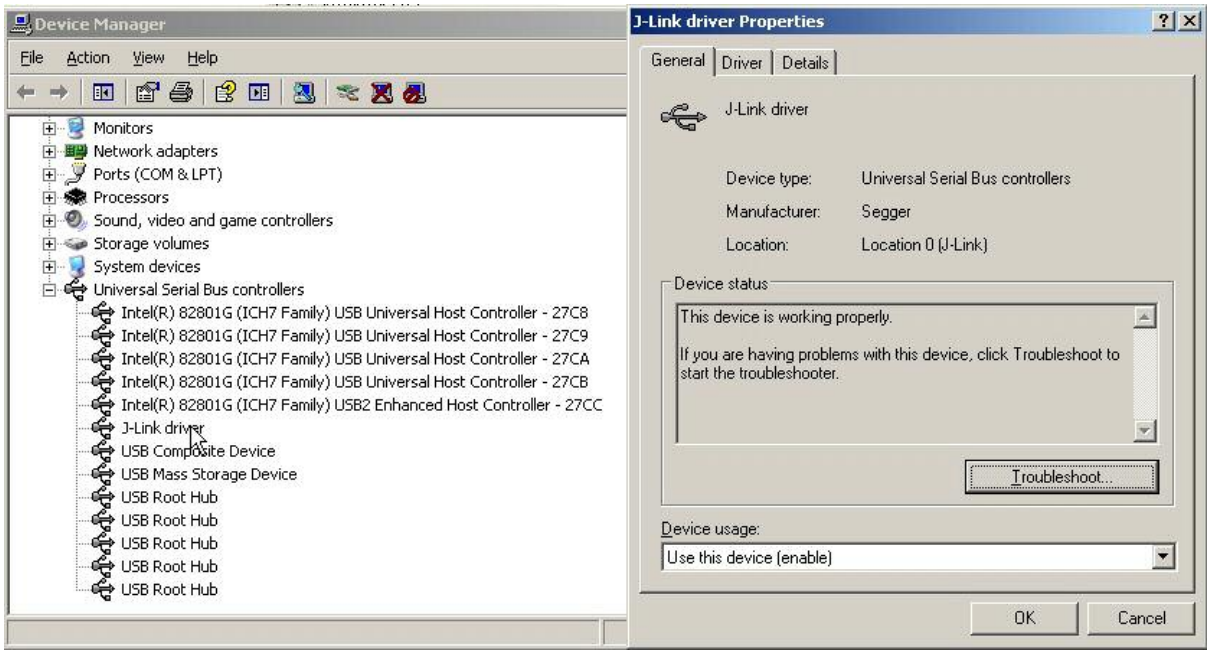
After connecting the JTAG interface “J-Link” via USB to the computer, Windows reports that a new hardware was detected. Windows will automatically find the driver and install it.

If Windows does not automatically locate the driver, the driver can be found in the installation folder of the “J-Link” software, e. g. in the directory:

C:\Program Files\SEGGER\JLinkARM_V425k\USBDriver\x86

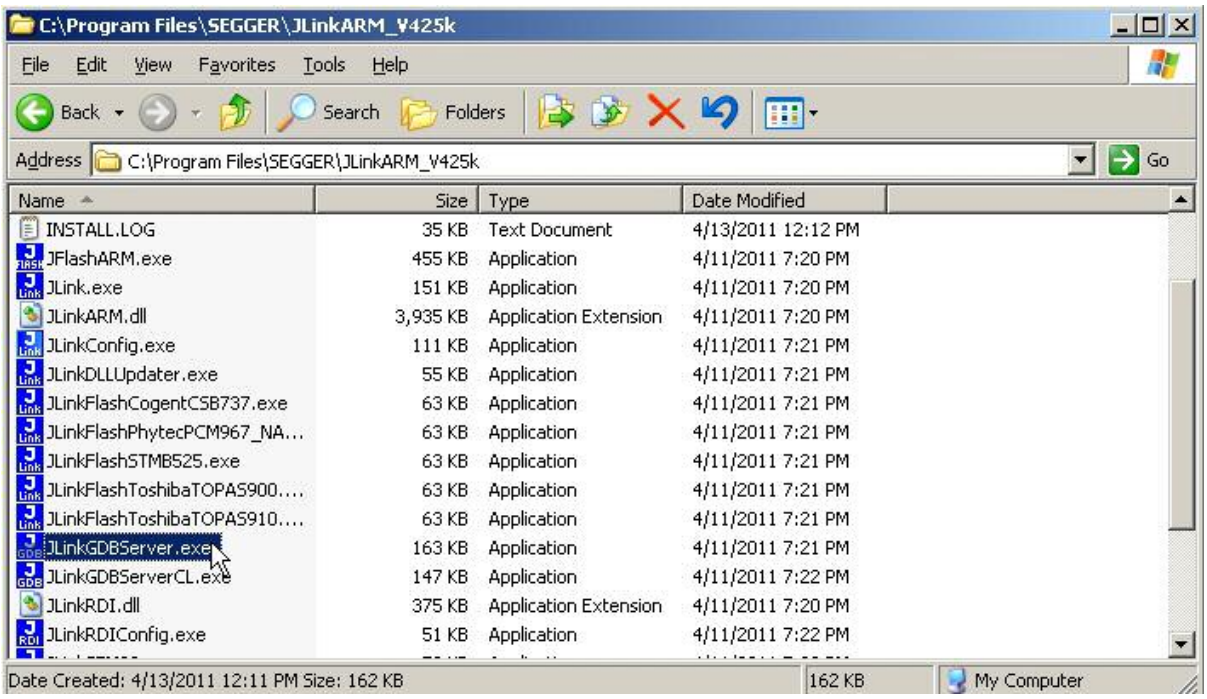


When the installation has finished, check the driver via the device manager.

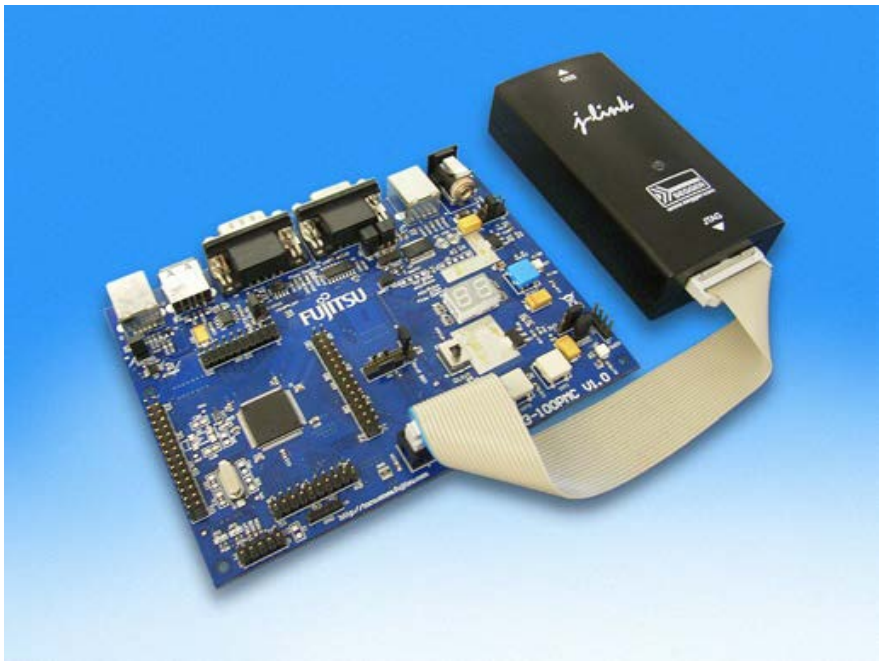


4.3 Test J-Link GDB Server

To test the “J-Link” GDB server installed to the computer, look for the executable file *JlinkGDBServer.exe* located in the “J-Link” software directory.

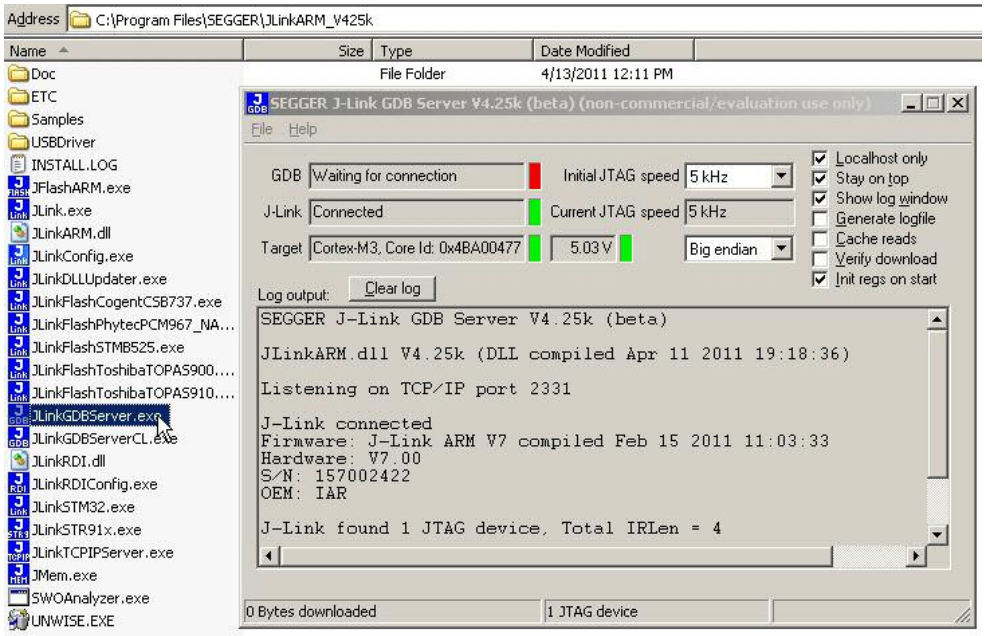


Connect the SK-FM3-100PMC board via the JTAG interface “J-Link” to the USB interface of the host PC.



When the physical connection between the computer and the board via the JTAG interface “J-Link” is established, run the program *JLinkGDBServer.exe*. The GDB server requires a licence before starting.

Enter the license purchased from Segger or start evaluation mode, for non-commercial usage only!



The figure shows that J-Link GDB Server started and a connection between host and target over JTAG is available.

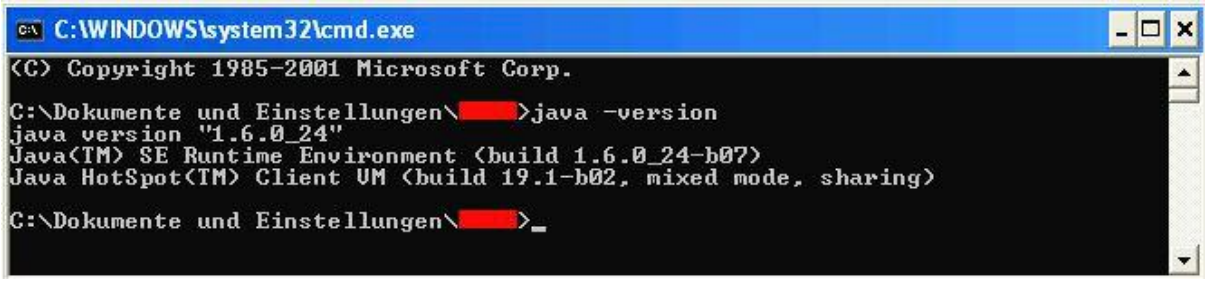
5 Java Runtime Environment JRE

HOW TO INSTALL THE JAVA RUNTIME ENVIRONMENT

5.1 Checking for Java JRE

The installation of Eclipse requires the availability of Java as a virtual machine on system.

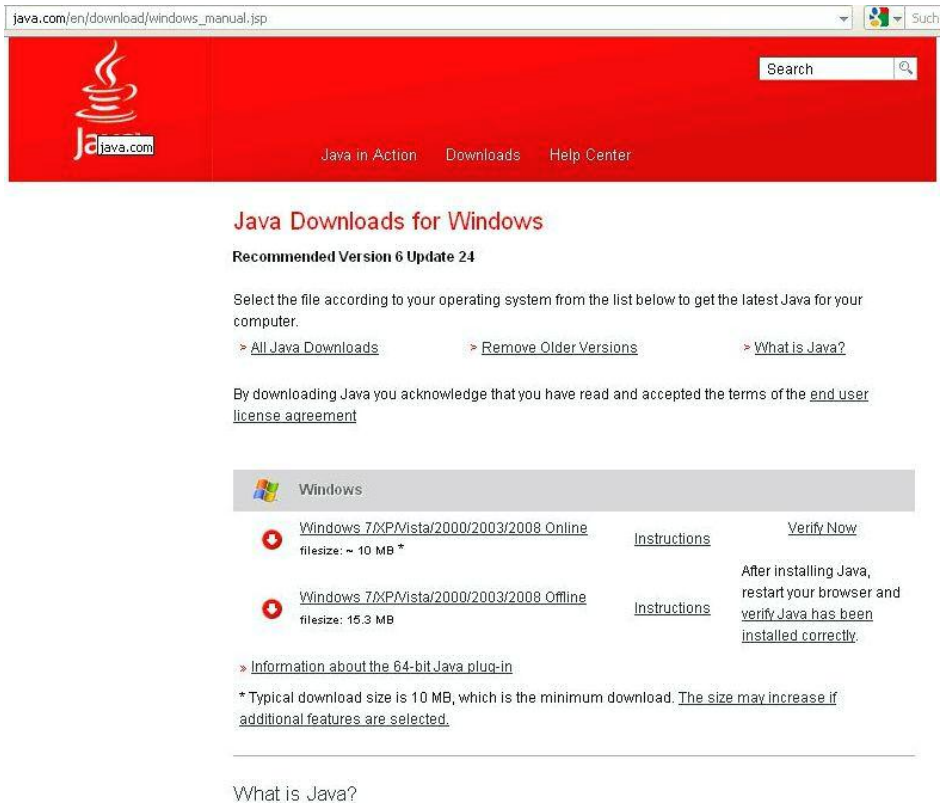
To check, that Java already exists on the system, type the command **`Java -version`** on DOS console.



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

5.2 Installing Java JRE

Download JRE from: http://java.com/



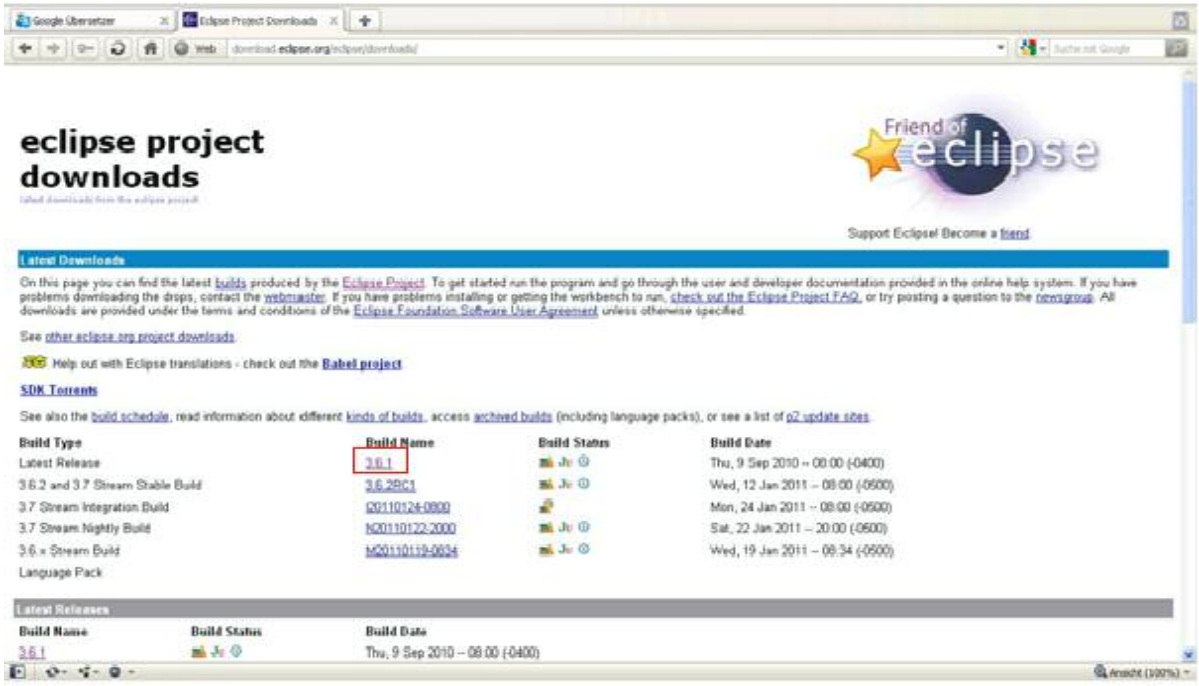
Java installation can be done online or offline. Download one of the installation programs and start the installation procedure to install JRE.

6 Eclipse platform

HOW TO INSTALL THE ECLIPSE IDE

6.1 Eclipse platform

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



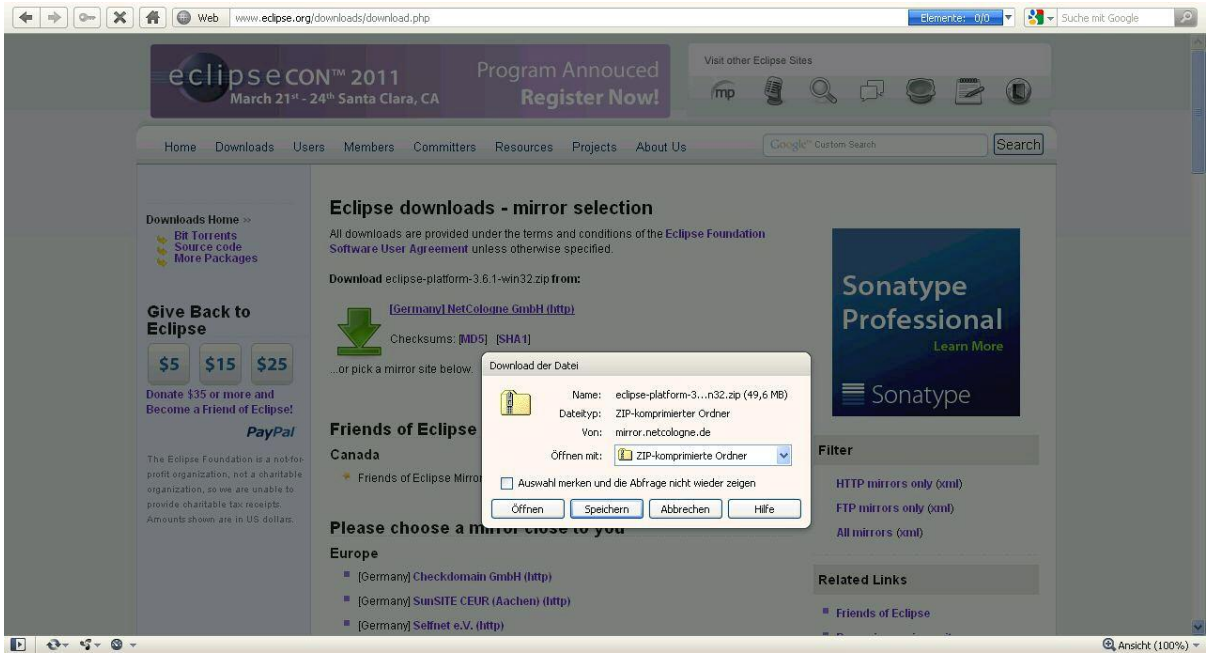
The Helios release 3.6.1 (or later) consists of various packages. These packages are available on the left menu of the download website of the Eclipse project.

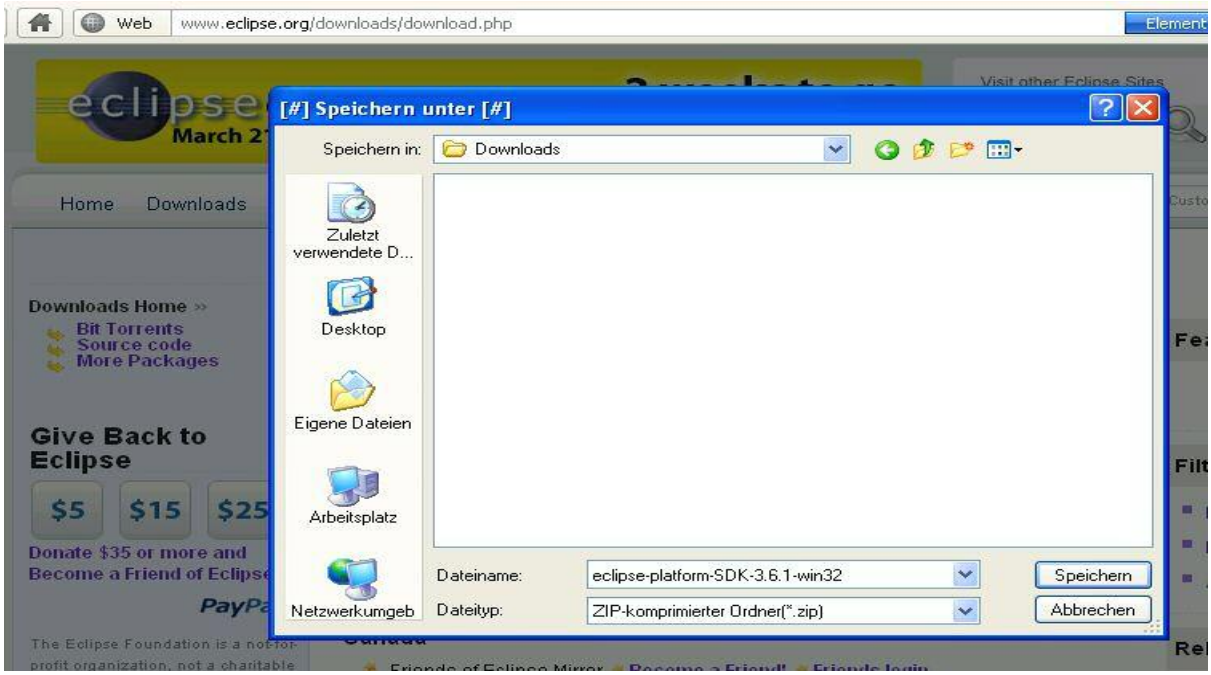
For our system it is required a minimum eclipse platform to be realized. The package needed for this can be found by the menu section “**Platform** Runtime Binary”.

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

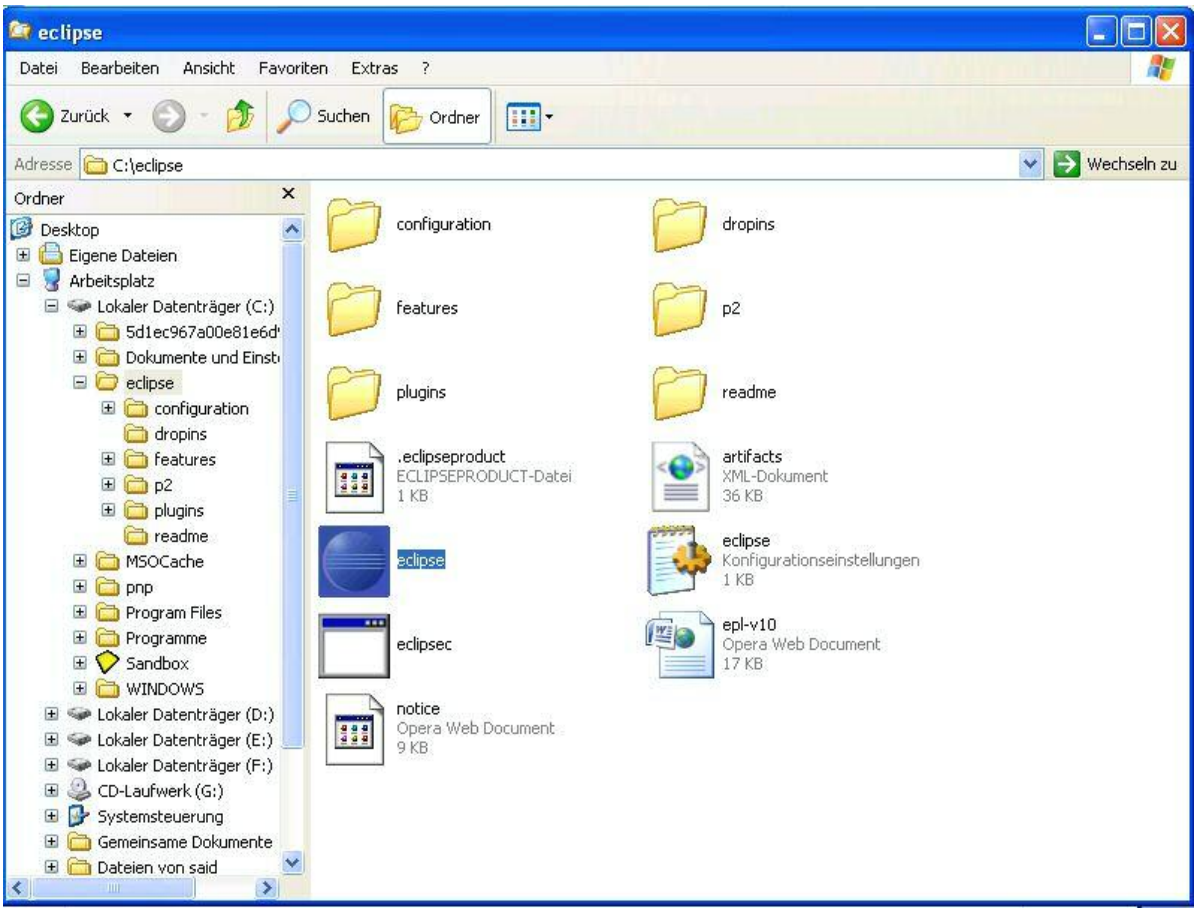
For Windows systems with 32 bit CPU use the first “http” location of this list to download the adequate Eclipse package for this system.

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*-X.Y.Z-*win32.zip*, decompress this file, to e.g. *C:*\.

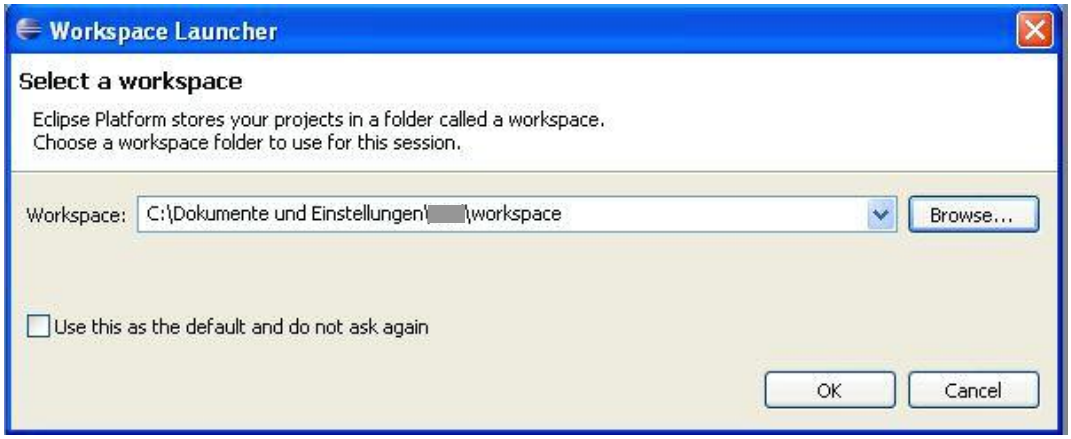


With the installation of Eclipse platform runtime binary, this installation of Eclipse is finished.

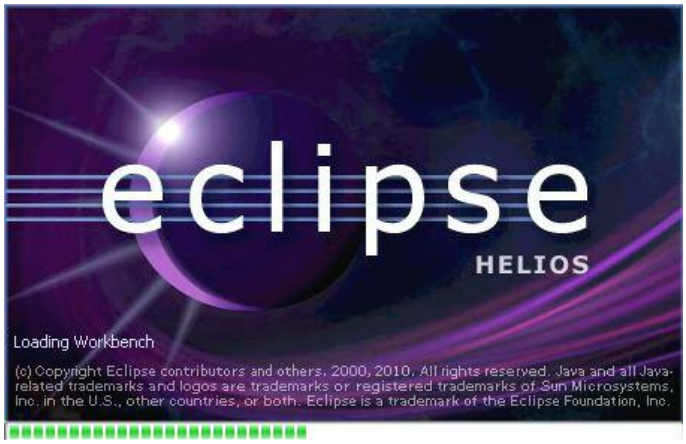
6.2 Start Eclipse IDE

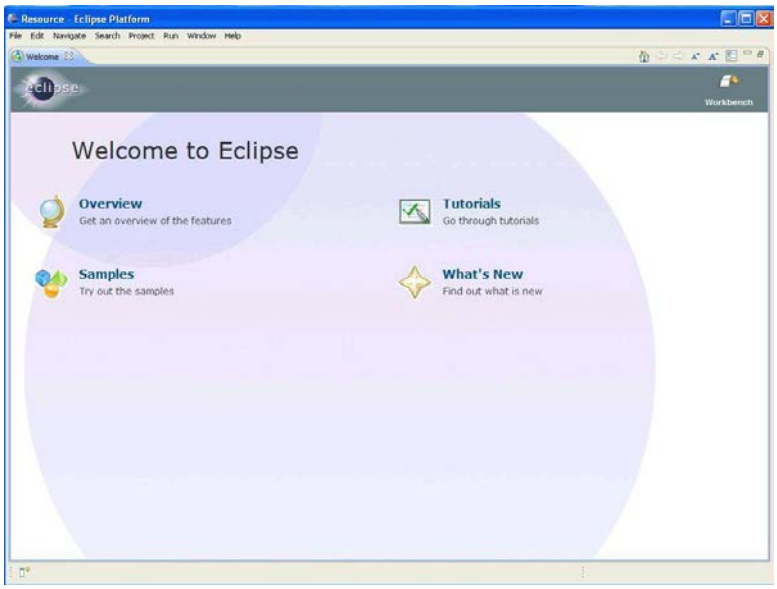
The Eclipse IDE is now ready to start; for this start *eclipse.exe* from the folder *C:\eclipse*.

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



After the selection of the workspace, Eclipse starts.





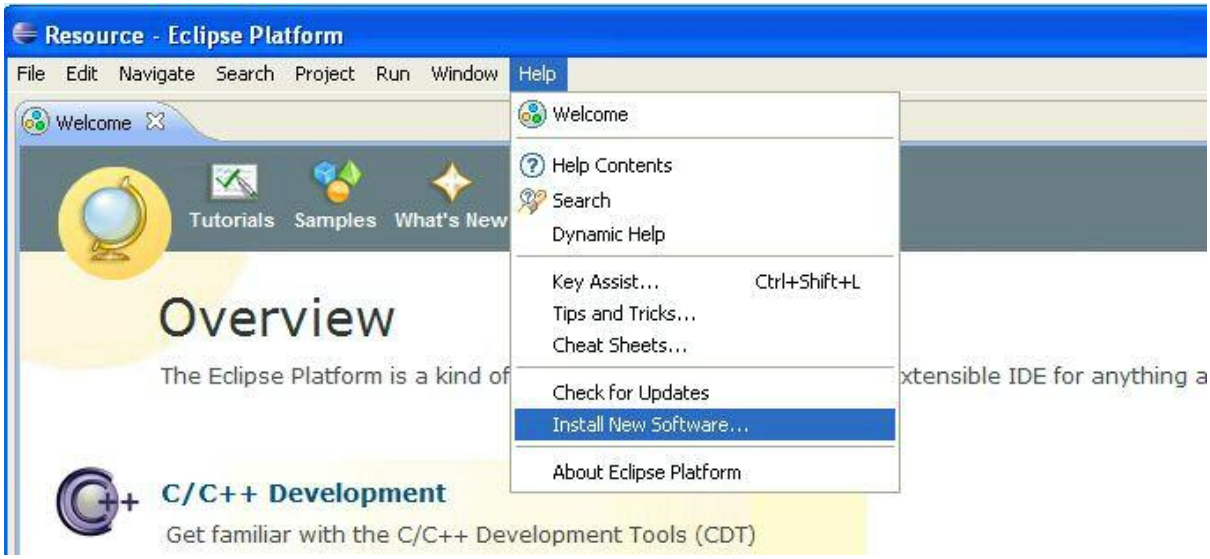
7 C/C++ Development Tooling CDT

HOW TO INSTALL THE C/C++ DEVELOPMENT TOOLING

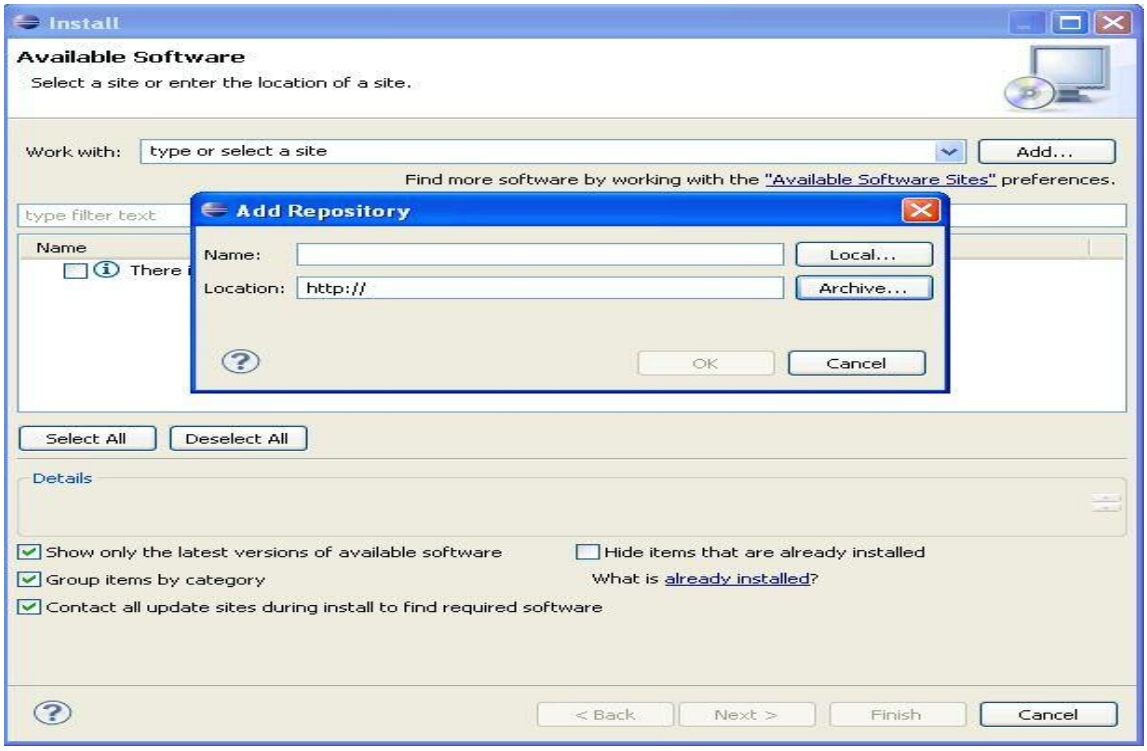
7.1 Installation of new Software on Eclipse

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.

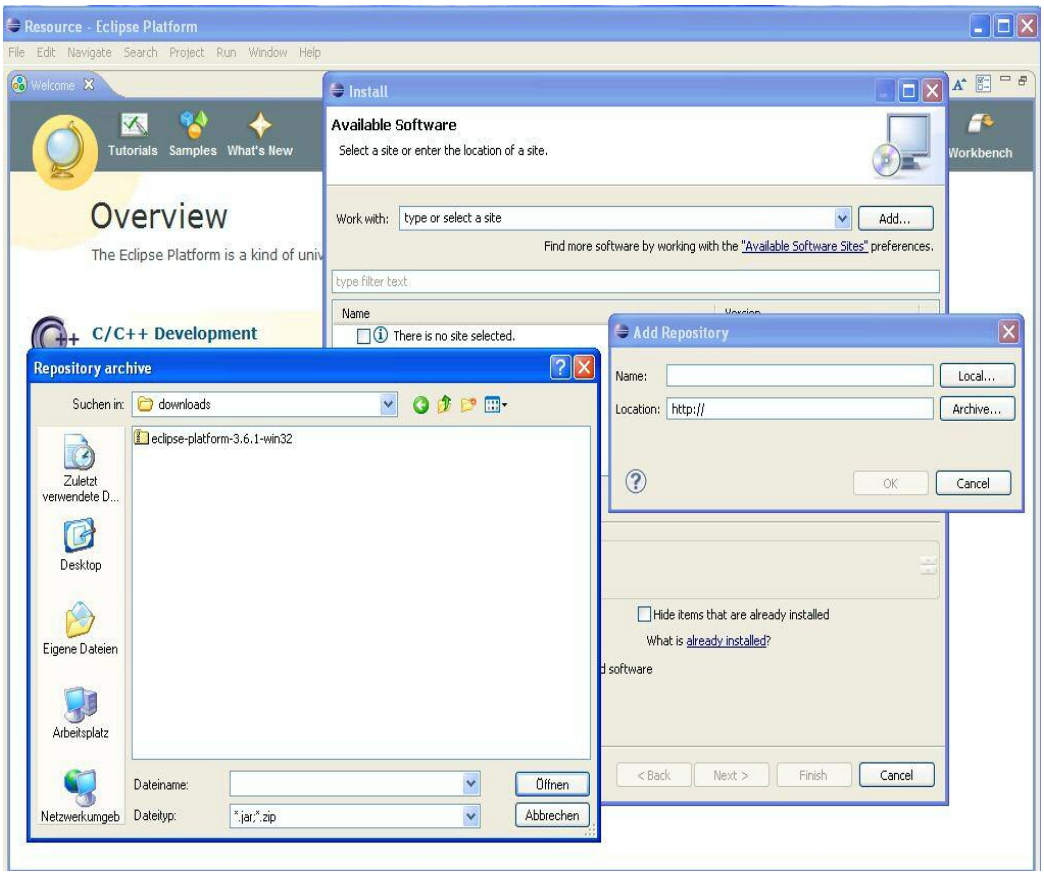


The installation of CDT plug-in or any another package to the Eclipse platform depends on the procedure, which the user selects to add this software to the platform. After clicking of the *add* button the *Add Repository* window appears.

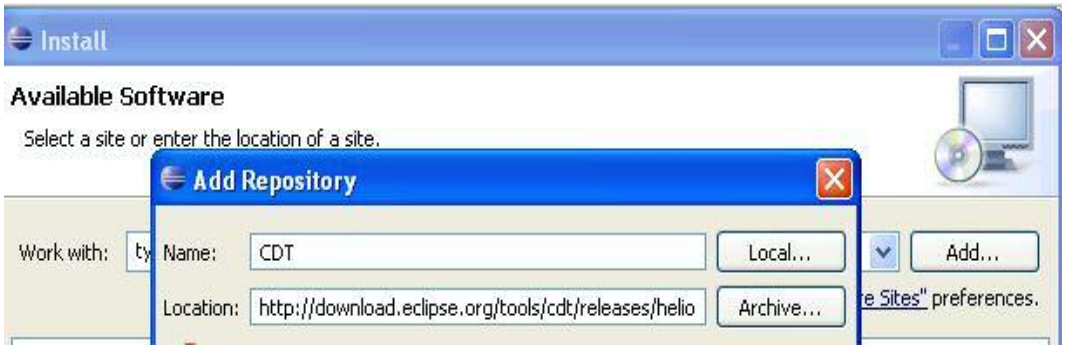


Eclipse supports two different methods to implement new plug-ins to the platform:

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



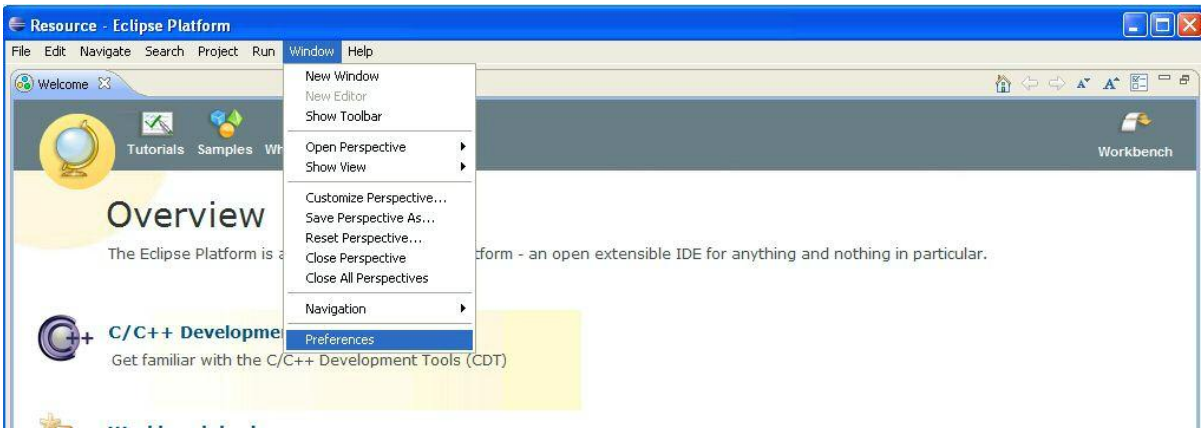
When the plug-in is available from a http project website, a new installation or update of this software is done online



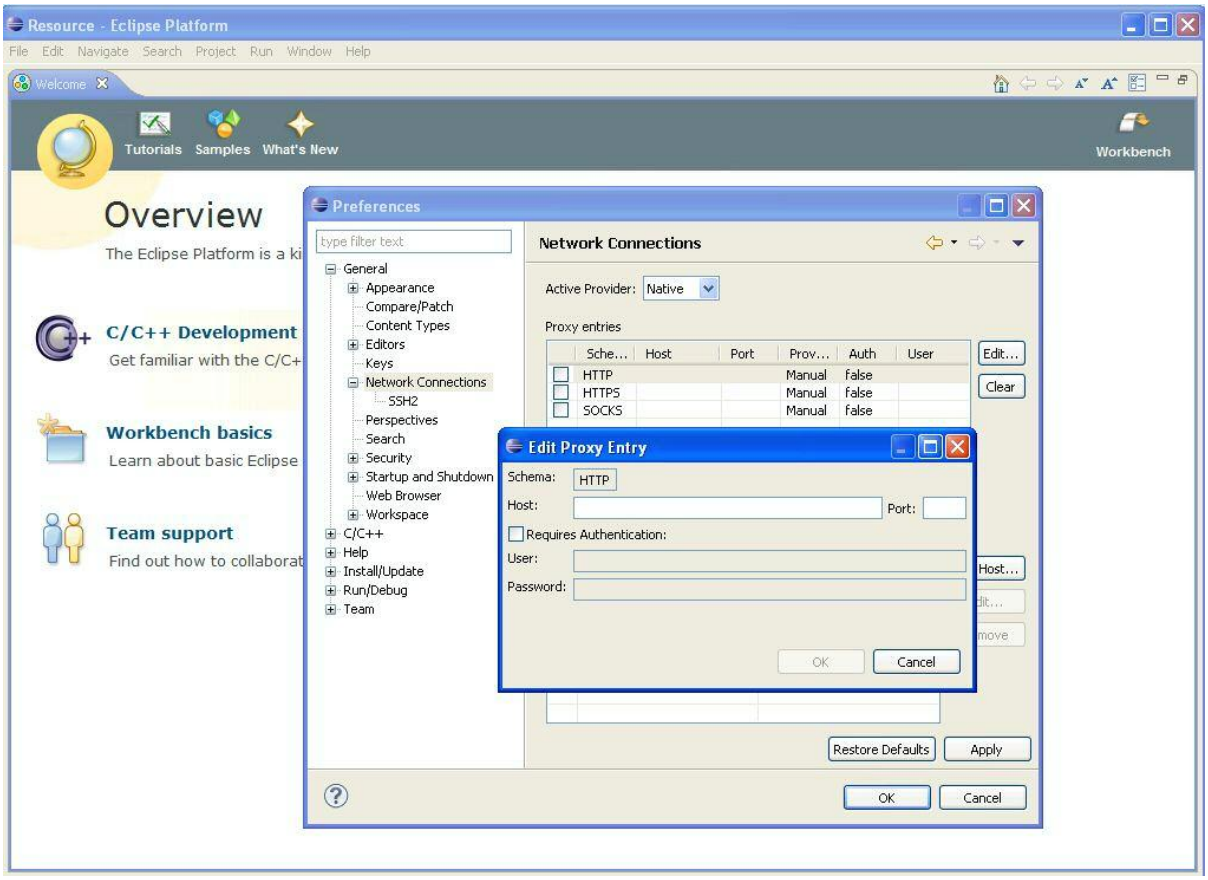
The online method is recommended. For this procedure first adapt the Eclipse network settings to the network configuration before initiate the installation procedure.

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.



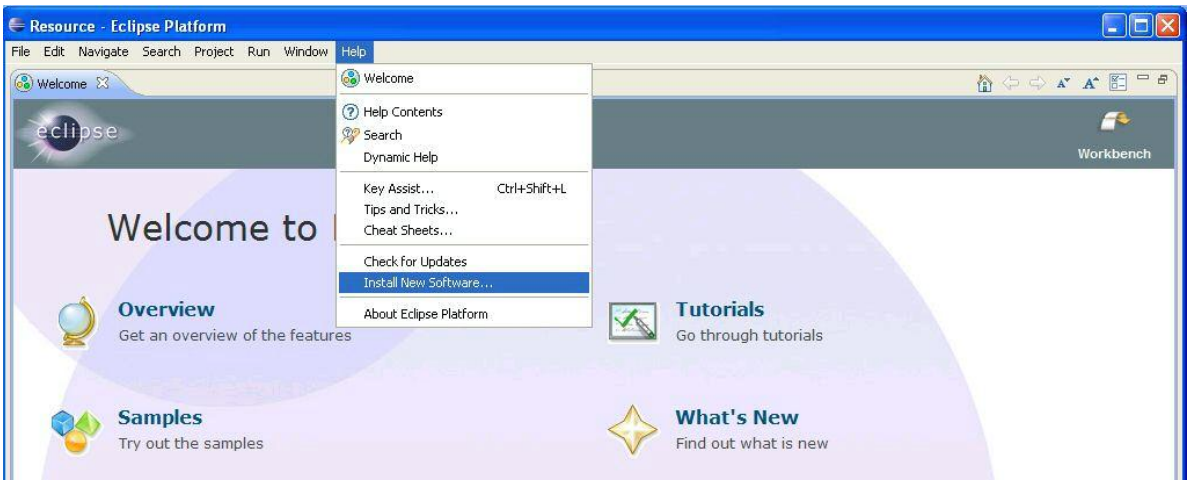
After this change click the *Apply* button to save the new network configuration. Now the online installation of the CDT plug-in can be done.

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.

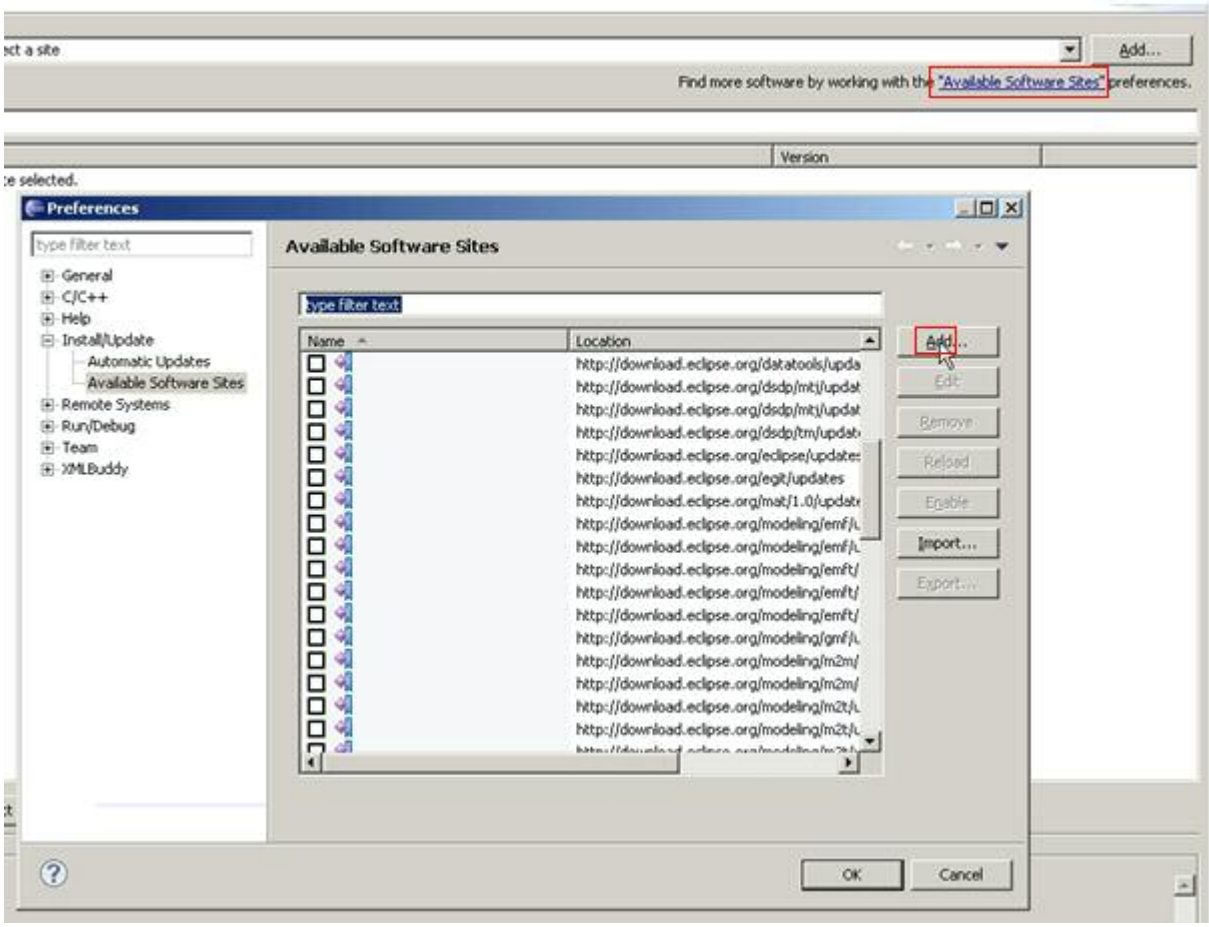
For the integration of new CDT plug-ins on eclipse-platform, the demonstration of this installation follows below.

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

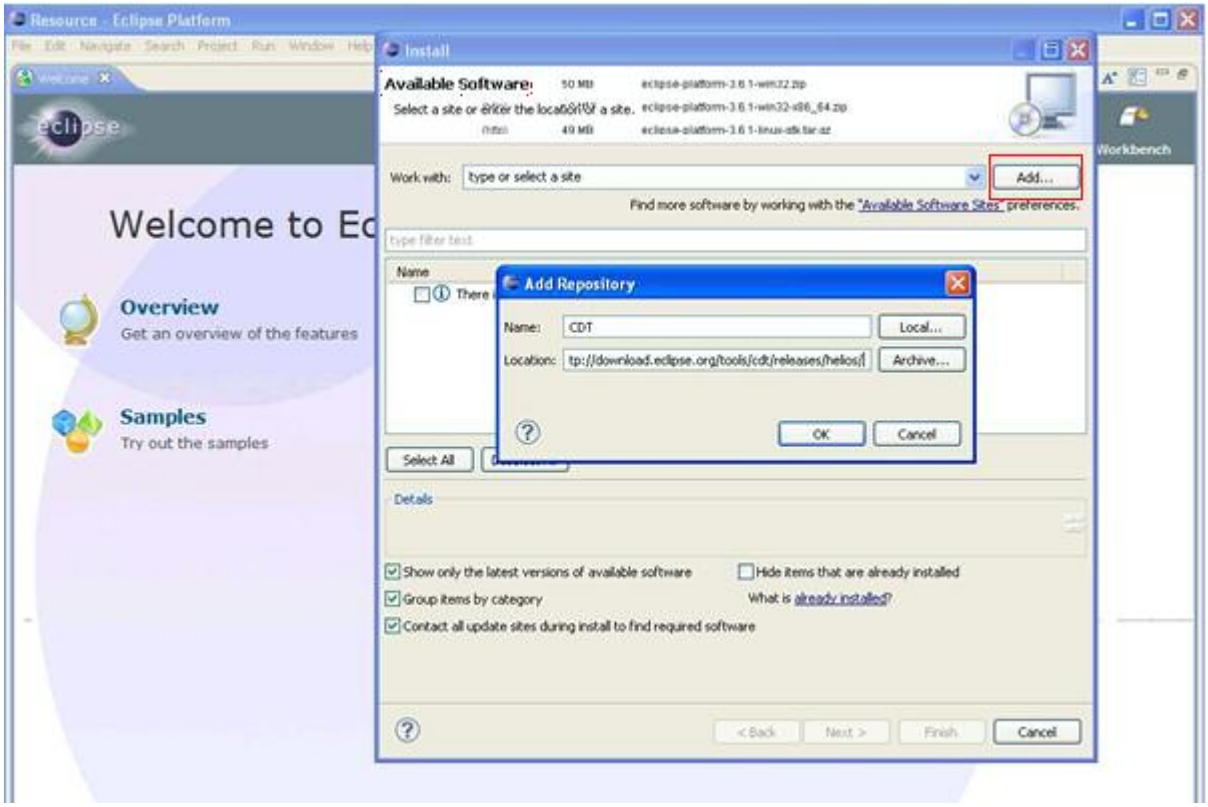


On the next window, click on *Available Software Sites* to look for a CDT downloading mirror, if existing. The mirror is:

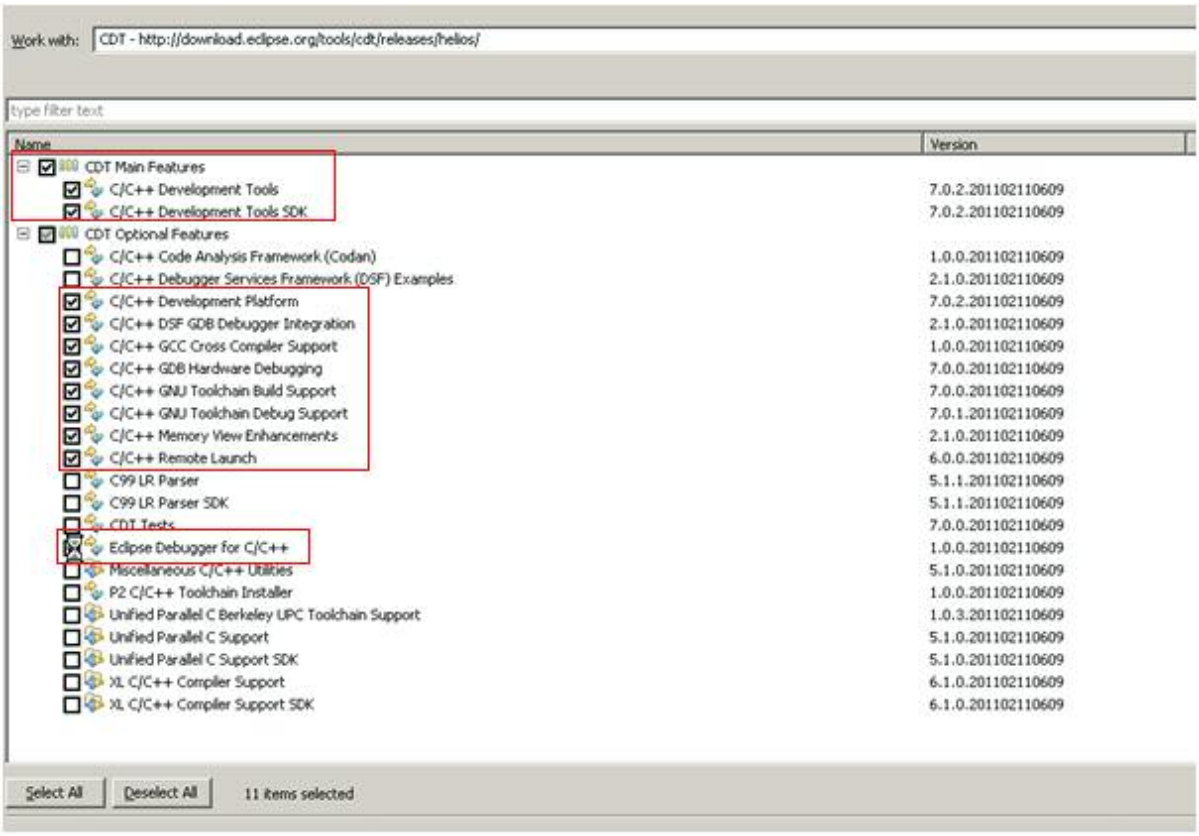
http://download.eclipse.org/tools/cdt/releases/helios/".



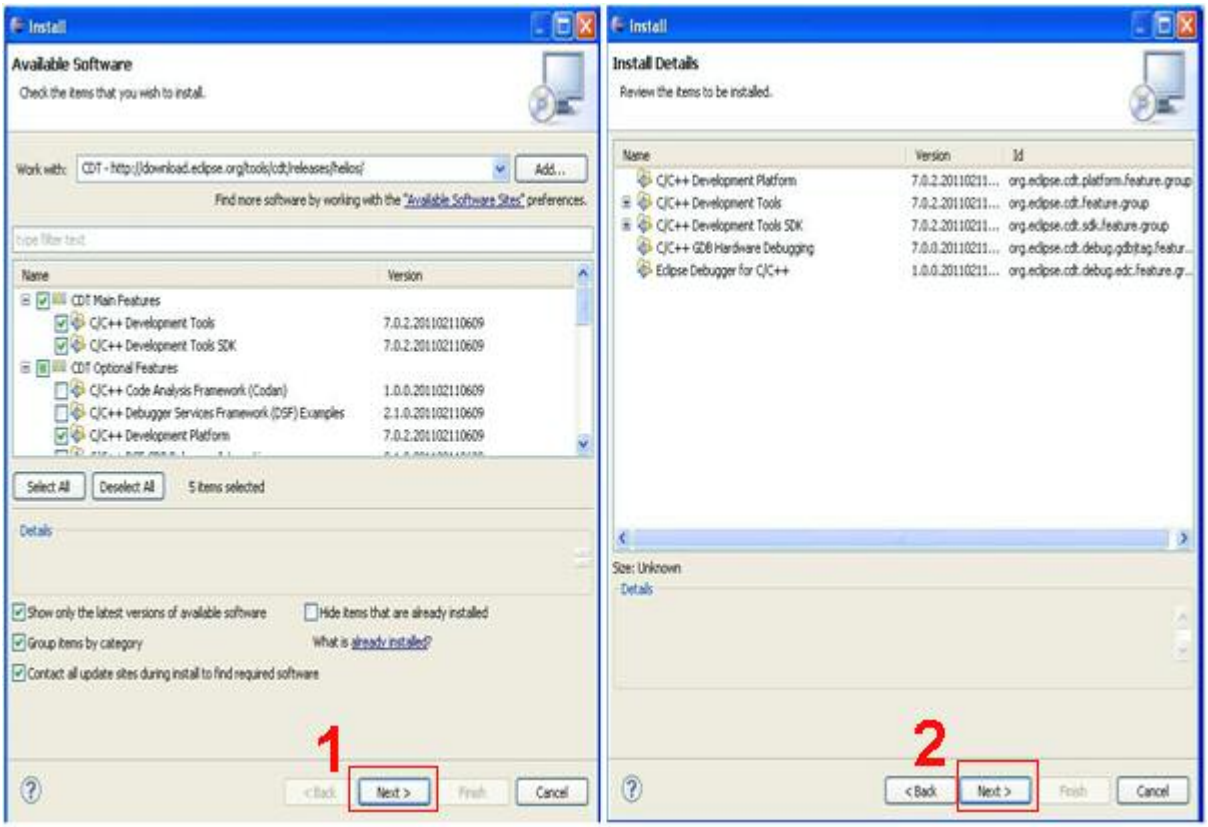
Otherwise click on *Add* to set the required mirror. Enter *CDT* for the name and http://download.eclipse.org/tools/cdt/releases/helios/ for the web location.

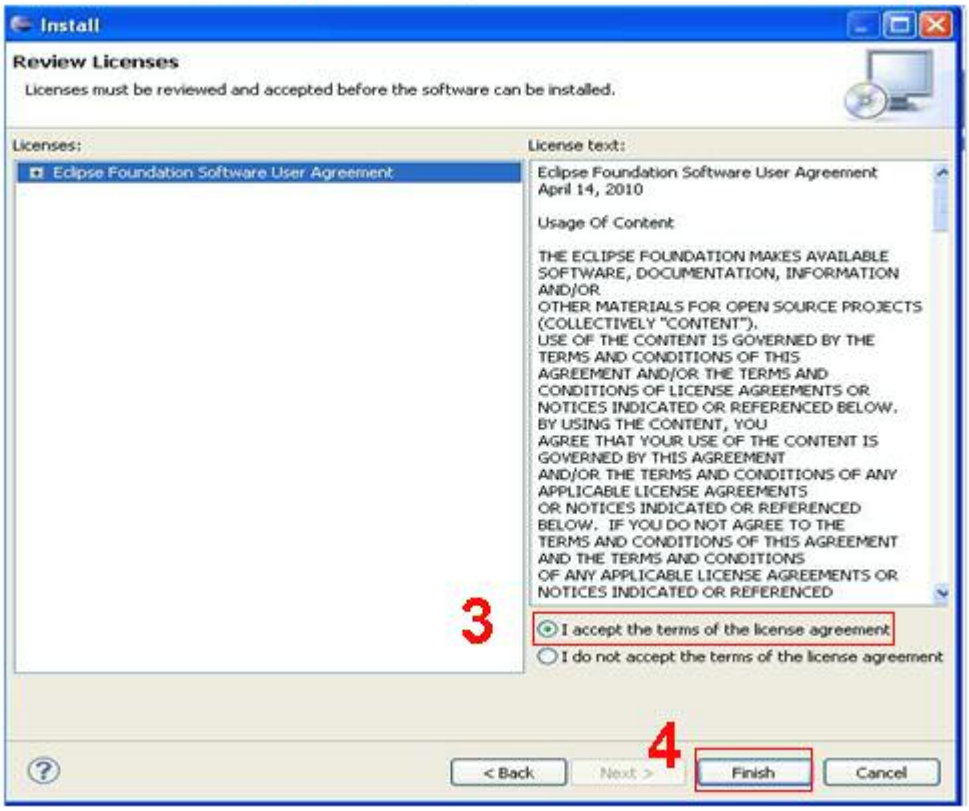


Click on *OK* and the next window below will be displayed. Select both *CDT MAIN Features* and the *CDT Optional Features* listed below only.

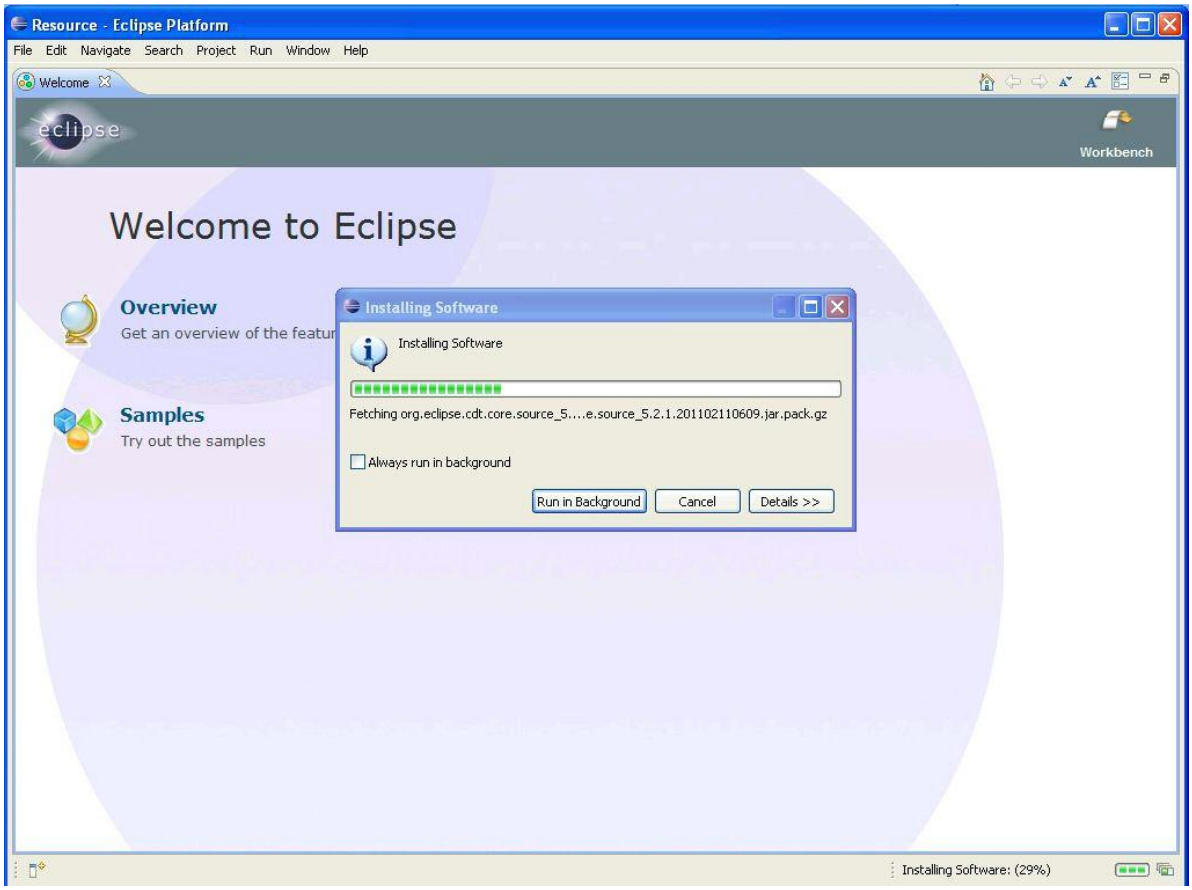


Follow the next steps to start the plug-in installation.





Eclipse starts then the installation of CDT plug-in.



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

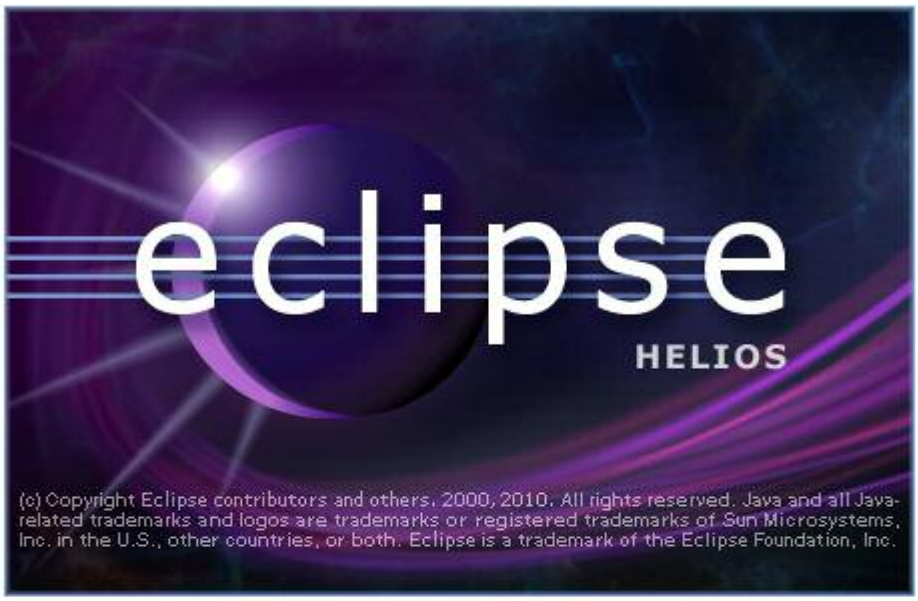
8 Working with the Eclipse IDE

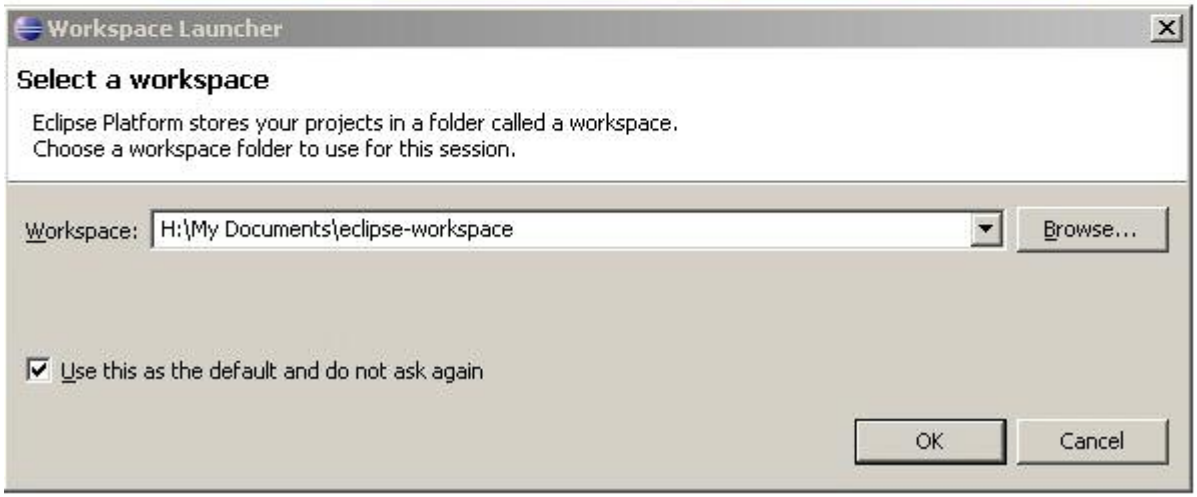
HOW TO HANDLE THE ECLIPSE IDE

8.1 C/C++ perspective

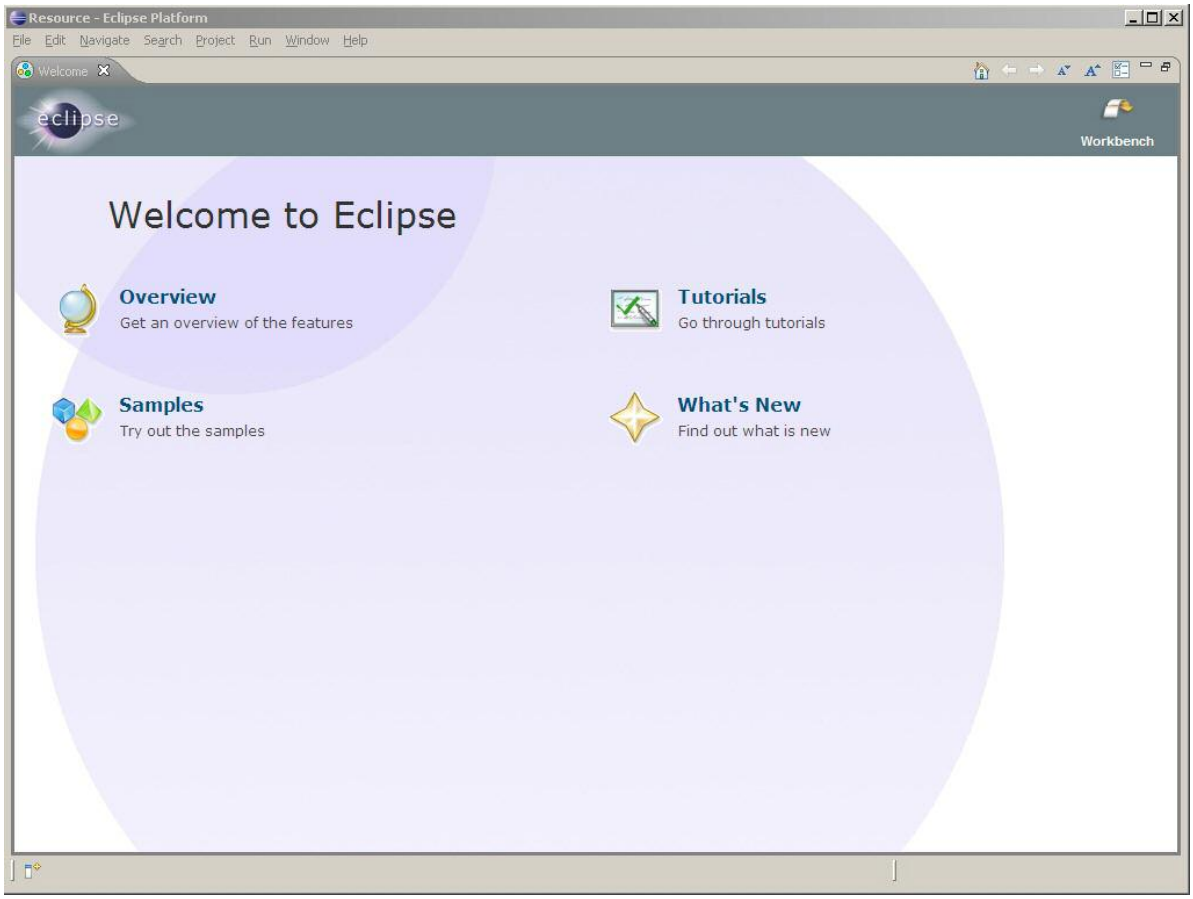
Start the Eclipse IDE.

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).



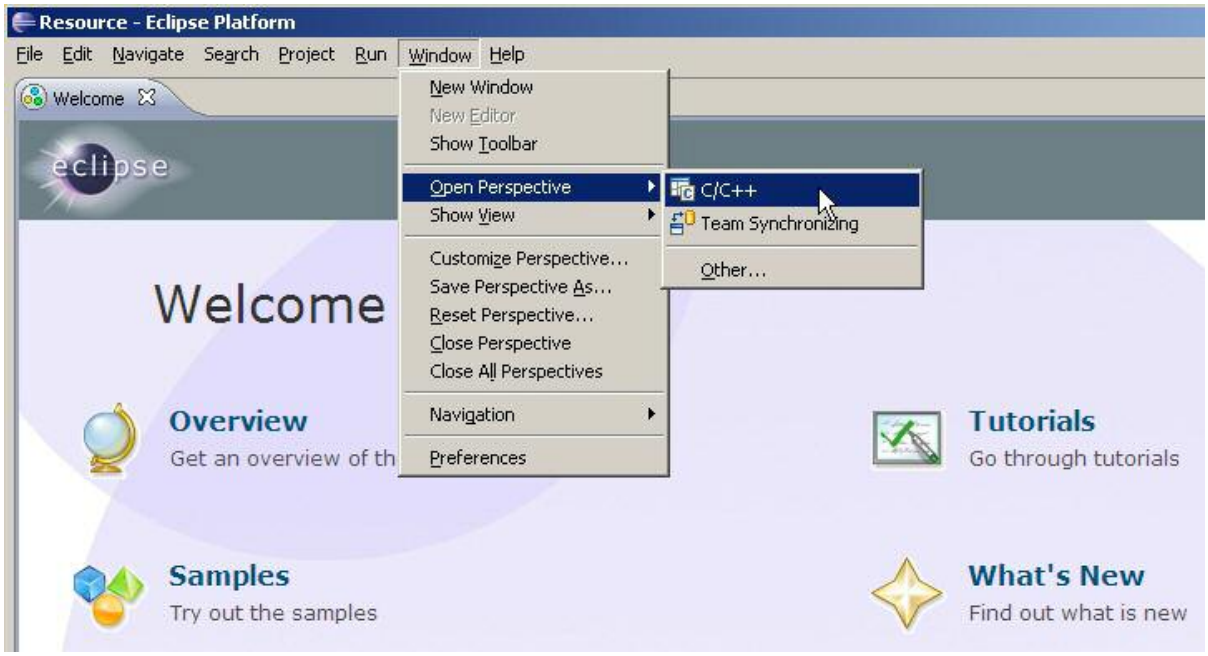


Now Eclipse will officially start and show the “Welcome” page shown below.



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

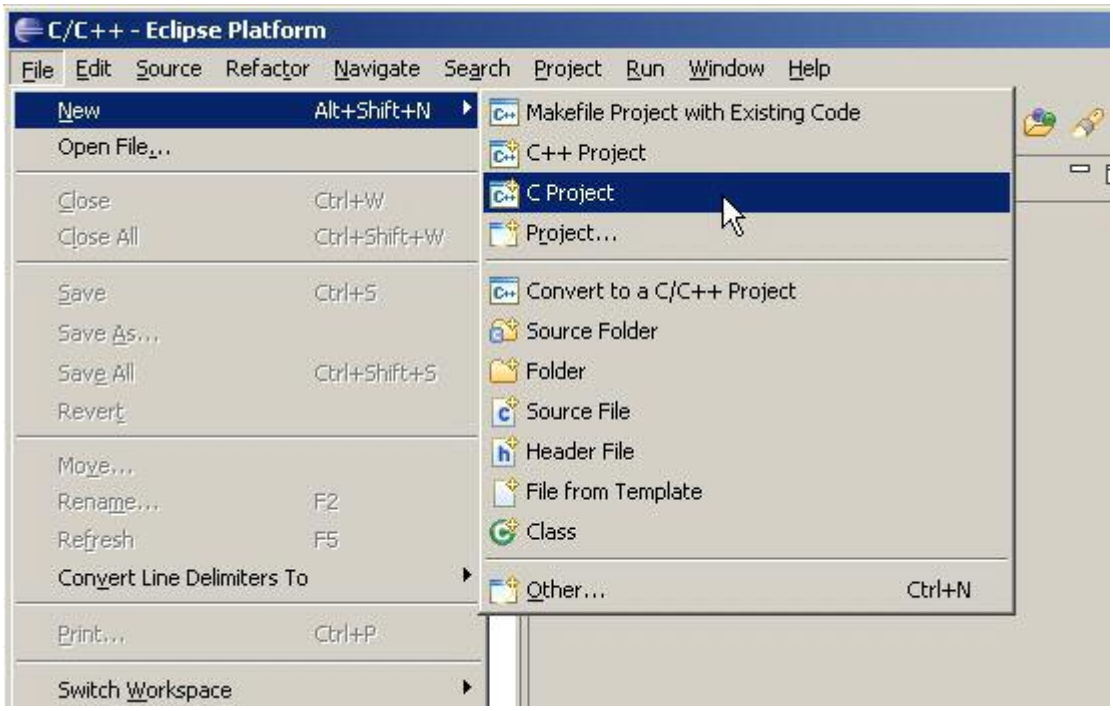
Choose *Window→Open Perspective*, then click on *C/C++* to open Eclipse in the C/C++ perspective.



8.2 Creating a C or C++ project with Eclipse

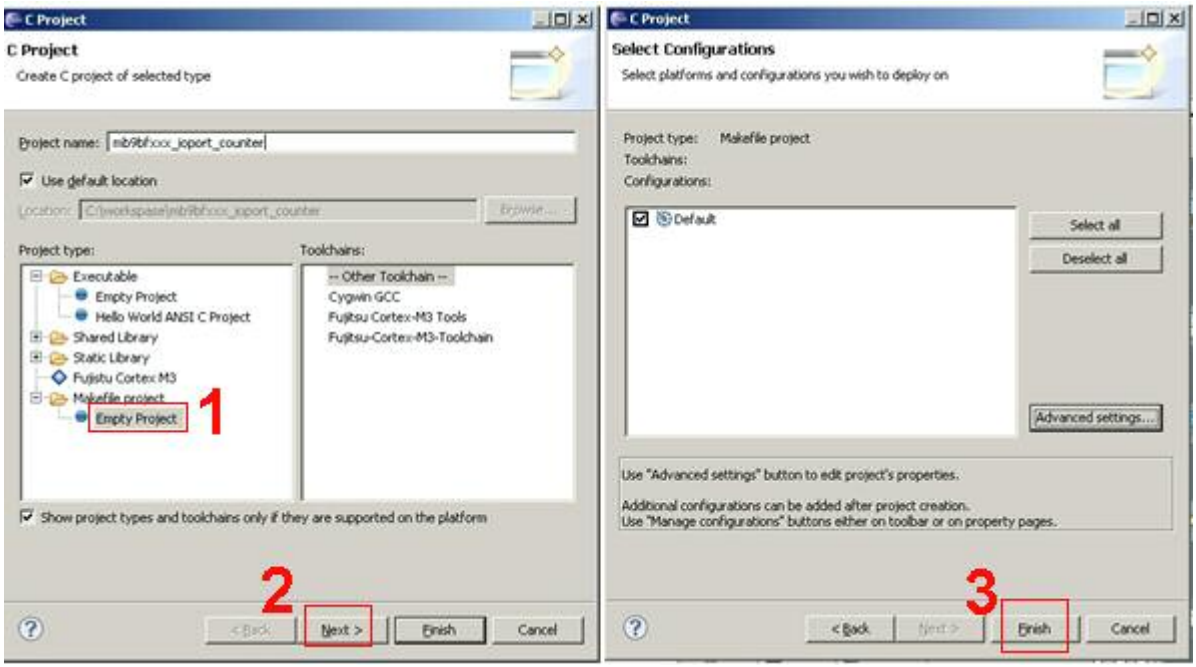
In the Eclipse C/C++ perspective a new project for your target can be created, here: Fujitsu Cortex M3.

For this choose *File→New→C Project*.



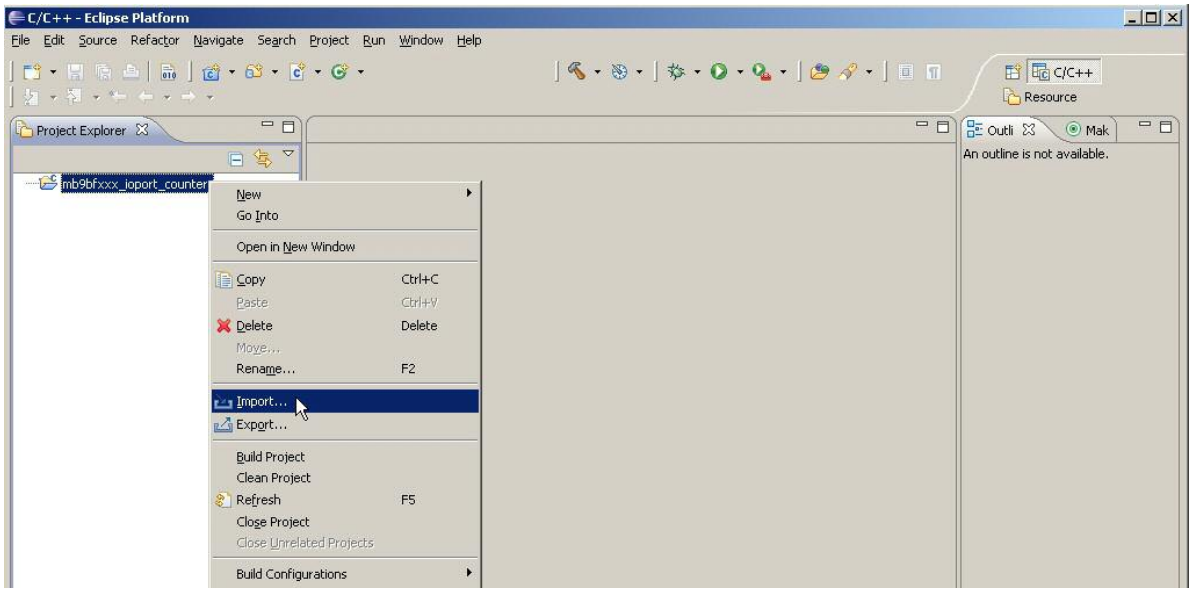
In the "New Project wizard" shown below-left, expand the *Makefile project* branch by clicking on its "+" sign and then select *Empty Project*. Enter the sample project name e.g. "mb9bfxxx_ioport_counter". Then click on *Next* to continue.

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

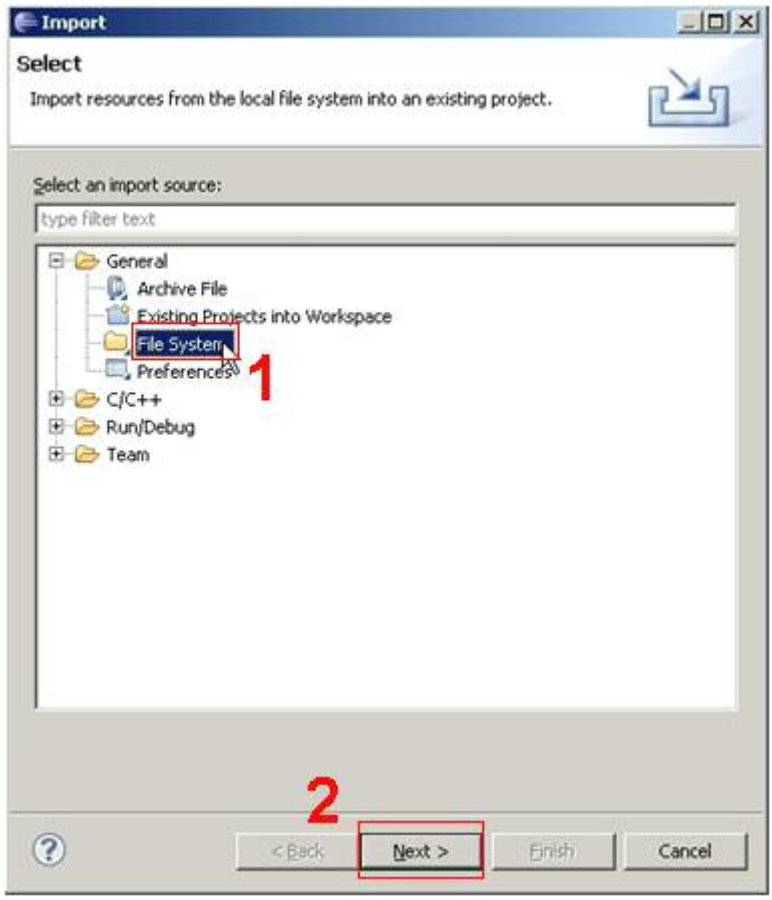


Now the C/C++ perspective shows a valid project, as shown below in the C/C++ Projects view on the left, but there are no source files in that project. Normally you would select *File→New→Source File* and enter a file name and start typing. This time, however, we will import already existing source files.

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

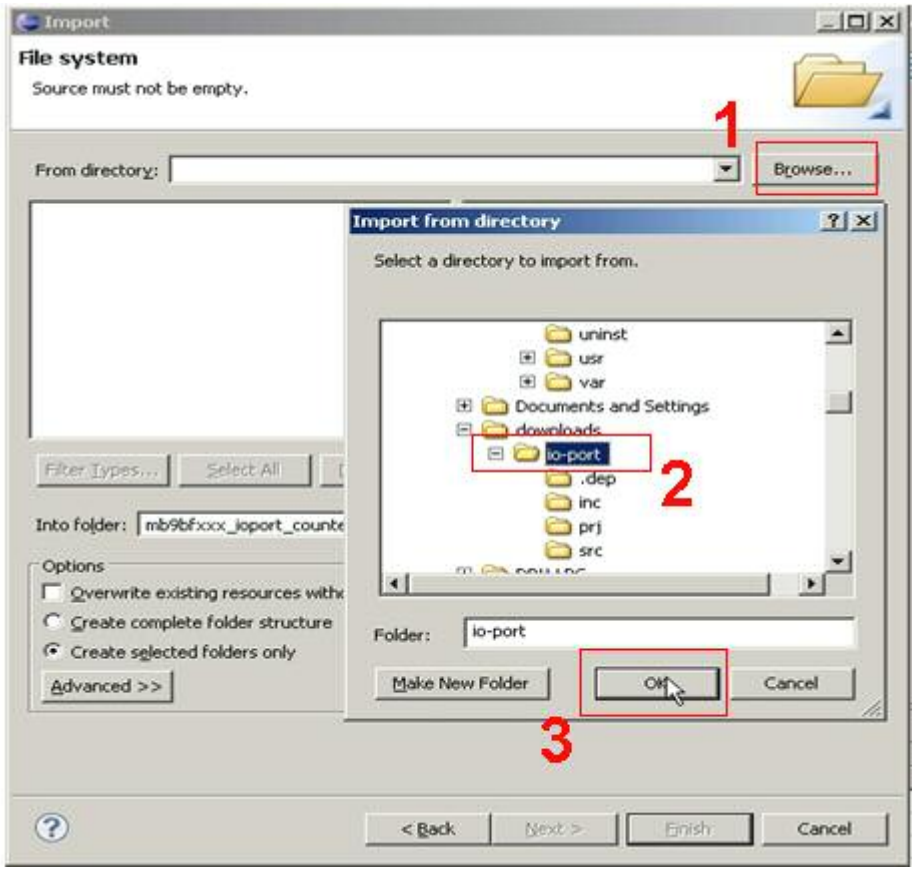


In the “Import” screen below, click on *File System* and then click *Next* to continue.

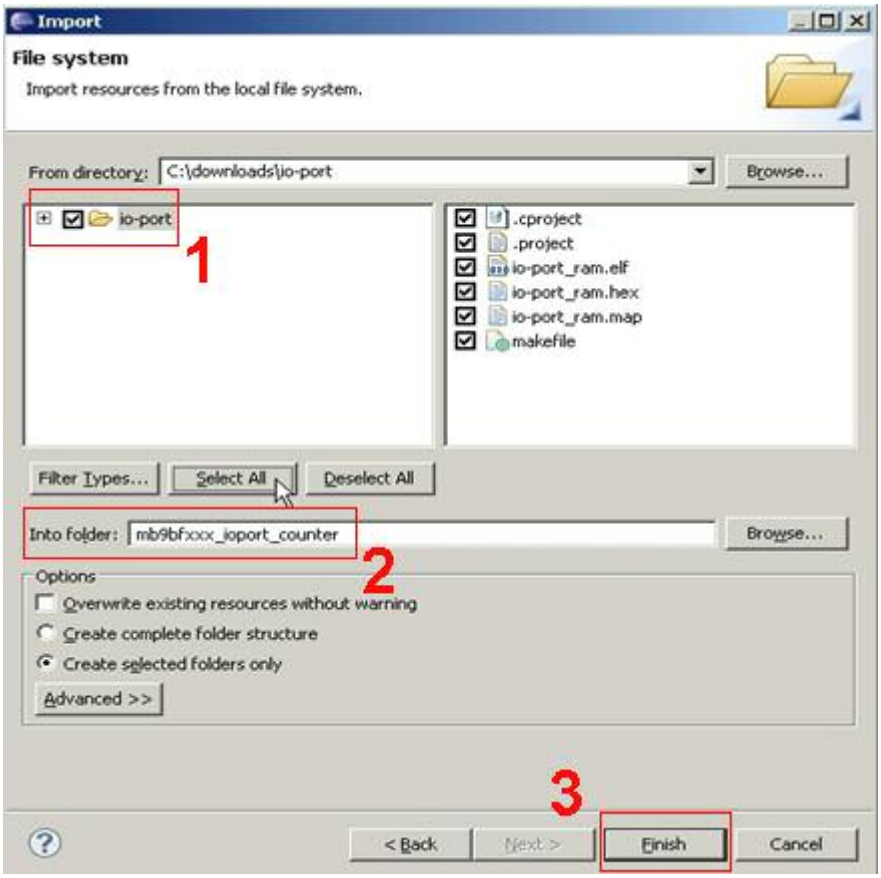


In the *Import→File System* screen below, use the *Browse* button associated with the *From directory* text box to search for the sample project to be imported.

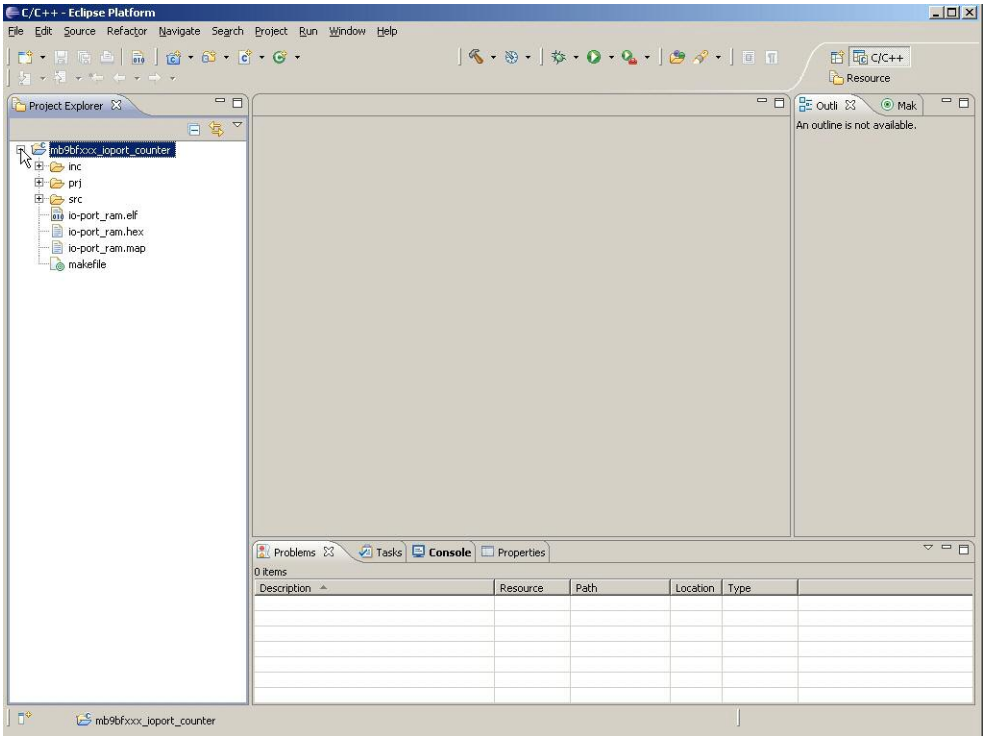
The project template *io-port* used in this application note, which is included in the note’s software package archive. The sample project *io-port* should be then saved, in a directory folder e. g. *C:\downloads\io-port*.



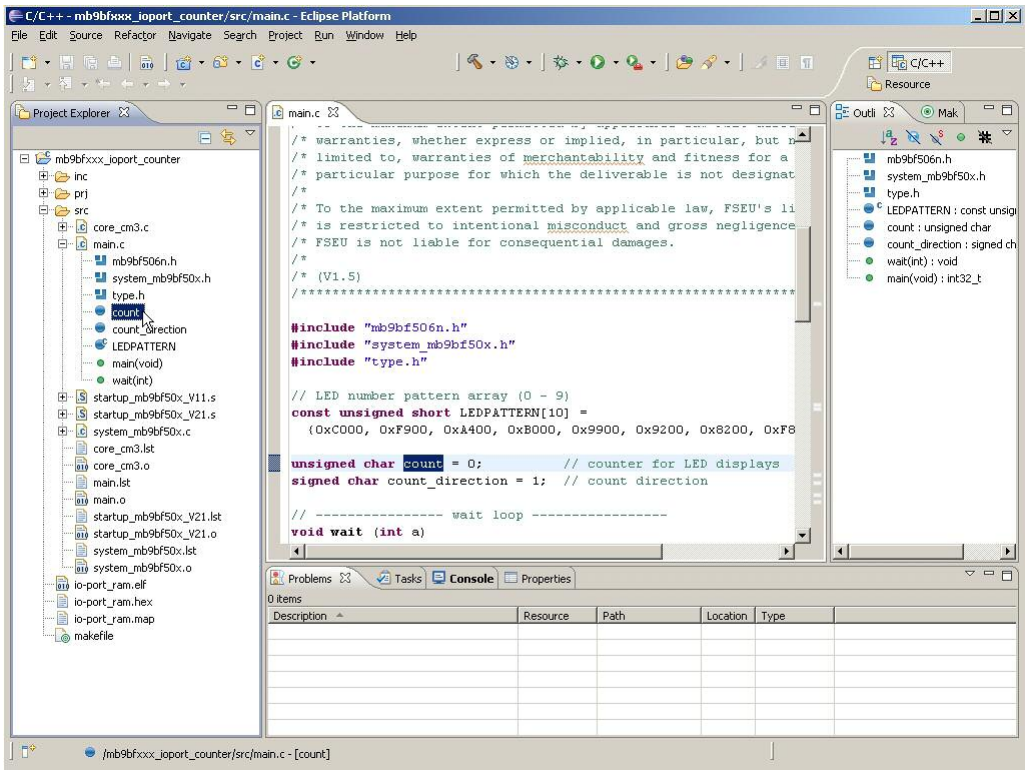
Check the box for the folder of the *io-port* example and then click the ***Select All*** button below because we want to import each of its files. Click *Finish* to start the file import operation.



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.

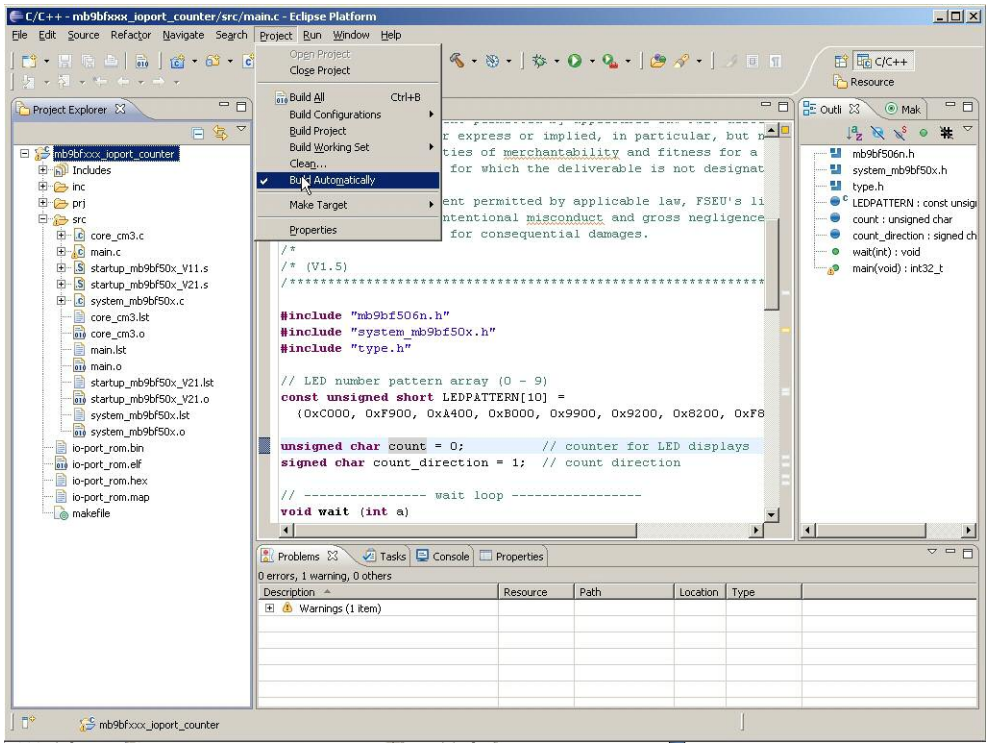


In the Eclipse window below, the *main.c* file has been selected by clicking on it and it thus be displayed in the source file editor view in the centre. In the project explorer window the *main.c* module is expanded to reveal its variables and functions. By clicking e. g. on the variable `count`, the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

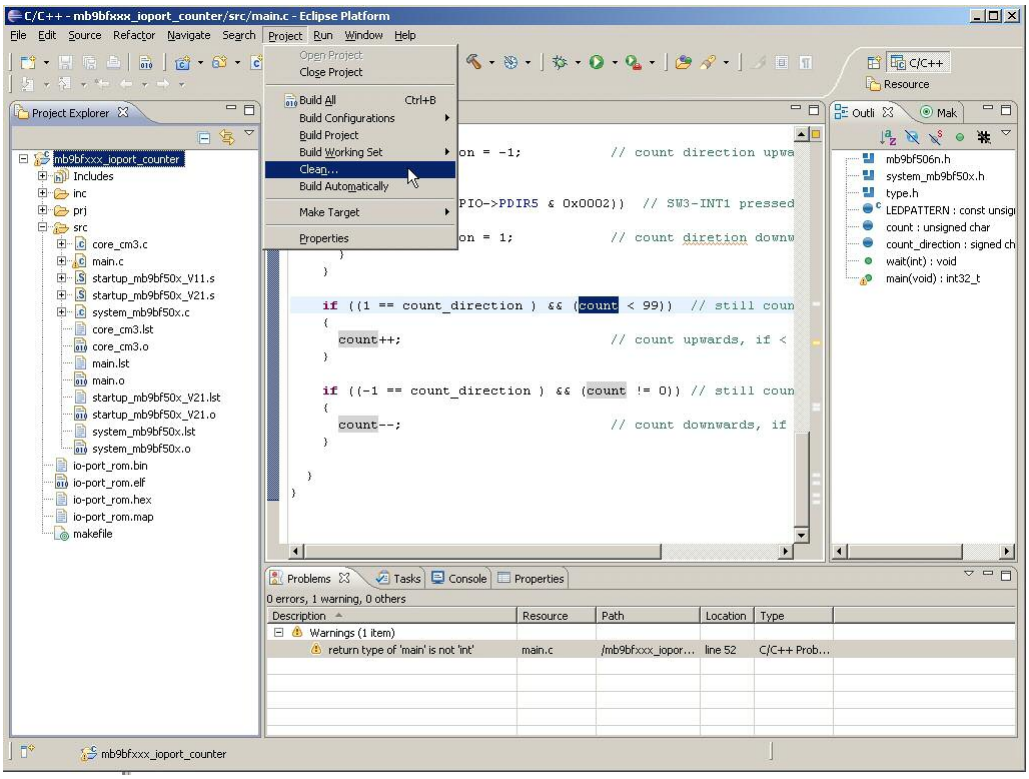


8.3 Cleaning the selected project

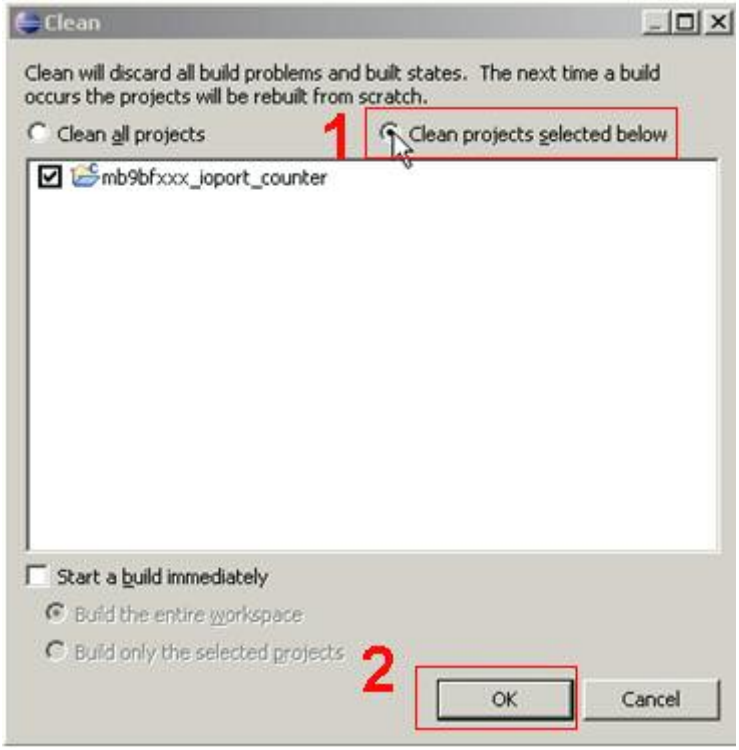
For compiling a project, first disable the automatically build. Select the project and from the category *Project* on the IDE menu uncheck *Build Automatically*.



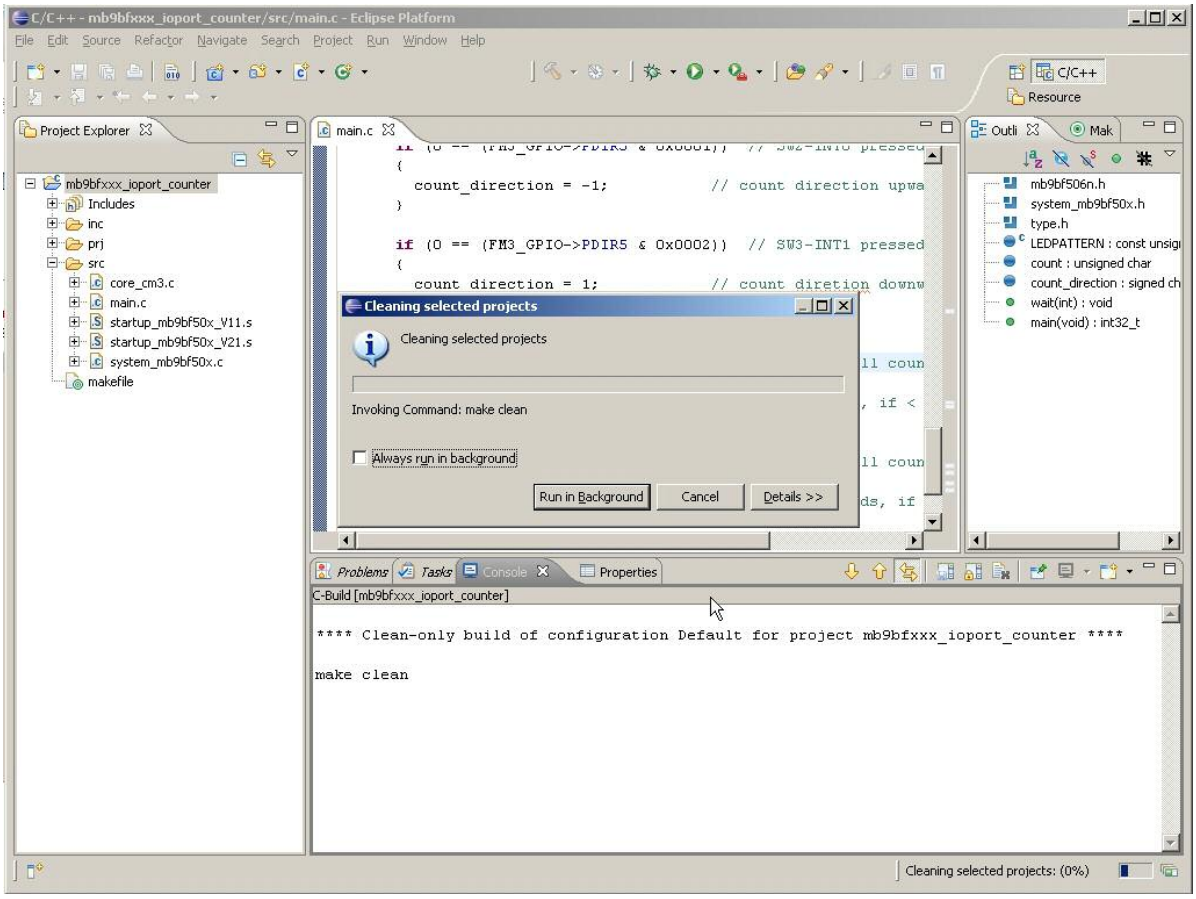
Now clean the project. In the same way select the project *mb9bfxxx_ioport_counter* from the project explorer window the category *Project*, and on the IDE menu choose *Clean...*.



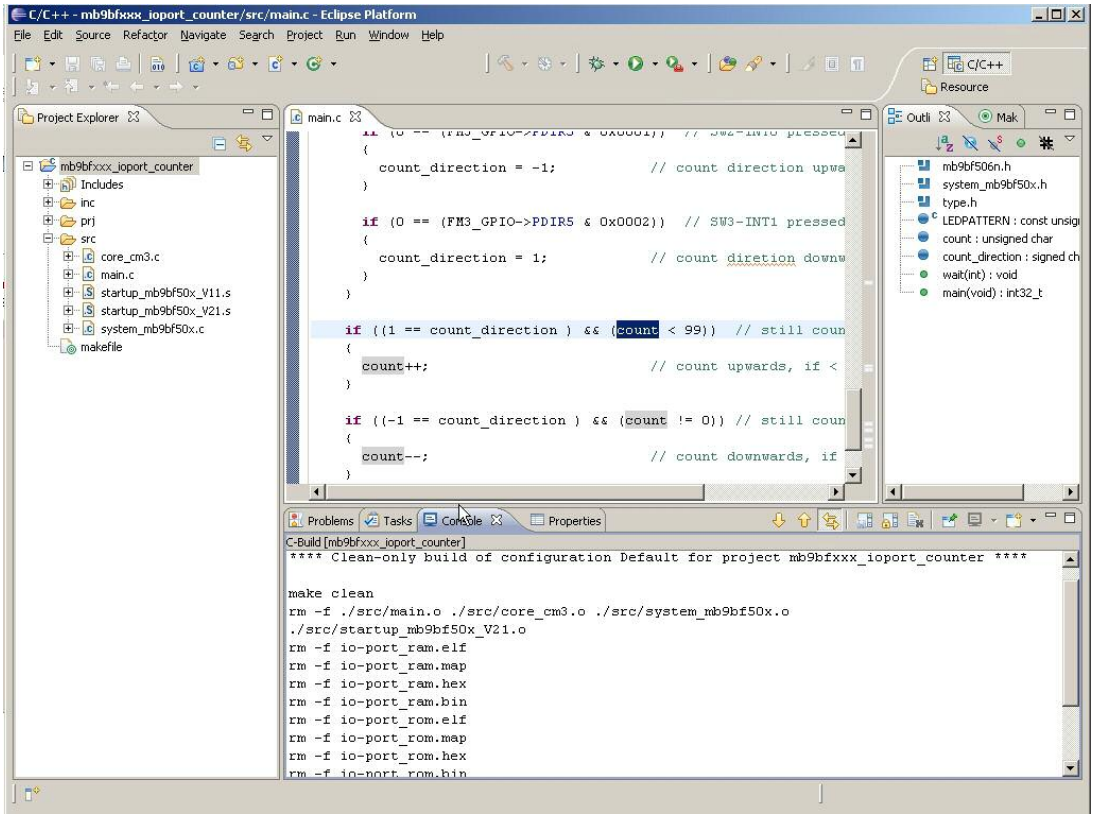
On the clean window deselect the option *Clean all projects* and select our project. Deselect also the option *Start a build immediately*.



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



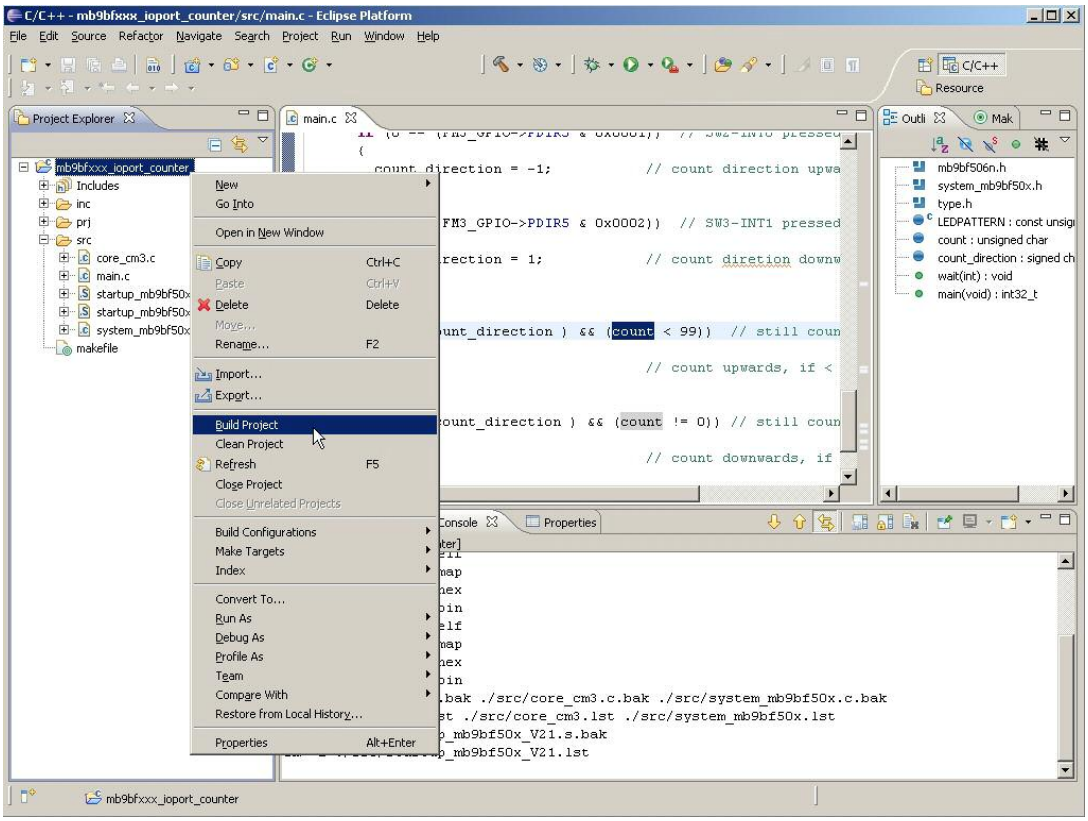
To show the results of the clean process, look at the “Console” panel located below.



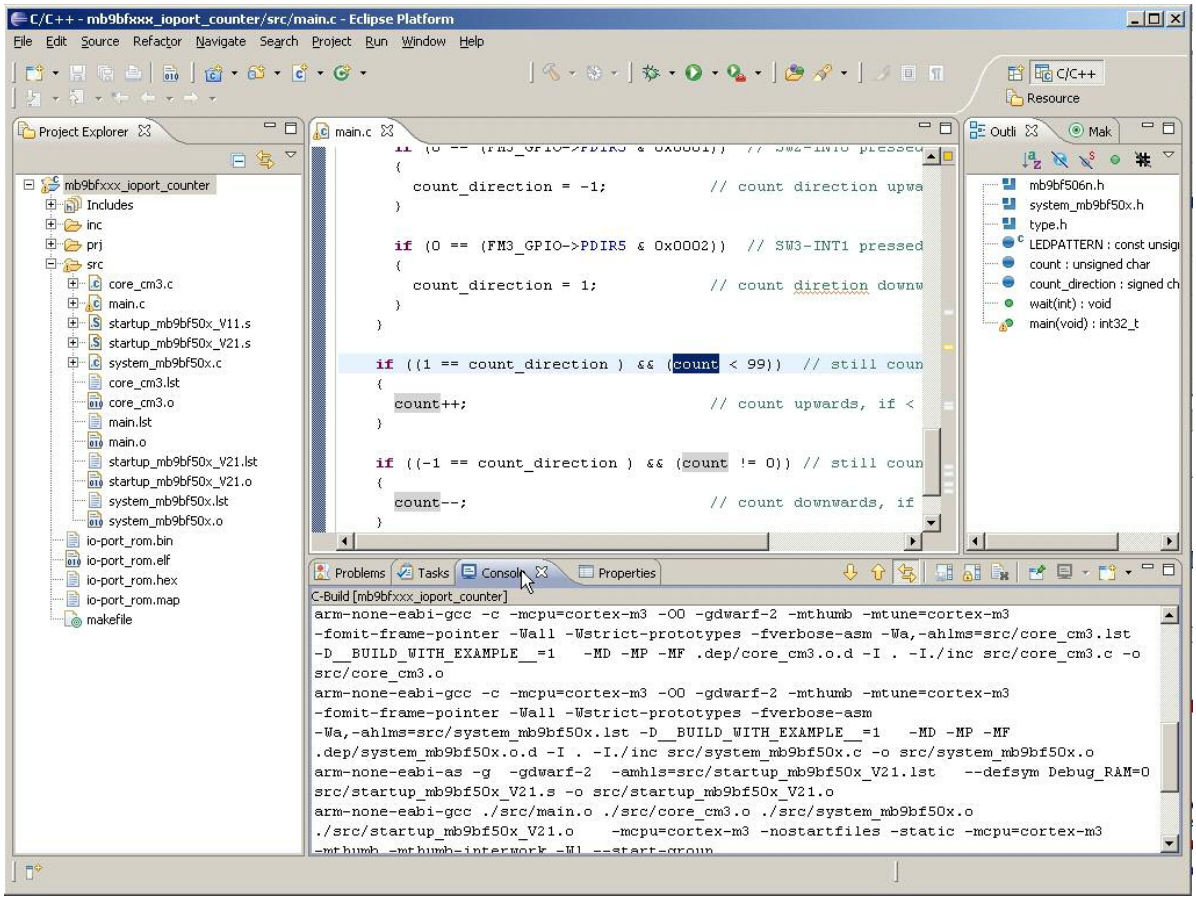
8.4 Building the selected project

Important note: If you use the makefiles 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 toolchain. 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.

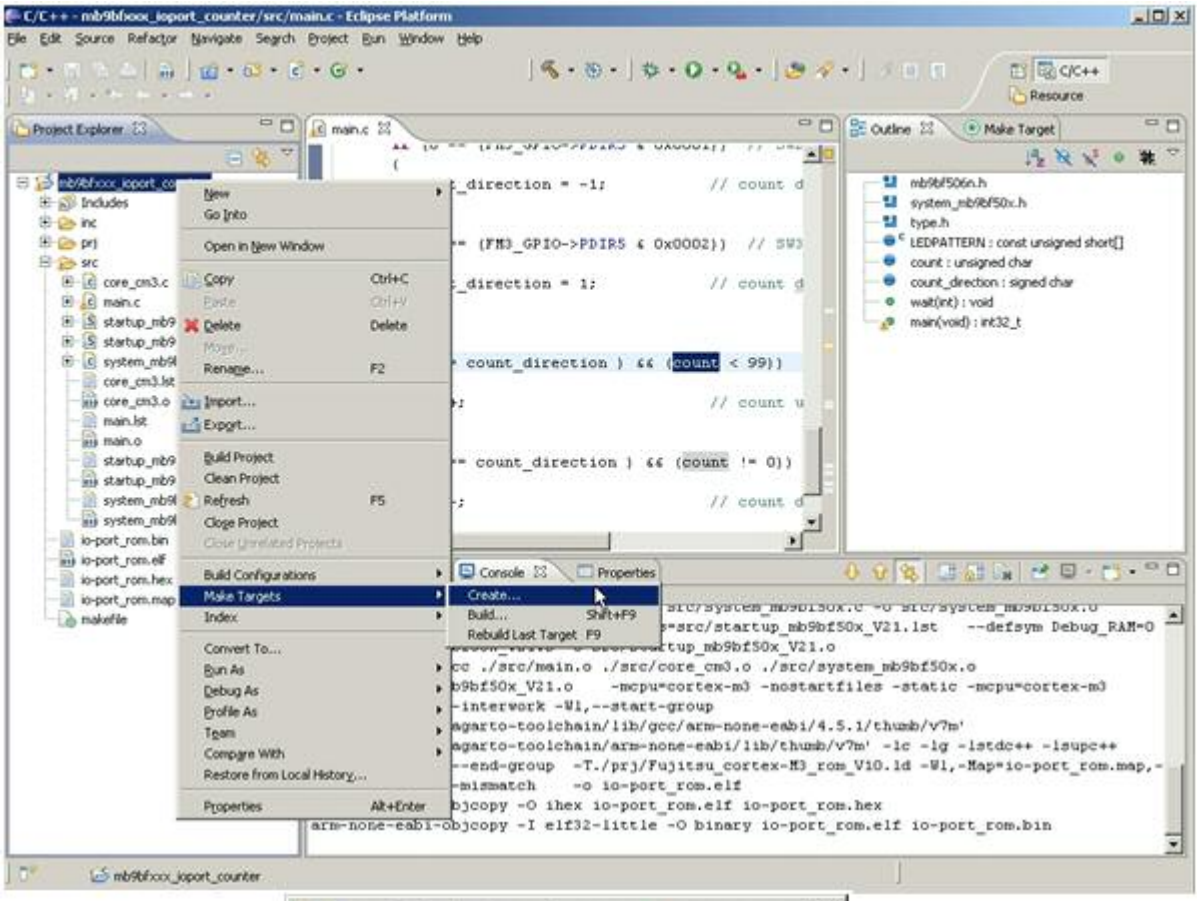
8.5 Create make target

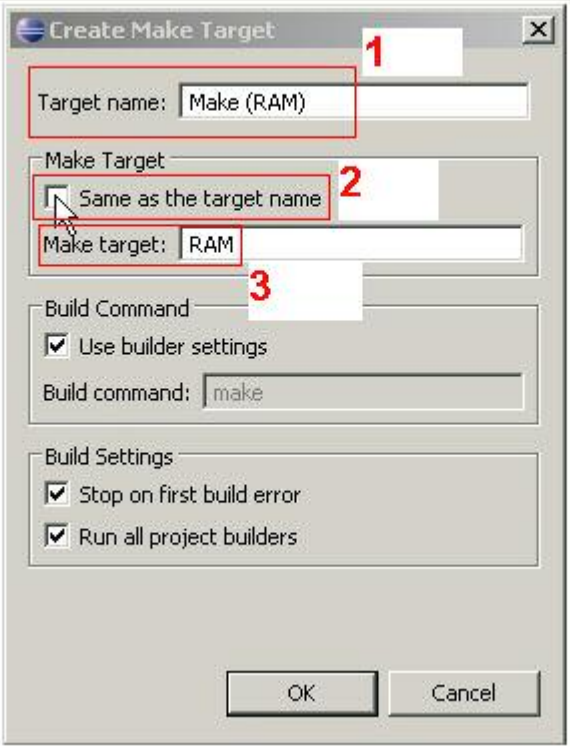
The make targets are pre-defined in the example project *mb9bfxxx_ioport_counter*. This paragraph shows the creation process, if a new project is set-up or the targets were deleted accidentially.

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 Fujitsu 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”.

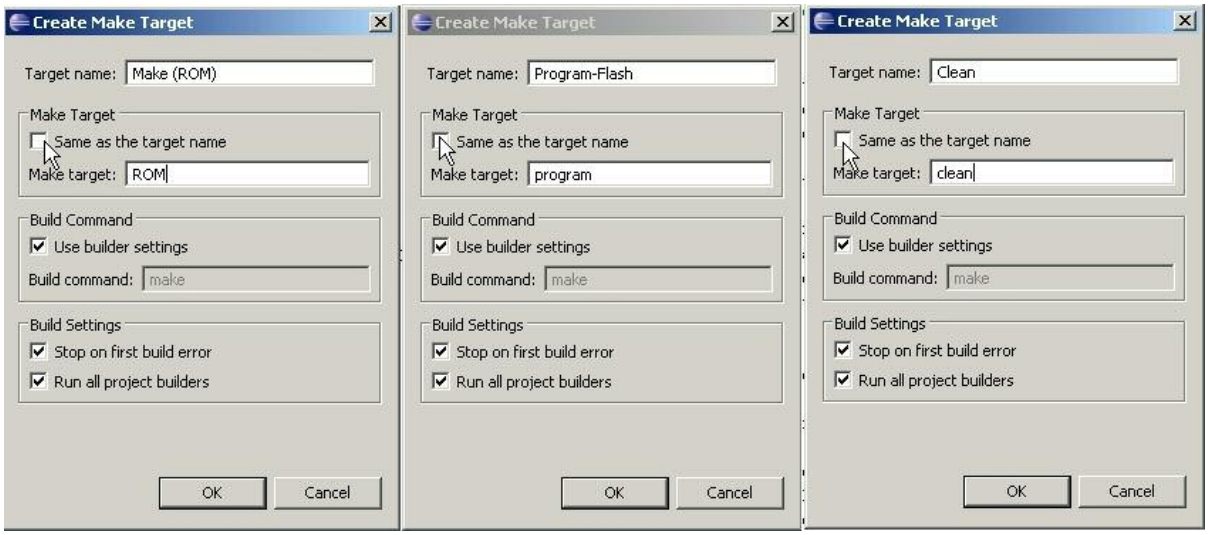
To create a make target, select the project *mb9bfxxx_ioport_counter* on the “Project Explorer” view. Click with the right mouse button on the selected project and select *Make Targets*.



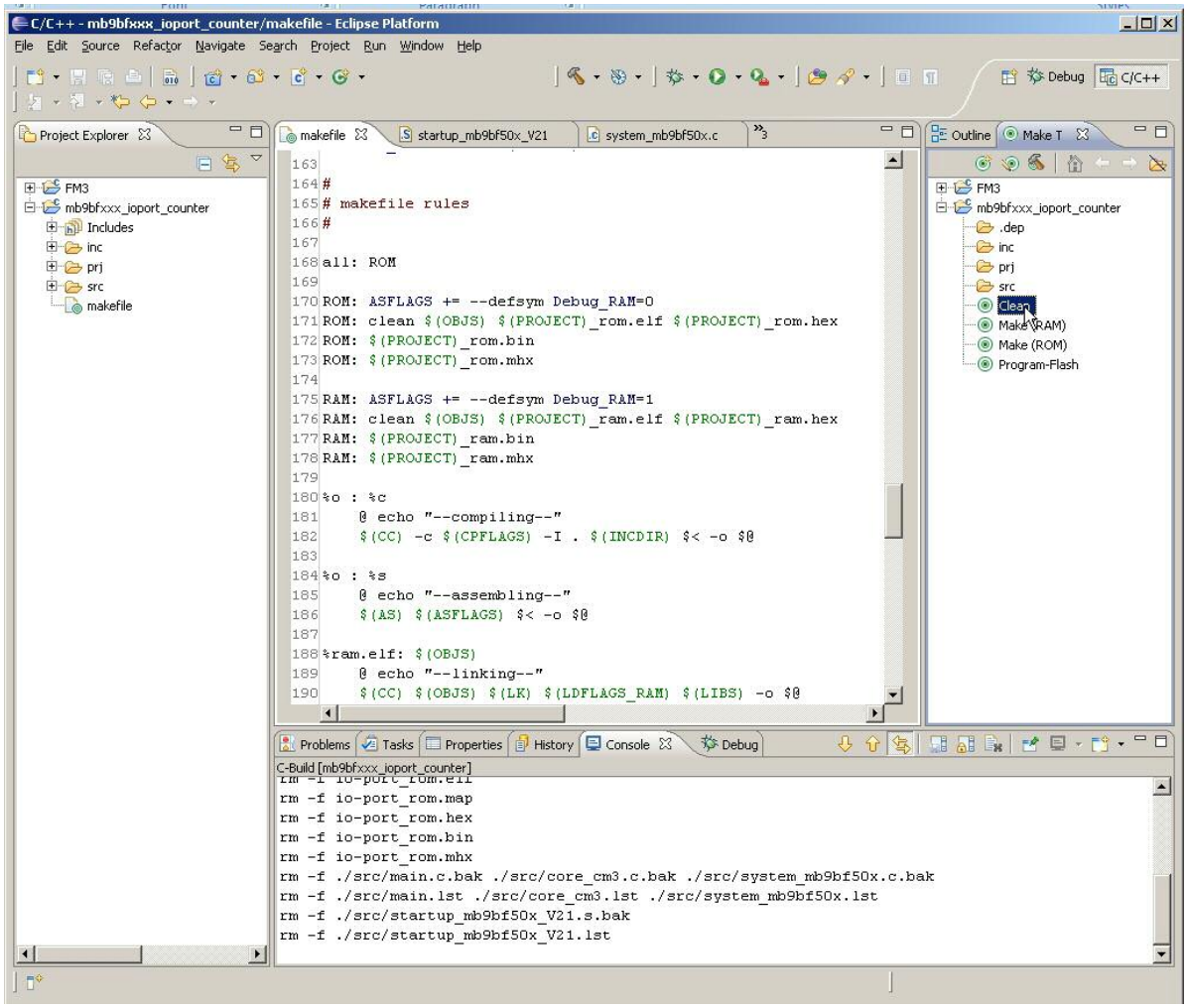


Enter “Make (RAM)” for the target name, uncheck *Same as the target name* and write “RAM” in the text box “Make target”. Click on *OK* to create a “Make (RAM)” build target.

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.

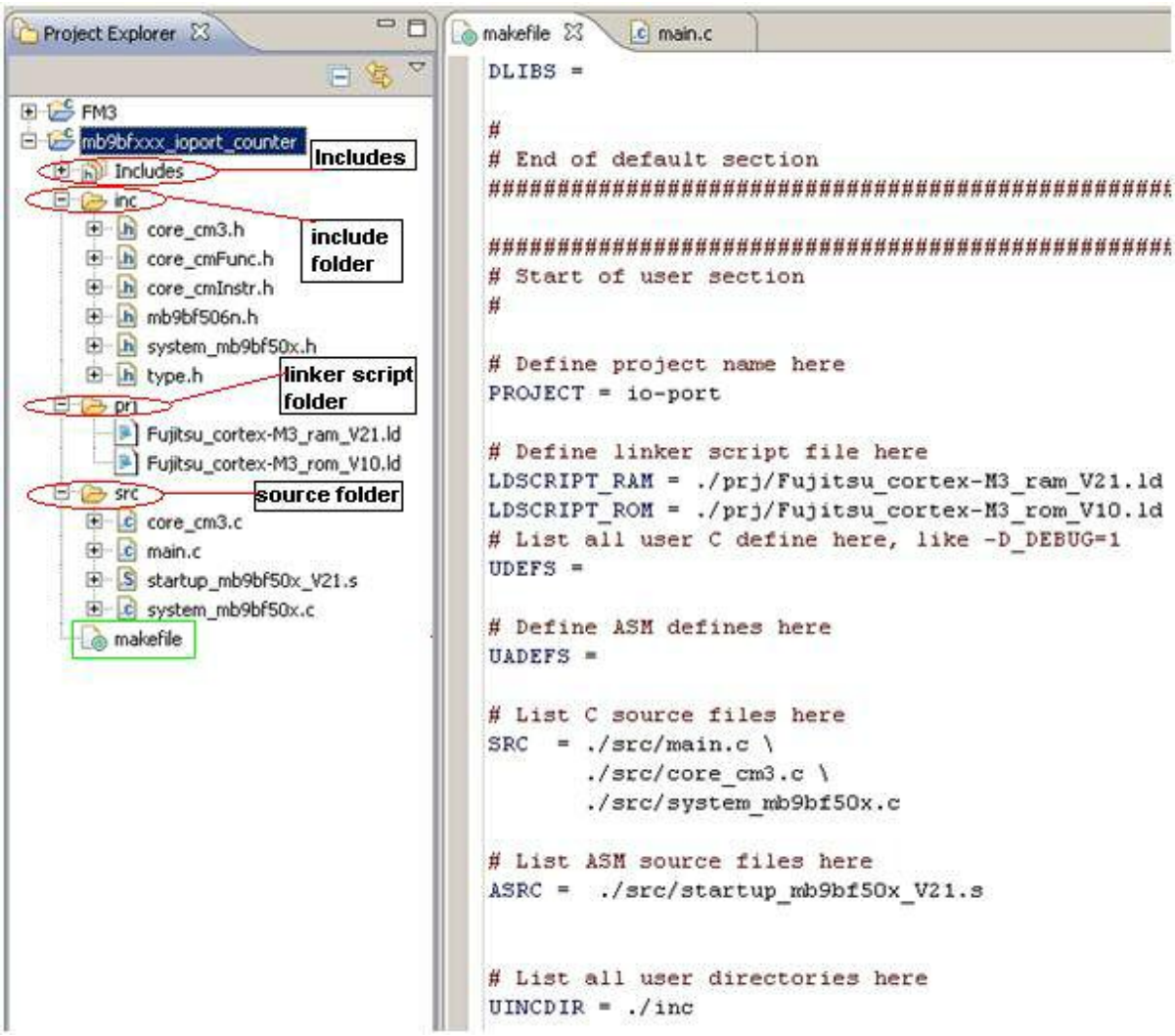


On the IDE “Console” view, the output shows that the clean process was successfully done.

9 Example Eclipse Project Template

USING THE EXAMPLE SOFTWARE PROJECTS

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.

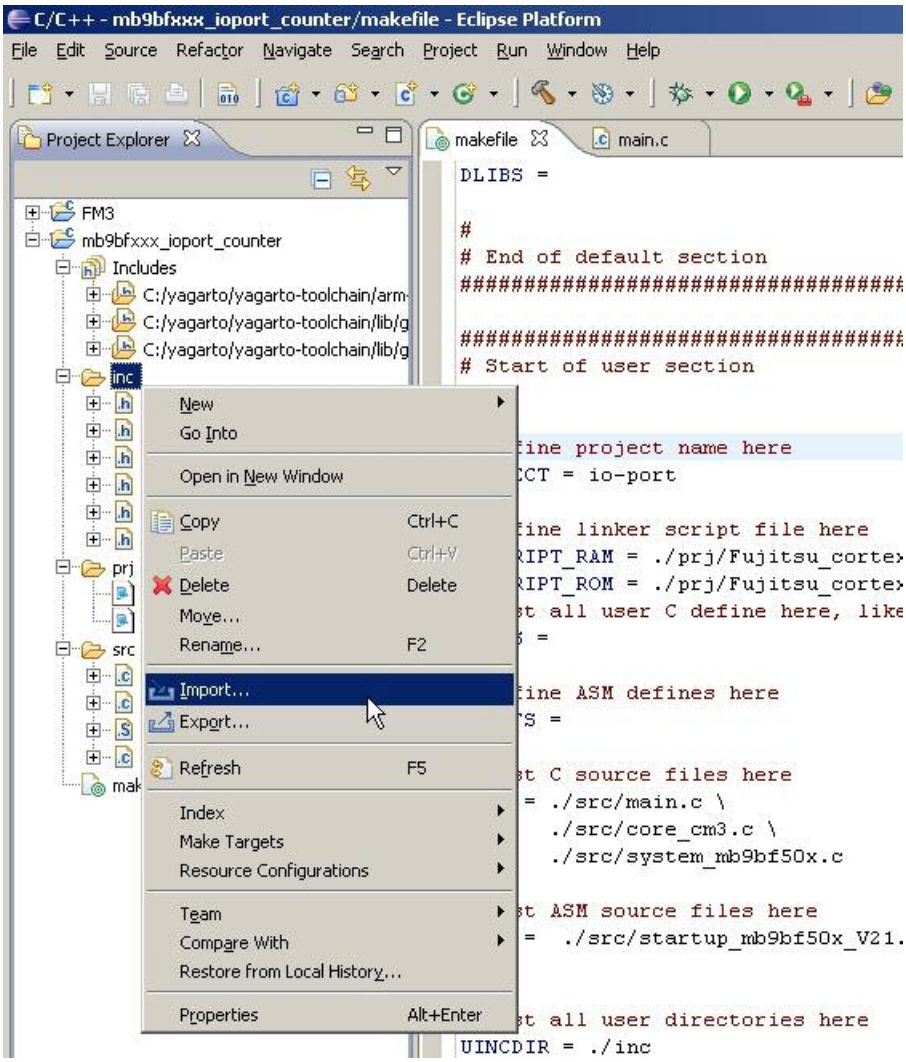
To add other sources file use the folder ***src***.

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!

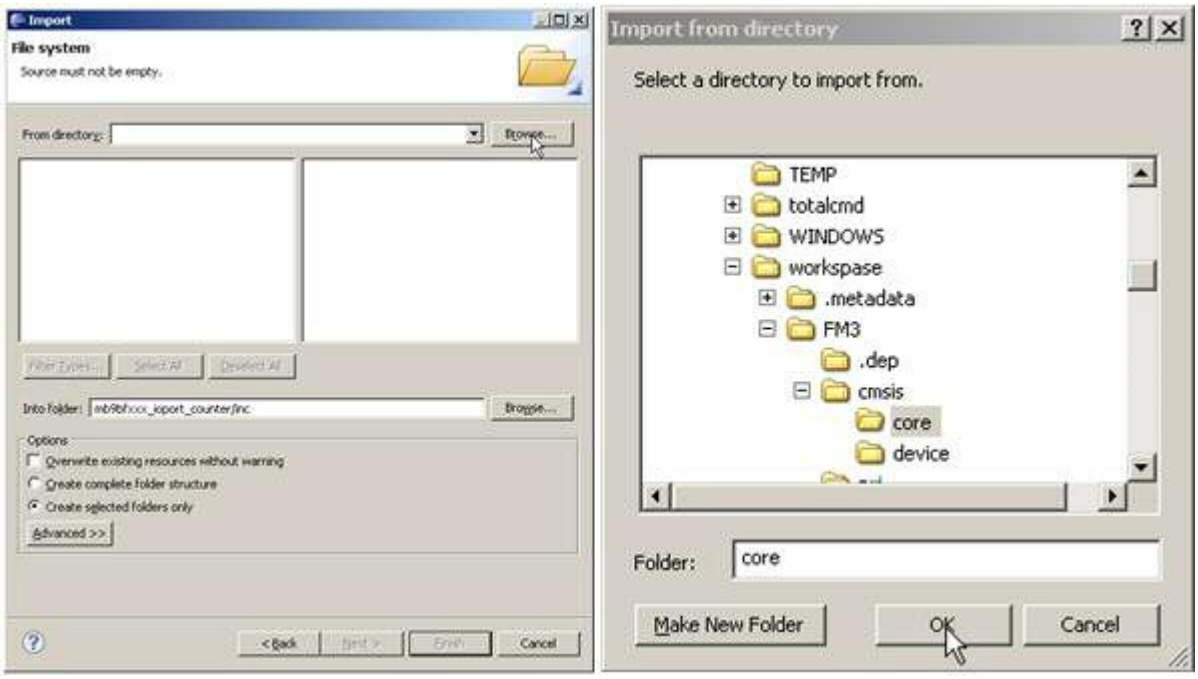
9.1 Add other Files to the Template Folder

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

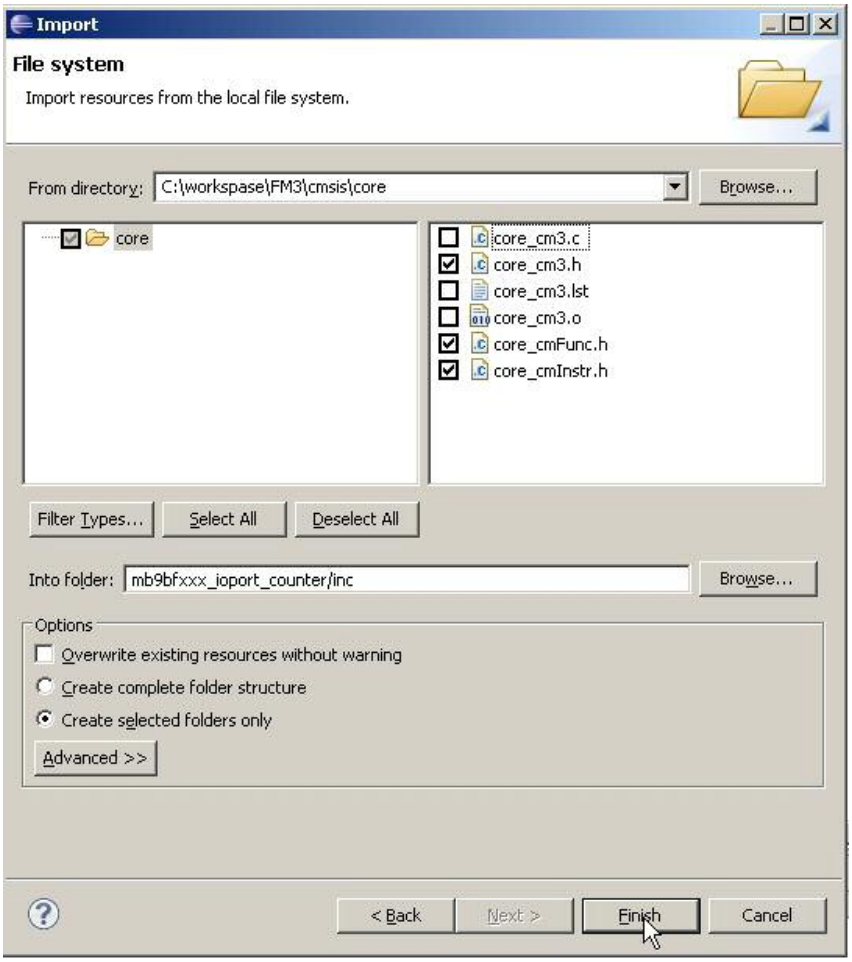


Select *File System* and click on the *Next* button.

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



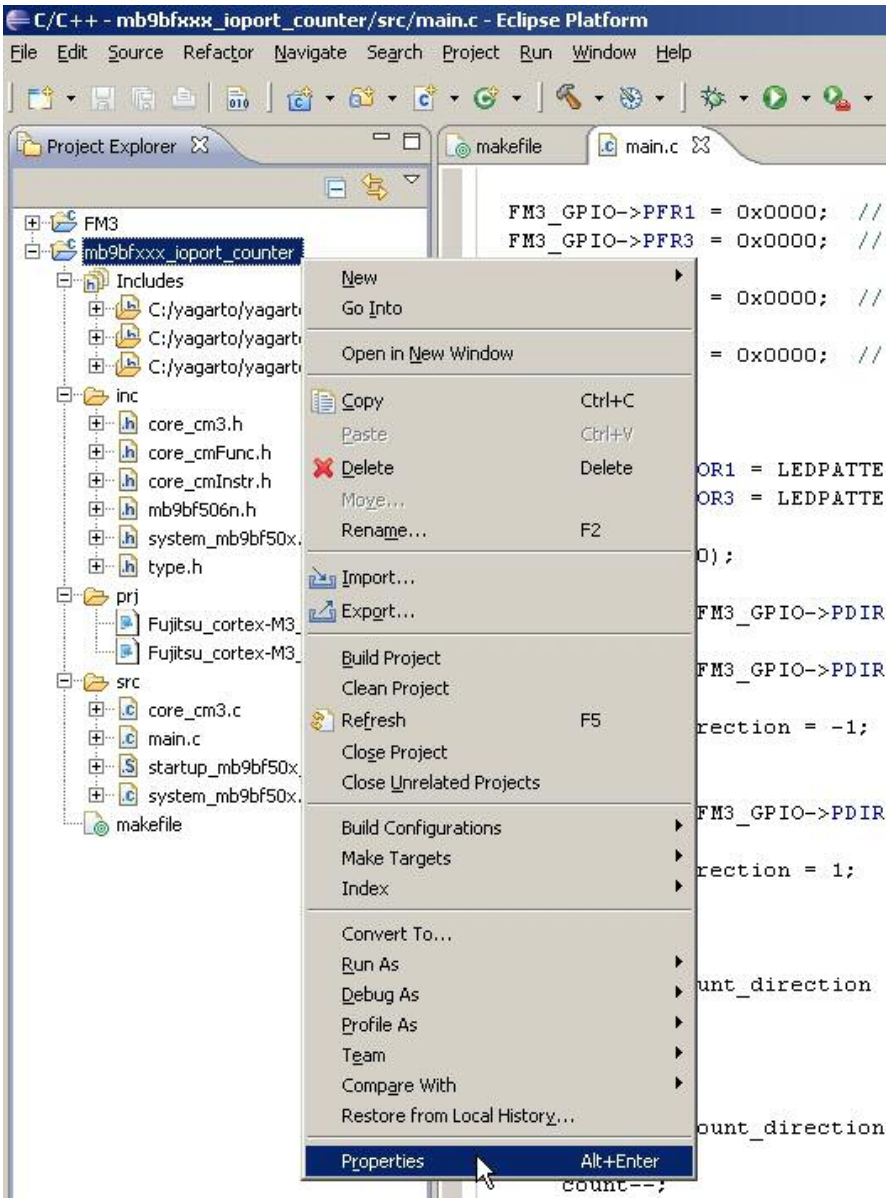
After this select the files, which should be imported, by checking them in the list.



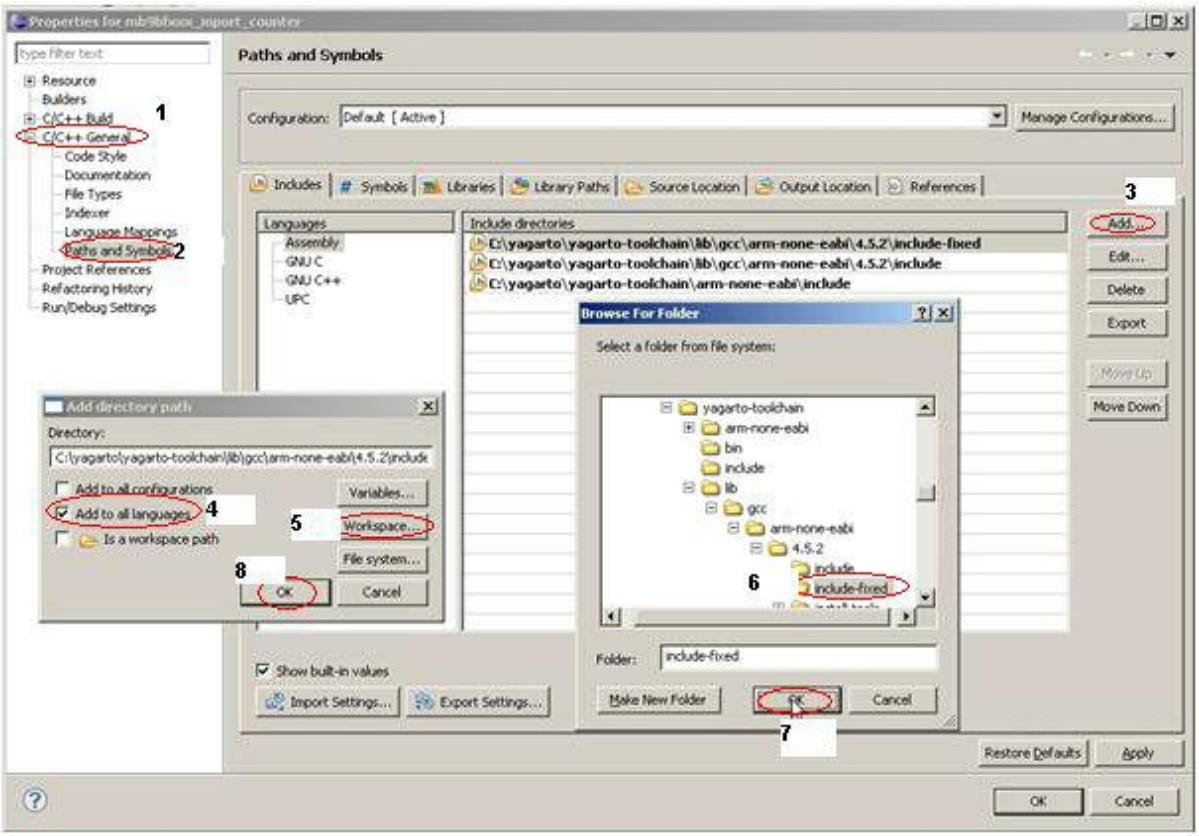
With a click to *Finish*, the selected header files are added to the folder *inc*.

9.2 Add other Libraries to the "Includes" Directory

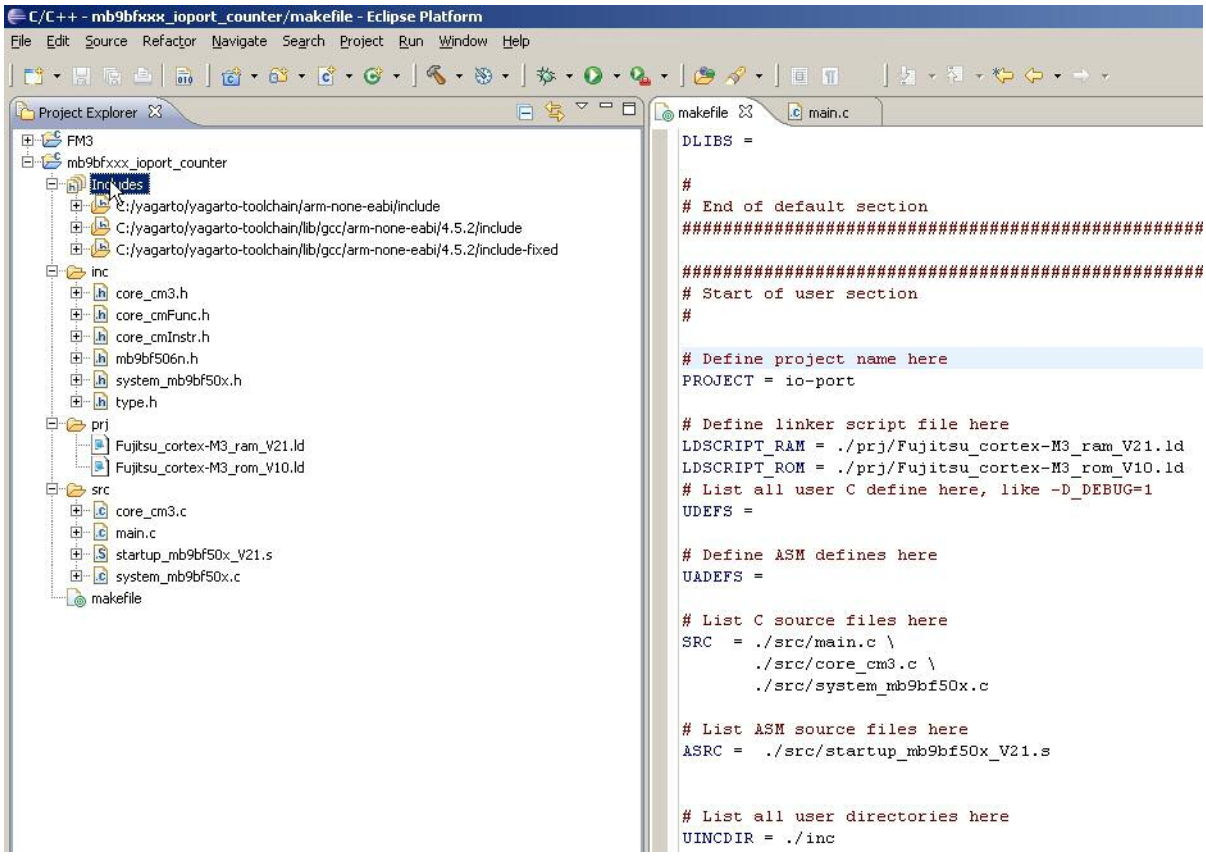
Some library headers (e.g. "*stdint.h*") must be included explicitly from the Yagarto installation directory. To set the *Includes* directory in your template or to add new libraries in this directory, select the project and click with the right mouse key to *Proprieties*. Here changes to the configuration 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 an 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 chapter 2. 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*).

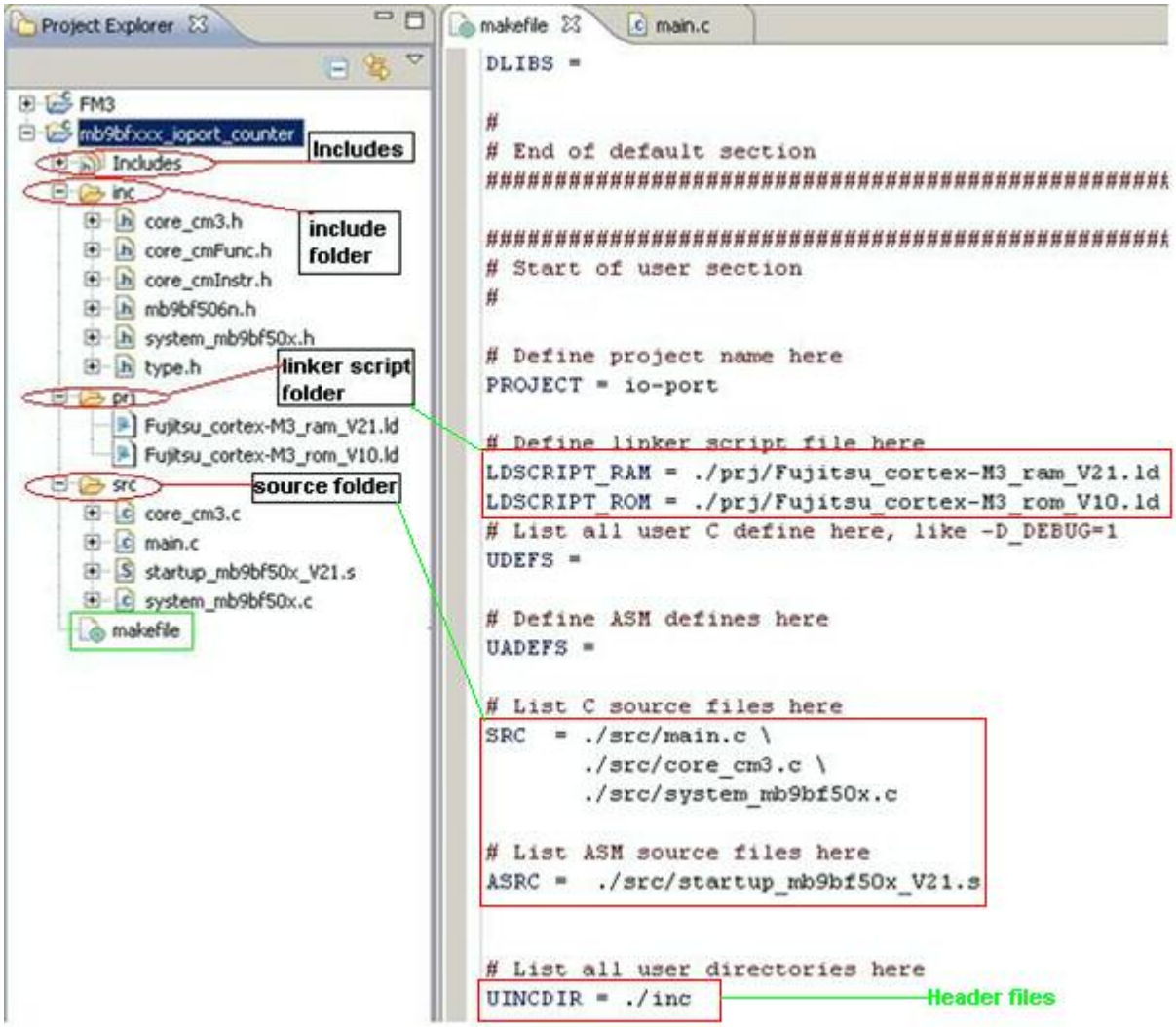
```
HEX  = $(CP) -O ihex
BIN  = $(CP) -I elf32-little -O binary
SREC = $(CP) -O srec
```

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/Fujitsu_cortex-M3_ram_V21.ld
LDSCRIPT_ROM = ./prj/Fujitsu_cortex-M3_rom_V10.ld


# List C source files here
SRC  = ./src/main.c \
       ./src/core_cm3.c \
       ./src/system_mb9bf50x.c


# List ASM source files here
ASRC = ./src/startup_mb9bf50x_V21.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.

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

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_mb9bf50x_V21.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
ROM: $(PROJECT)_rom.mhx
```

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=arme-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-noen-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/Fujitsu_cortex-M3_ram_V21.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-eabo-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 the 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-0.4.0/src

# specify OpenOCD executable
OPENOCD = $(OPENOCD_DIR)openocd.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."
```

The last part implements to target rule `progjlink`.

```
# program the FM3 internal flash memory
jflash = <HERE YOUR PATH TO SEGGER TOOLS>/JLinkARM_V442c/JFlashARM.exe
jflash_p = <HERE YOUR PATH TO SEGGER
TOOLS>/JLinkARM_V442c/Samples/JFlash/ProjectFiles/MB9xFxxx.jflash

jflash_c += -openprj$(jflash_p)
jflash_c += -open$(PROJECT)_rom.bin,0x0
jflash_c += -auto
jflash_c += -exit

# program the FM3 internal flash memory with J-FLash
progjlink:
      @echo "Flash Programming with j-link..."
      $(jflash) $(jflash_c)
      @echo "Flash Programming Finished."
```

Adjust the paths to your Segger J-Link tools installation. Note that the *JFlashARM.exe* needs a valid license!

10 Programming the Flash memory

HOW TO PROGRAM THE FLASH VIA OPEN-OCD AND/OR J-LINK

10.1 OpenOCD and Flash Programming

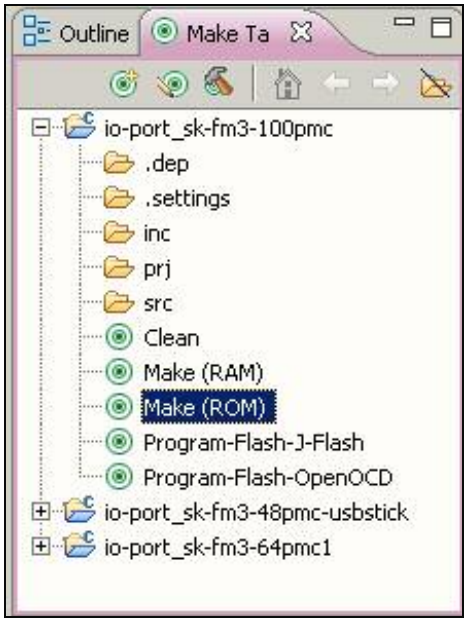
To use OpenOCD for programming the internal Flash memory, a target *Program-Flash-OpenOCD* was already created. See chapter 8.5 for usage.

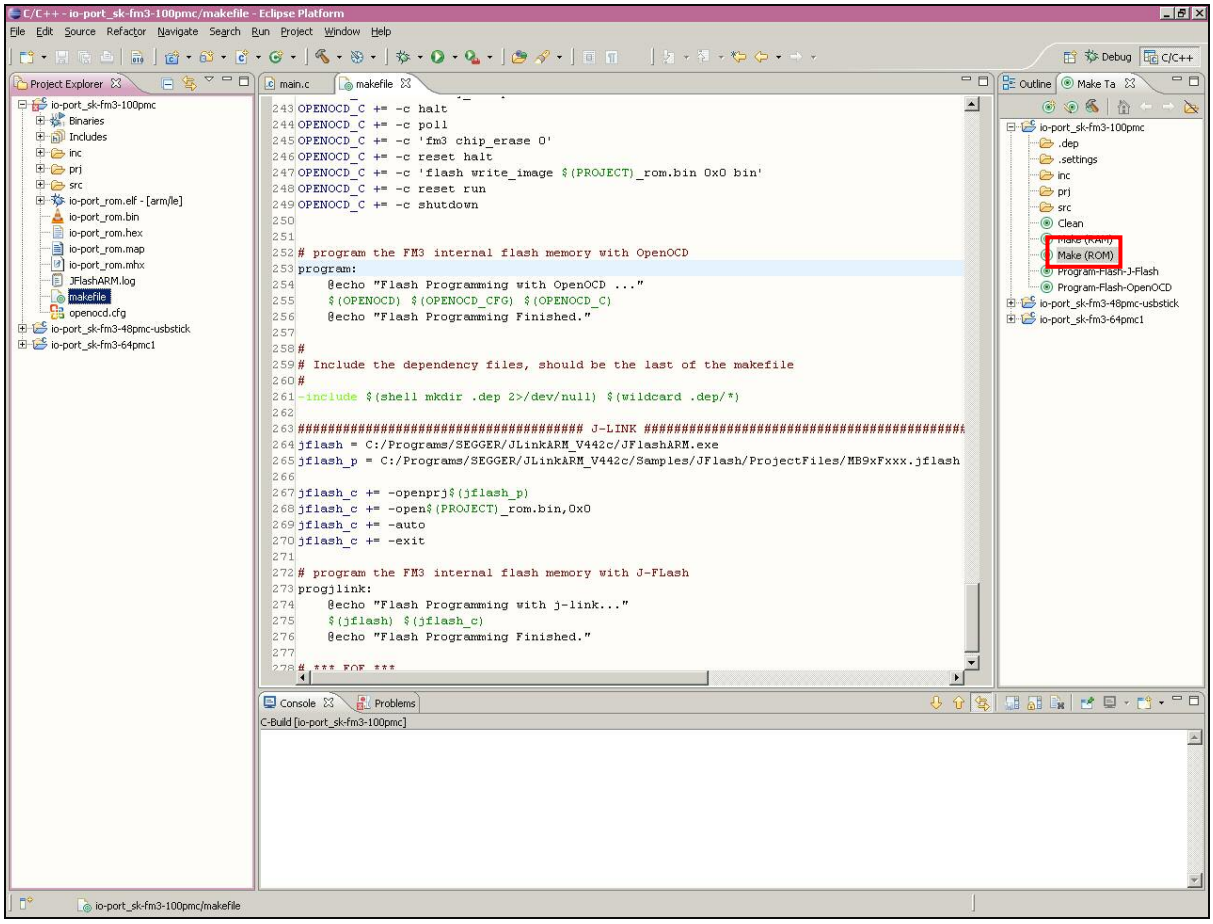
In chapter 9.3 a description of all section used in the makefile was given. The last section implemented in this makefile manages the make target *Program-Flash* used on Eclipse "C/C++ perspective" to program the internal Flash.

Connect the SK-FM3-100PMC board via JTAG interface to the USB interface of your computer. As tool for this connection use e.g. the JTAG dongle "KT-Link".

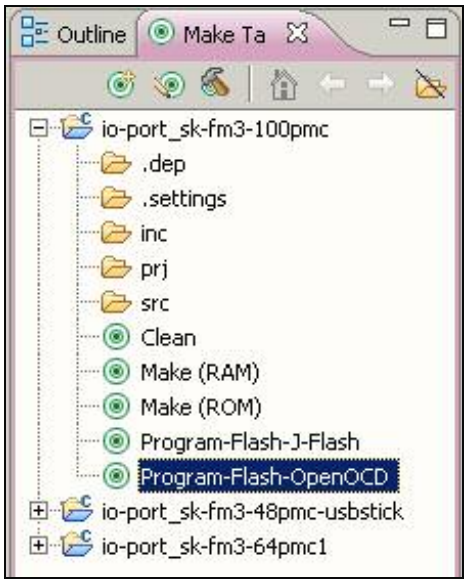
To program the internal Flash, first it is needed to build the target *Make (ROM)*. The binary file *io-port_rom.bin* will be then generated. See chapter 8.5 for usage.

Click on the target *Make (ROM)*.

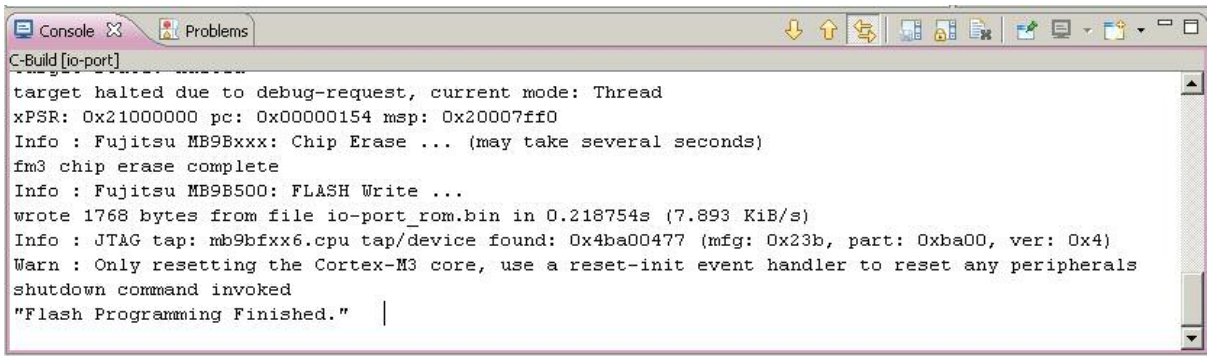




After building the project, the target *Program-Flash-OpenOCD* now can be build. For this click on the respective target to start the Flash programming with OpenOCD.



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

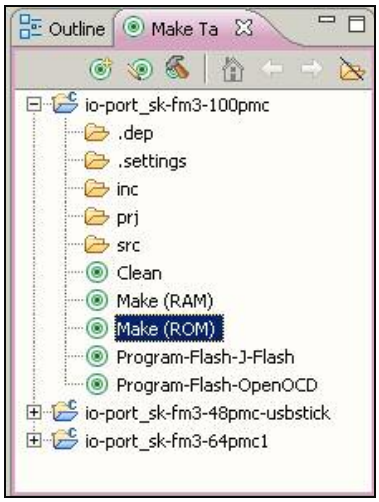


10.2 J-Link and Flash Programming

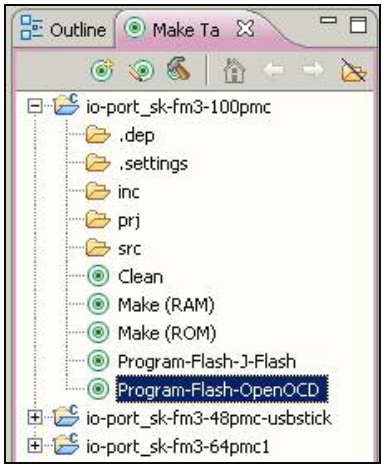
To use the J-Link for programming the internal Flash memory, a target *Program-Flash-J-Flash* was created.

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)*.



Afterwards use the target *Program-Flash-J-Flash*. Note that a valid license is needed for this.



11 Set up Eclipse External Tools

HOW TO SET UP EXTERNAL TOOLS FOR ECLIPSE

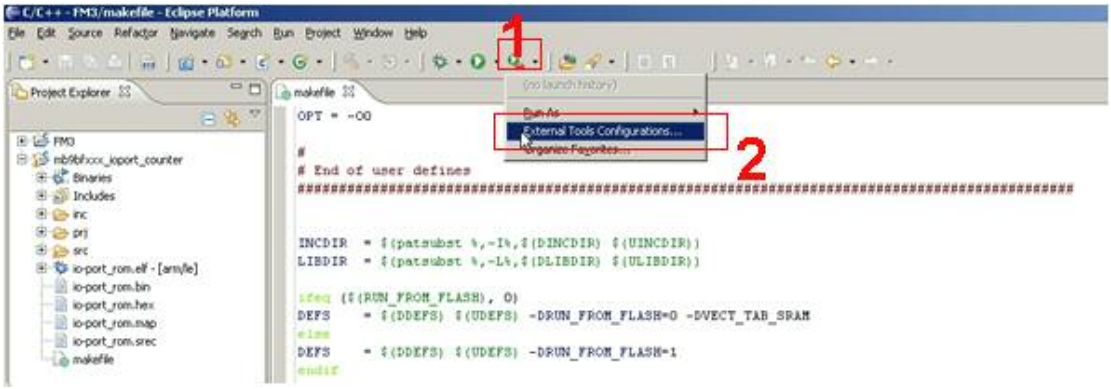
11.1 Further External Tools

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

Previously several tools were installed to the PC, such as the OpenOCD (OpenOCD configured with FTDI driver, described in chapter 3.3 or OpenOCD described in chapter 3.4) or the J-Link GDB Server (described in chapter 4).

Note that the example projects all have the external tools pre-defined in the application note's software package archieve, but the user has to adjust the configurations to his environment, i.e. his chosen installation paths, etc.

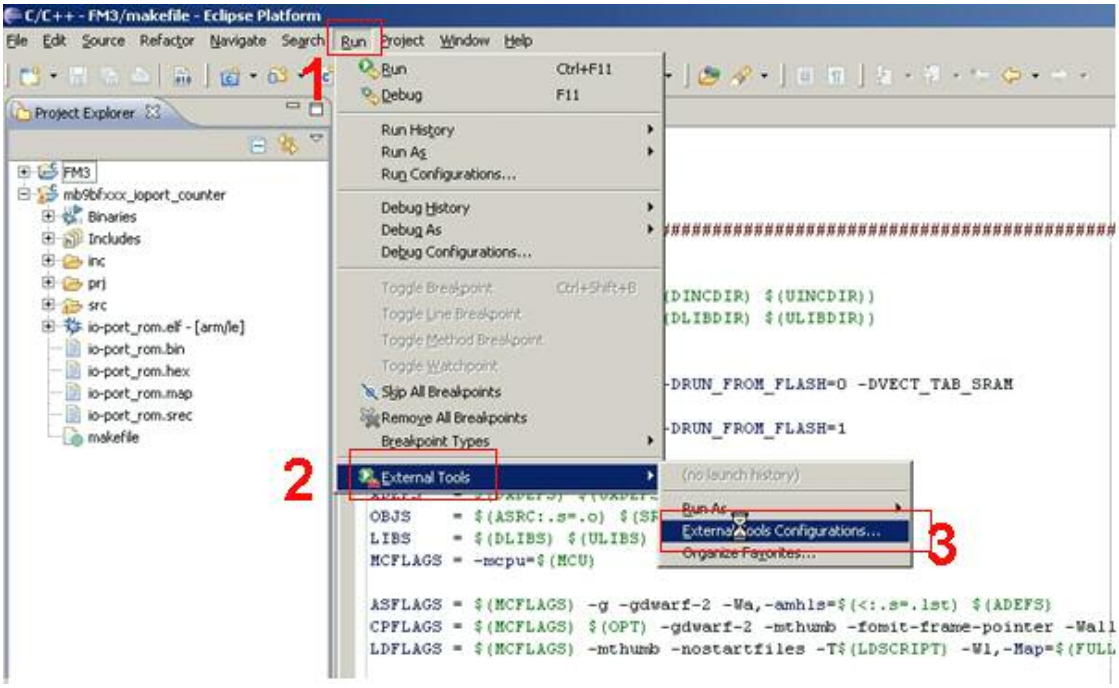
The tools installed by *External Tools Configurations..* menu can be conveniently started from the *Run* pull-down menu or via a toolbar button.



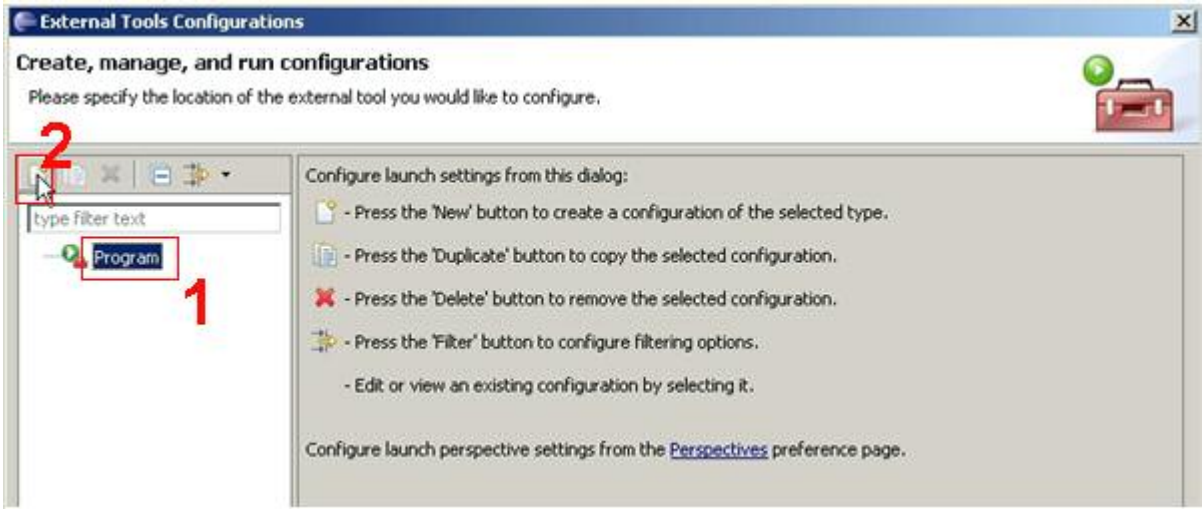
11.2 OpenOCD as an Eclipse external tool

If you have purchased a "KT-Link" JTAG interface, you can set up OpenOCD as an external tool and configure it for operation with the "KT-Link".

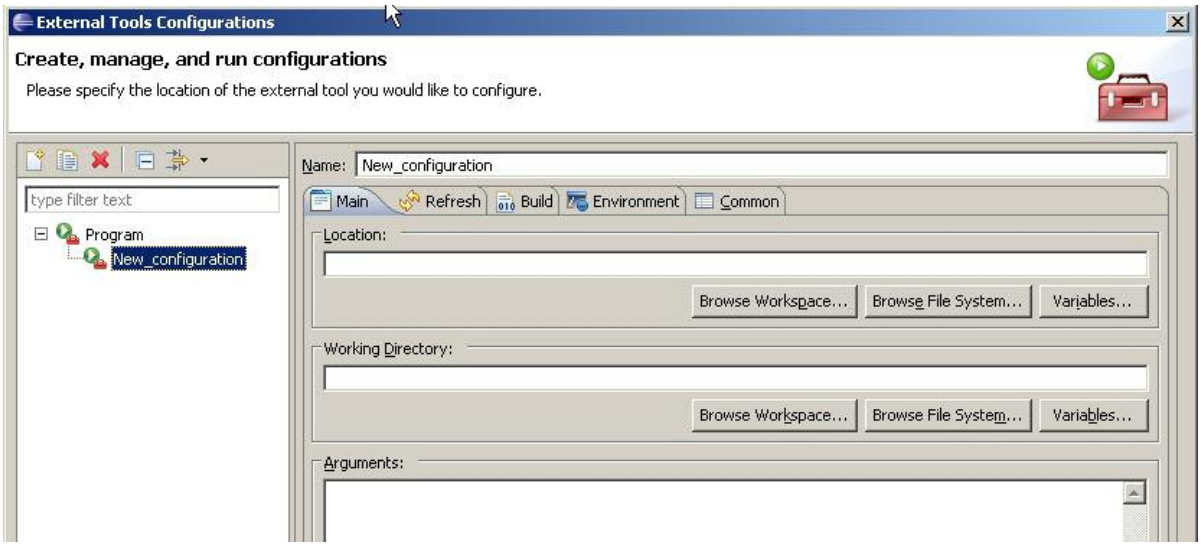
Click on *Run→External Tools→External Tools Configurations...* .



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

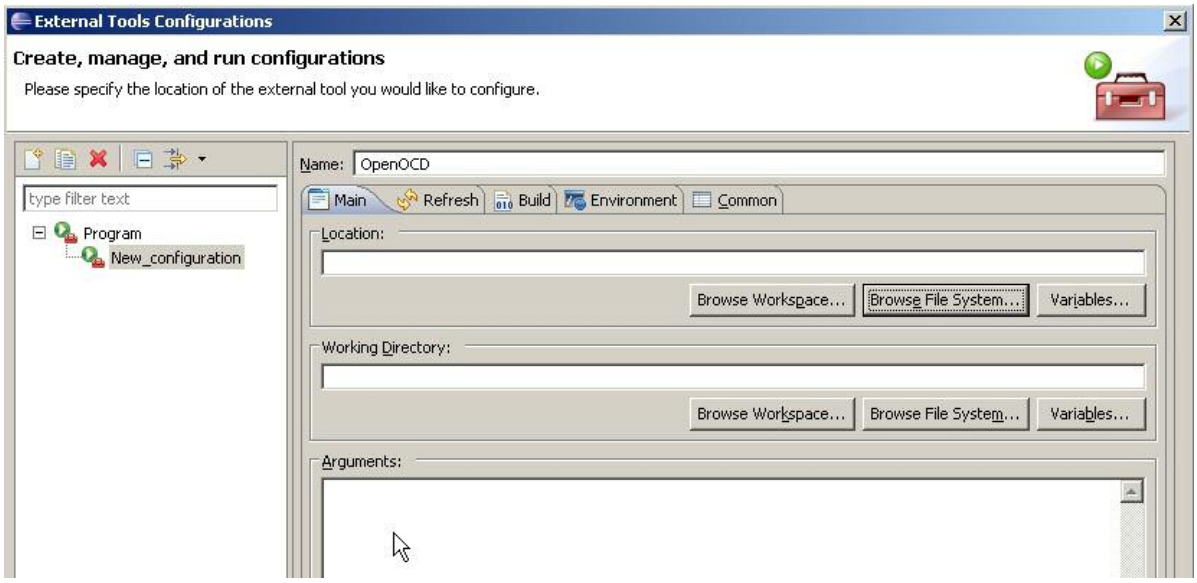


Double click *Program*.



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

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



In this application note there is the description of the installation of two versions of OpenOCD: OpenOCD configured in chapter 3.2 to use the FTDI driver device for “KT-Link” and OpenOCD installed in chapter 3.3 using the LibUSB driver for “KT-Link”.

The first OpenOCD installation was done in the folder *C:\OpenOCD_FTDI*. For this installation the OpenOCD run program *openocd.exe* can be located in:

C:\OpenOCD_FTDI\openocd-0.4.0\src\openocd.exe

The second OpenOCD installation was done in the folder *C:\OpenOCD_LibUSB*. For this installation the OpenOCD run program *openocd.exe* can be located in:

C:\OpenOCD_LibUSB\bin\openocd.exe

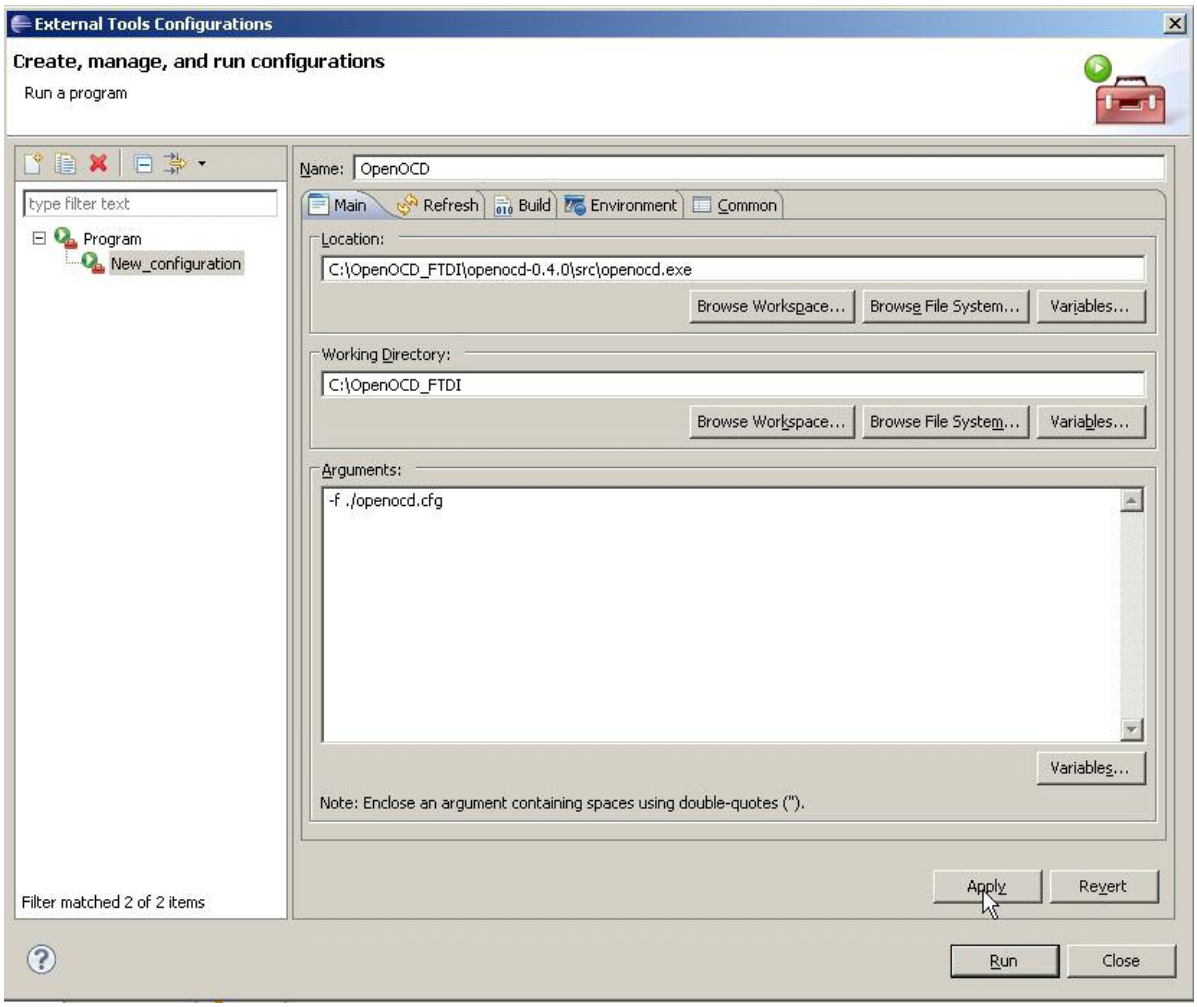
For the first installation use the FTDI driver for “KT-Link” (chapter 3.2.1) and for the second procedure use the LibUSB driver for “KT-Link” (chapter 3.3.2).

In this section the OpenOCD_FTDI version of OpenOCD as eclipse external tool is described. In the “Name” text box call this external tool “OpenOCD”

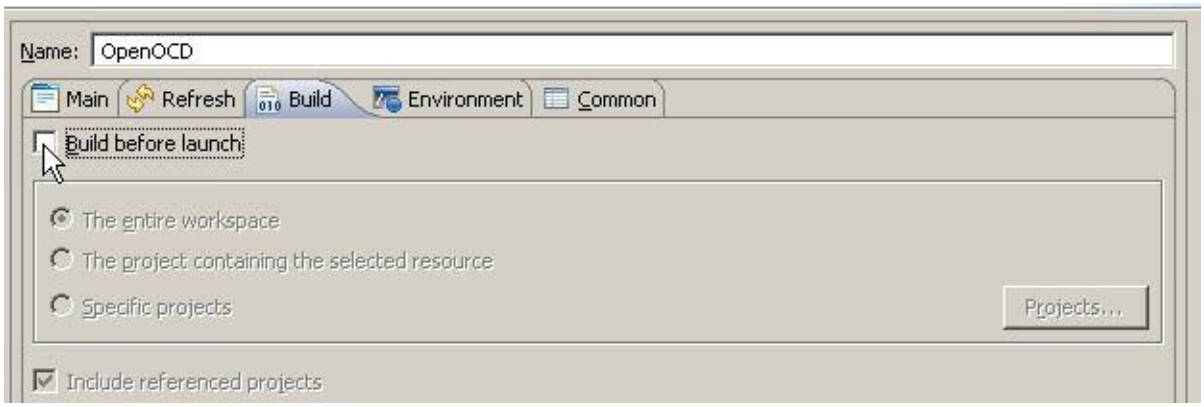
In the “Location:” pane, use the *Browse File System…* button to search for the OpenOCD executable. It is located in the folder: *C:\OpenOCD_FTDI\openocd-0.4.0\src\openocd.exe*

In the “Working Directory” pane, use the *Browse File System…* button to specify *C:\OpenOCD_FTDI* as the working directory.

In the “Arguments” pane, enter the argument “-f <*your project path*>\openocd.cfg” to specify the OpenOCD configuration file designed for the “KT-Link”.



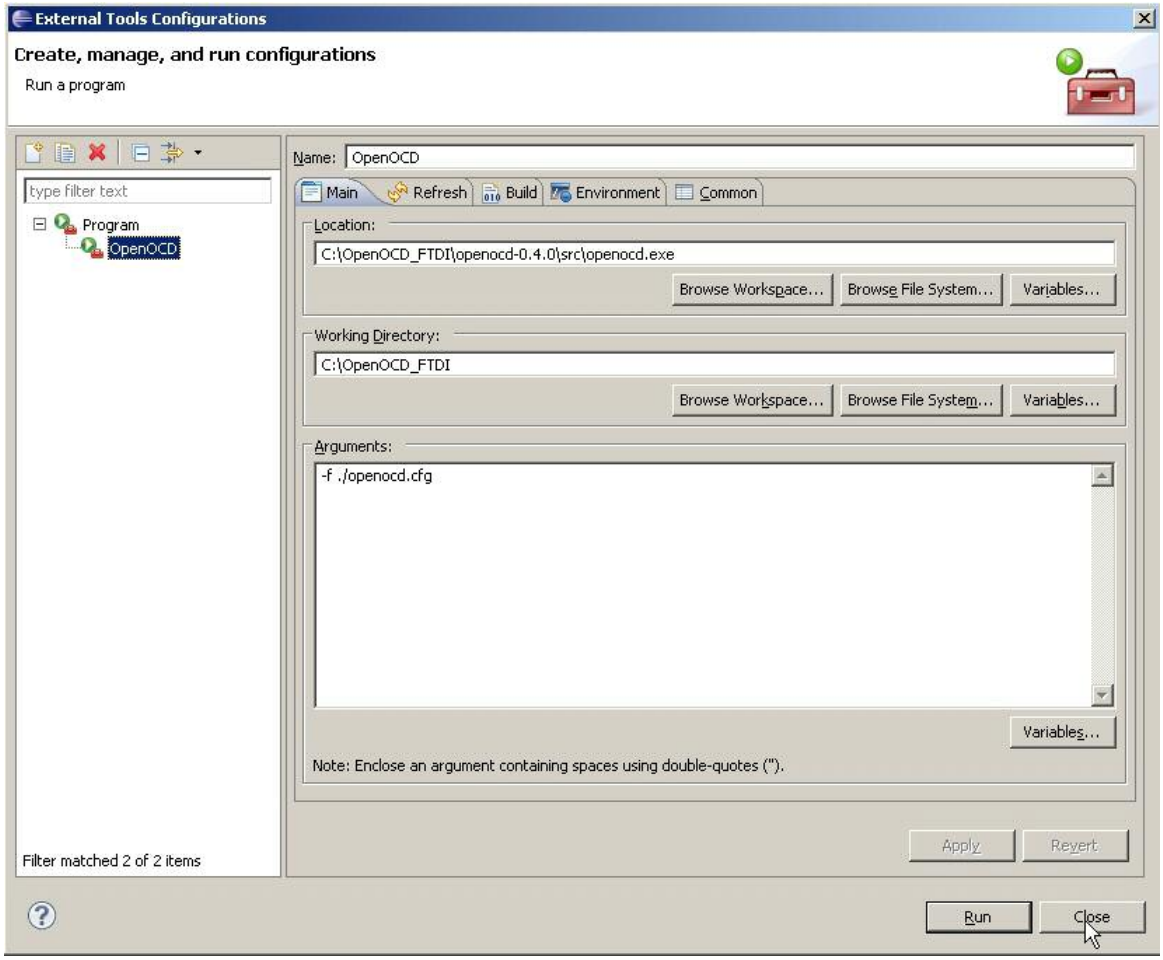
In the *Build* tab uncheck *Build before launch*.



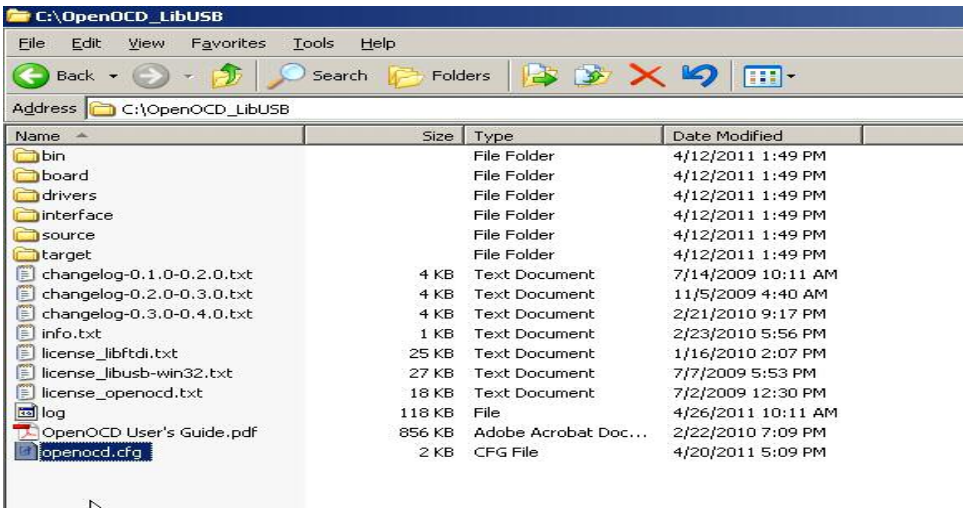
No changes are required to the other tabs in the other forms (*Refresh*, *Environment*, and *Common*).

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*.



In the same way set up *OpenOCD_LibUSB* as an Eclipse external tool. The driver for "KT-Link" must be previously changed from FTDI to LibUSB (see chapter 3.4.2). The OpenOCD configuration file *openocd.cfg* for "KT-Link" is the same for both drivers (FTDI or LibUSB). This configuration file must also be copied to the *OpenOCD_LibUSB* installation folder: *C:\OpenOCD_LibUSB*

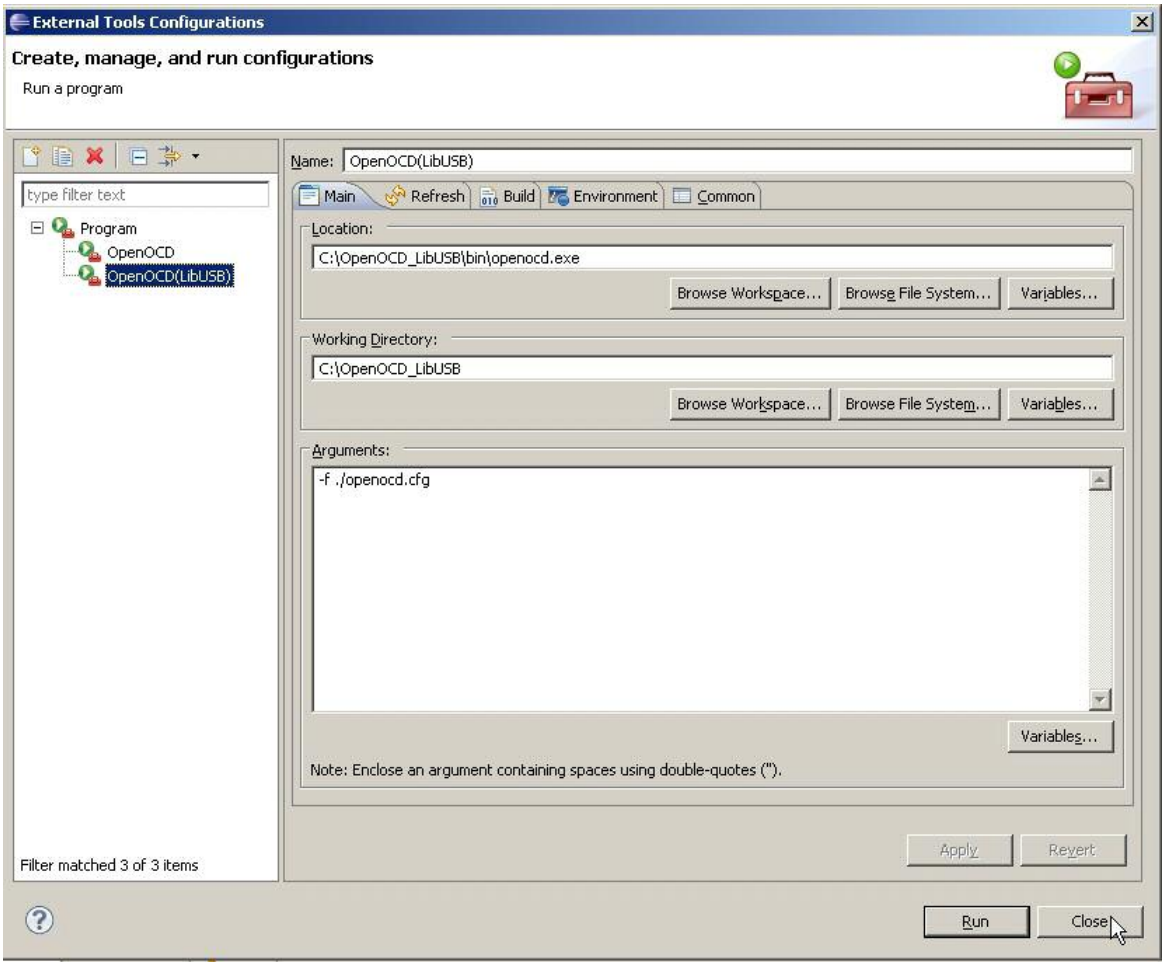


In the "Location:" pane, use the *Browse File System…* button to search for the OpenOCD executable. It is located in the folder: *C:\OpenOCD_LibUSB\bin\openocd.exe*

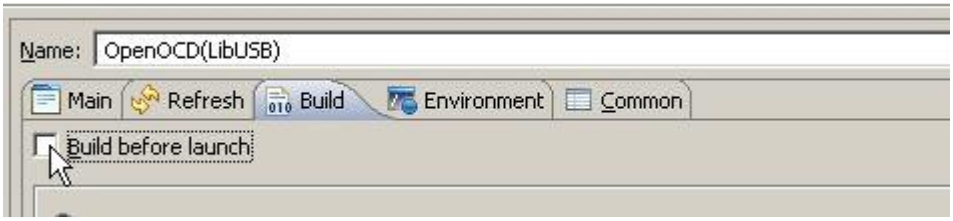
In the "Name" text box call this external tool "OpenOCD(LibUSB)"

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 designed for the “KT-Link”.



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



The set up of OpenOCD as an Eclipse external tool is done now. Note, that OpenOCD runs as a daemon, that means, that a program runs in the background waiting for commands to be submitted to it.

11.3 J-Link GDB Server as an Eclipse External Tool

The software delivered with the Segger JTAG interface “J-Link” can also be used as an Eclipse external tool. The software installation was explained in chapter 4.

In this chapter the “JLinkARM_V425k” version of JLink-software is used. This version supports the Flash download for the FM3 MCUs. Note that older versions do not support FM3.

The J-Link software installation was explained in chapter 4.1. For the installation folder *C:\Program Files\SEGGER\JLinkARM_V425k* was chosen. The full path to the J-Link GDB Server program is then:

C:\Program Files\SEGGER\JLinkARM_V425k\ JLinkGDBServer.exe

To set the J-Link GDB Server as an Eclipse external tool, choose *Run→External Tools→External Tools Configurations...* .

The “External Tools” window will appear. Click on *Program* and then *New* button to set a new External Tool.

In the “Name” text box name this external tool e.g. “JLink-GDB-Server”.

In the “Location:” pane use the *Browse File System…* button to search for the J-Link GDB server executable. It is located in the folder:

C:\Program Files\SEGGER\JLinkARM_V425k\ JLinkGDBServer.exe.

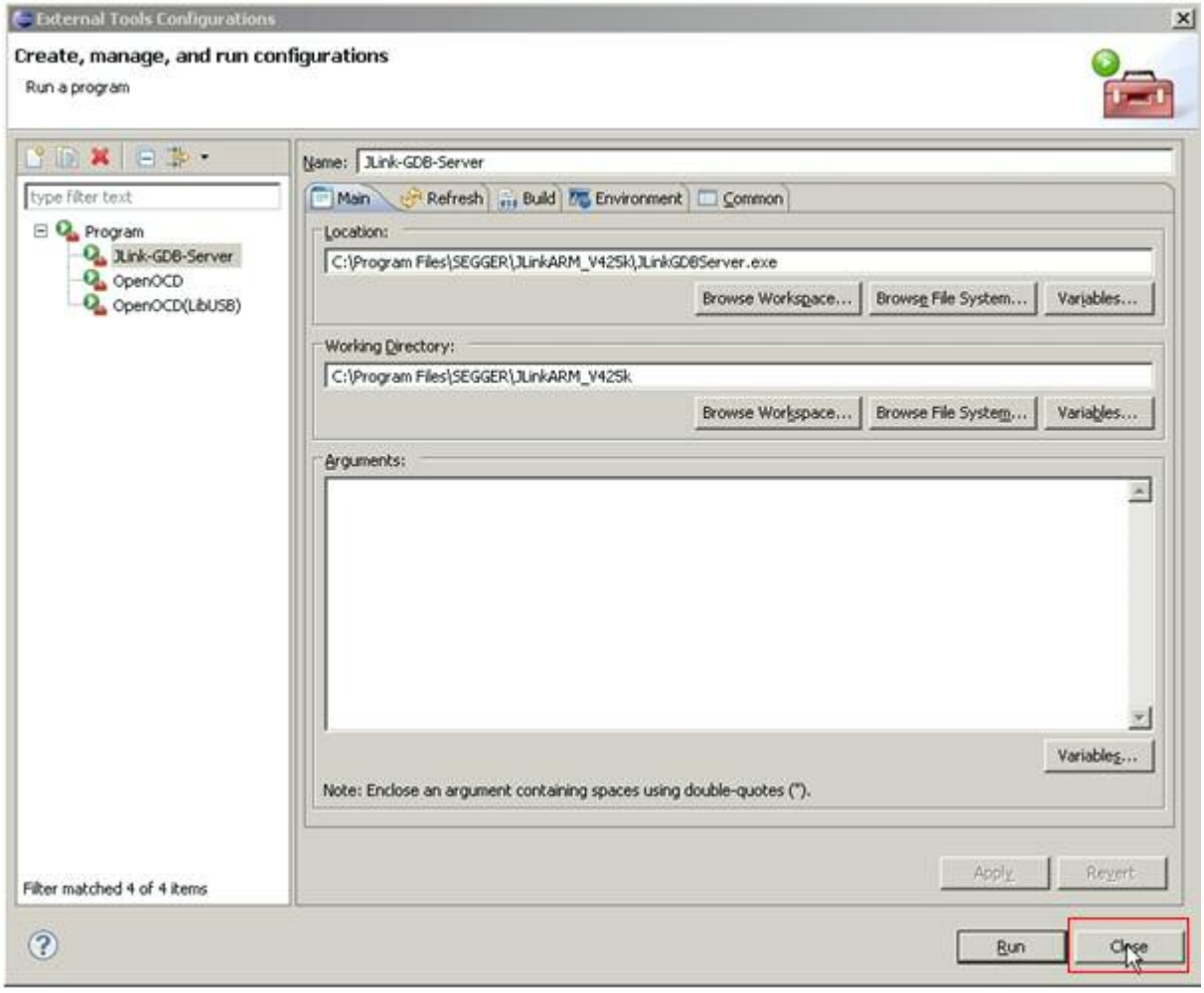
In the “Working Directory” pane, use the *Browse File System…* button to specify *C:\Program Files\SEGGER\JLinkARM_V425k* as the working directory.

J-Link GDB server is started without any arguments.

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

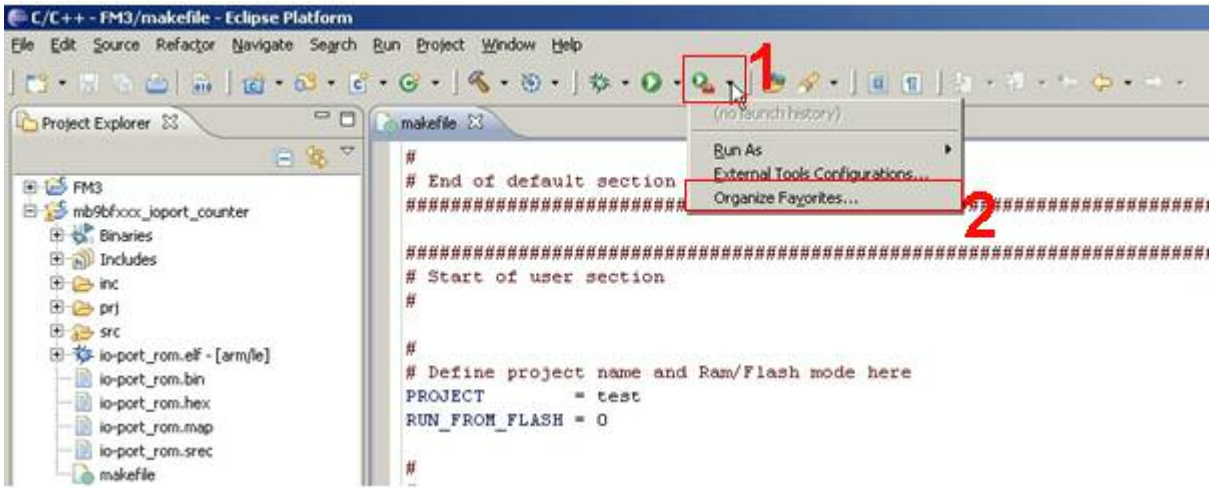
No changes are required to the other tabs (*Refresh*, *Environment*, and *Common*).

Click on *Apply* and *Close* to register J-Link GDB server as eclipse external tool.



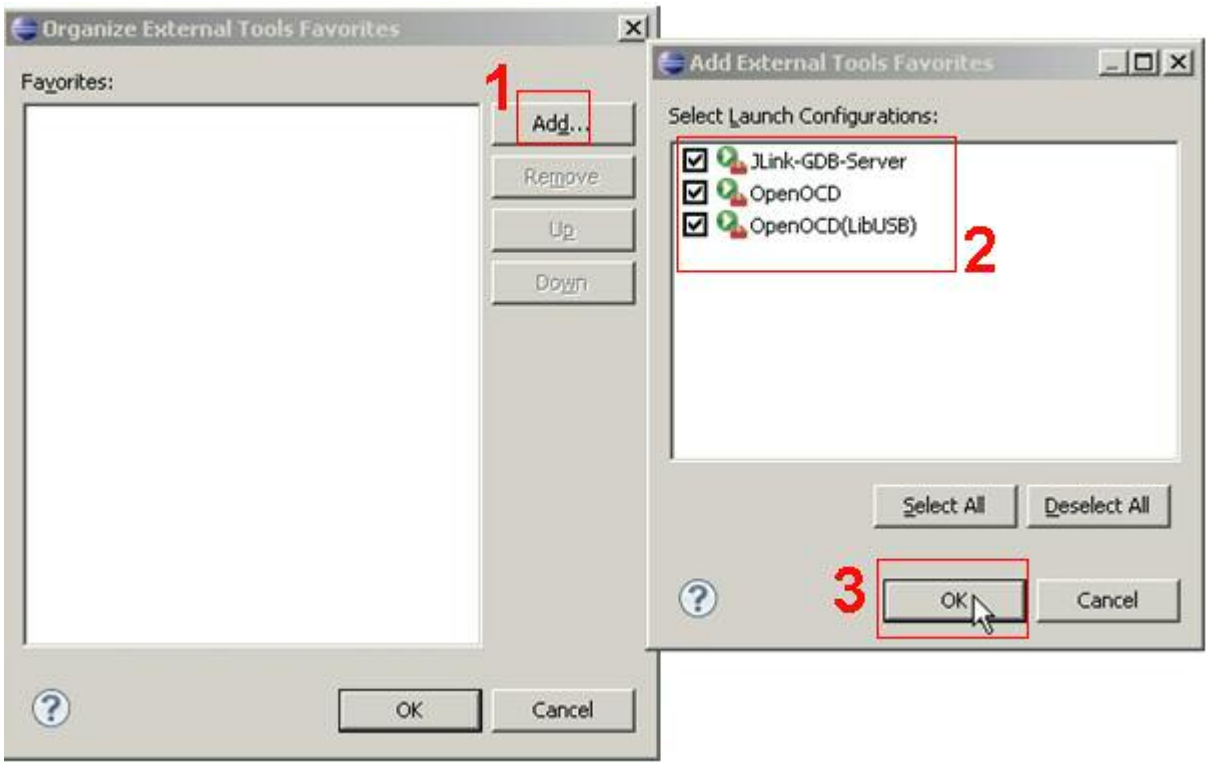
Now organize all external tools needed for debugging.

From the bar menu select the following configuration window:

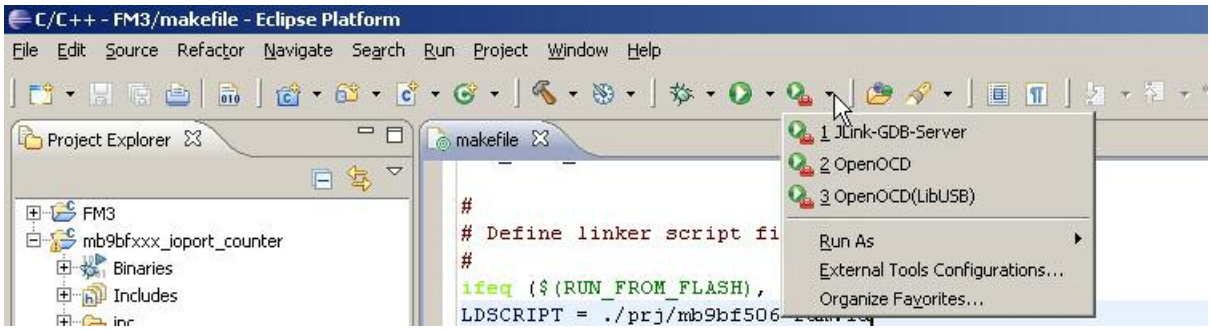


Click on *Organize Favourites.....*

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.



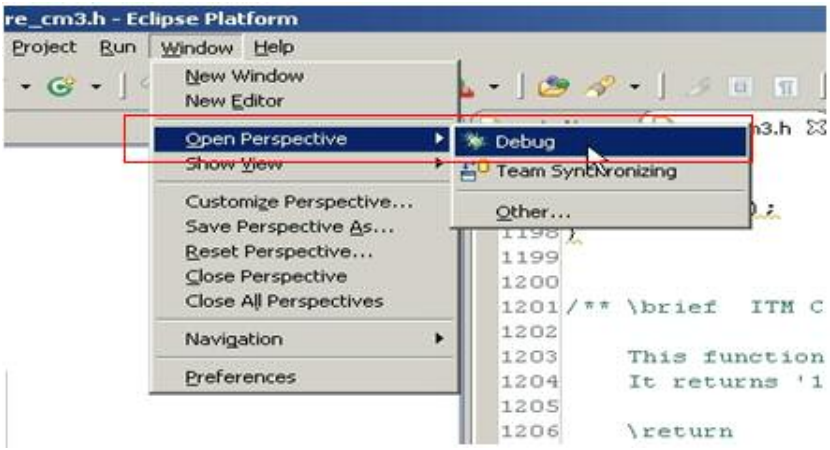
12 Eclipse CDT Debug Perspective

HOW TO USE THE DEBUG PERSPECTIVE

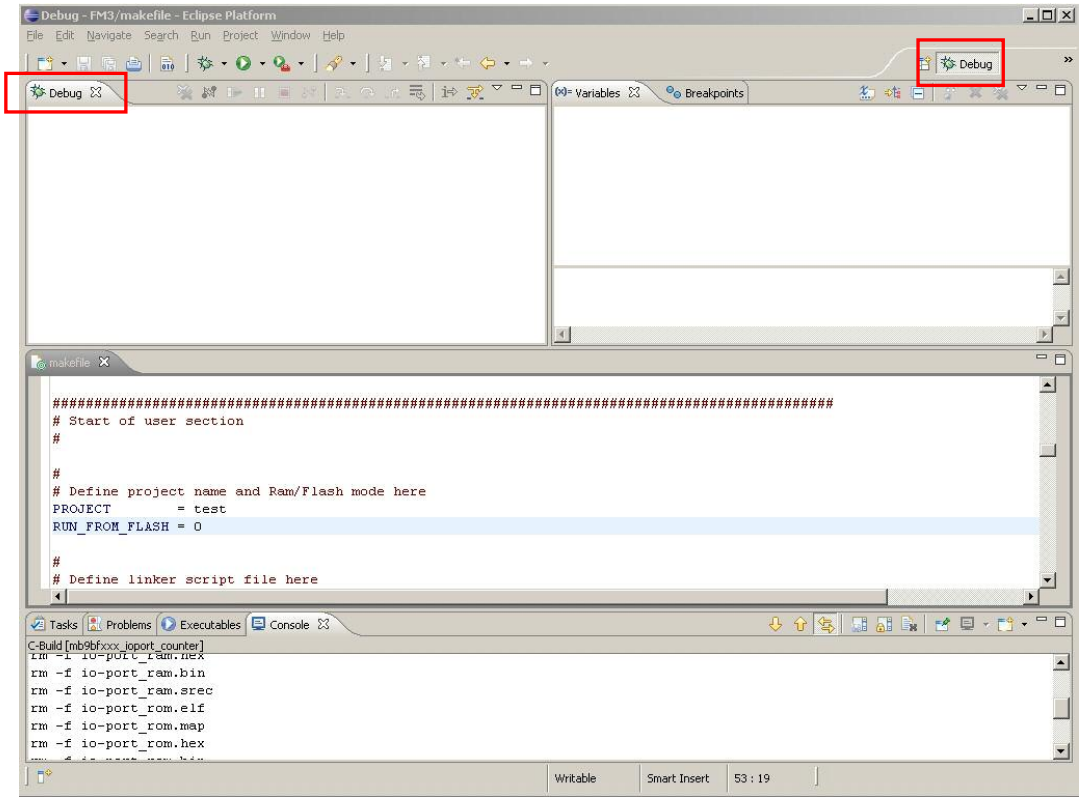
In chapter 8 a sample FM3 project was created and the build process to create all application output files (**.bin*, **.mhx* or **.hex*) needed to program the Flash was explained. These output files include also debug information files (**.elf*) needed for debugging program code in Flash or RAM.

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.



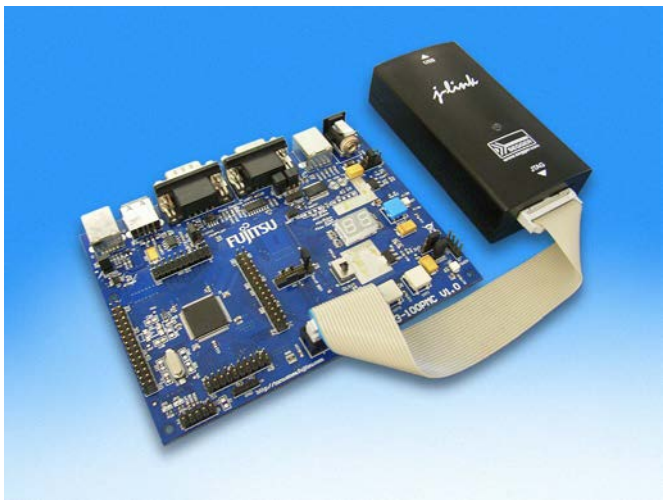
The next steps explain, how to use “J-Link GDB Server” (already added in the last chapter as an external tool) to download and debug the application on the Flash memory.

12.1 Programming and debugging on the Flash memory

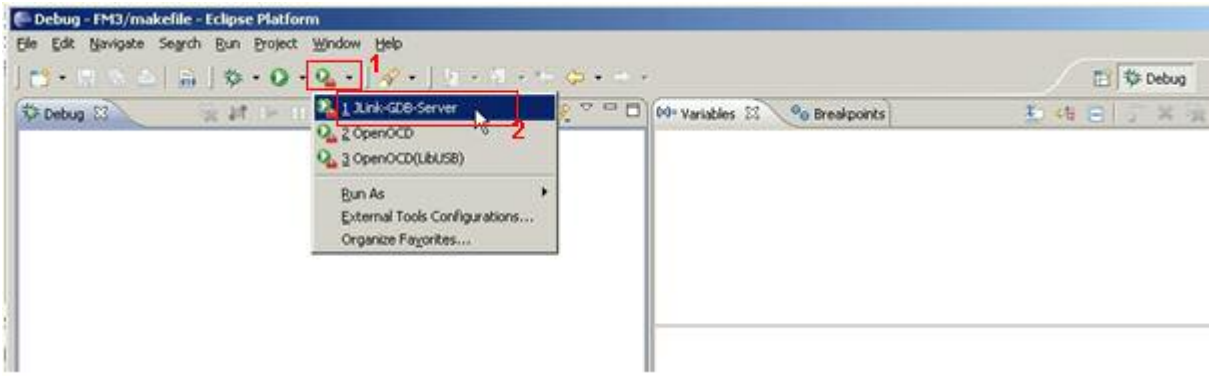
Now the debug process using “J-Link GDB Server” and OpenOCD is explained. The application created automatically by building the project is designed for debugging on the Flash.

12.1.1 Using J-Link GDB Server to download and debug the flash application

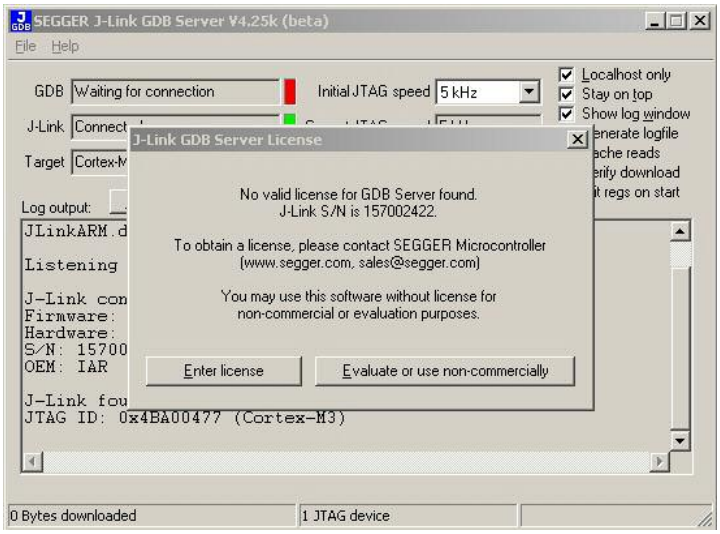
Connect the SK-FM3-100PMC board via the JTAG interface “J-Link” to the USB interface of the host PC.



When this is done, start the “J-Link GDB Server” by clicking on “J-Link GDB Server” and the external tool will be then started as shown below.

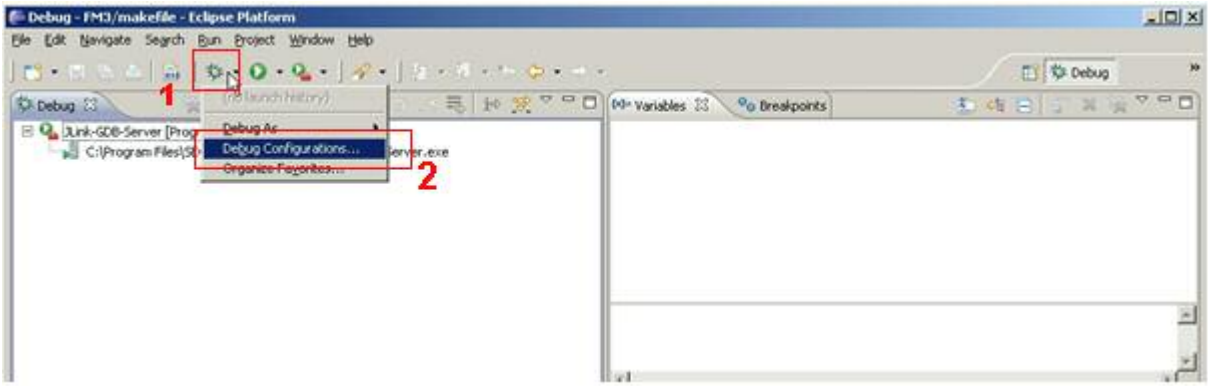


The GDB server requires a license before starting. See chapter 4.3 for terms of usage.

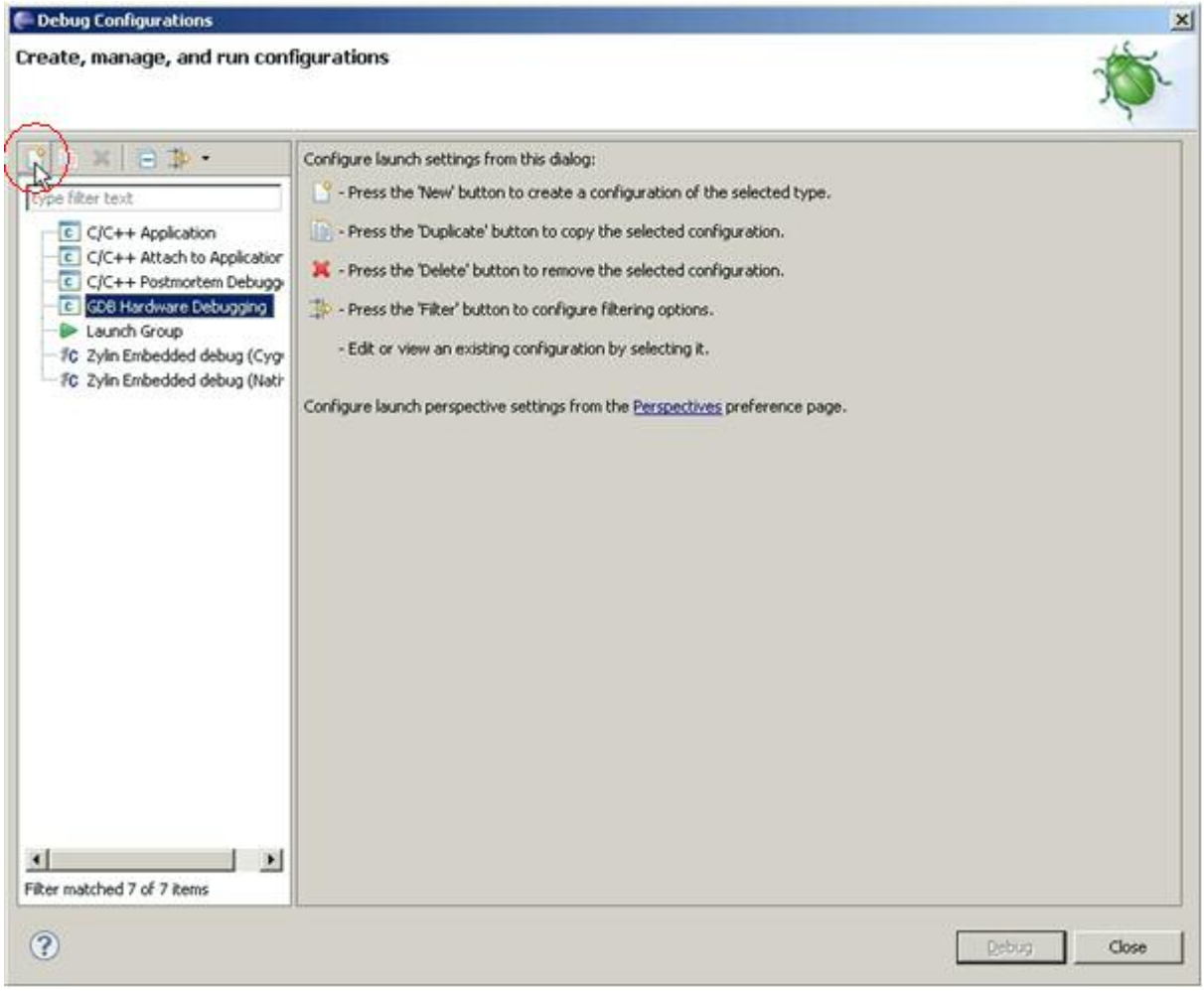


After confirming this step the “J-Link GDB Server” starts and waits for instruction from the “GDB Debugger”. We will now create a new “Debug Configuration”.

For this reason click on *Debug Configurations...* as shown in the following screen shot.

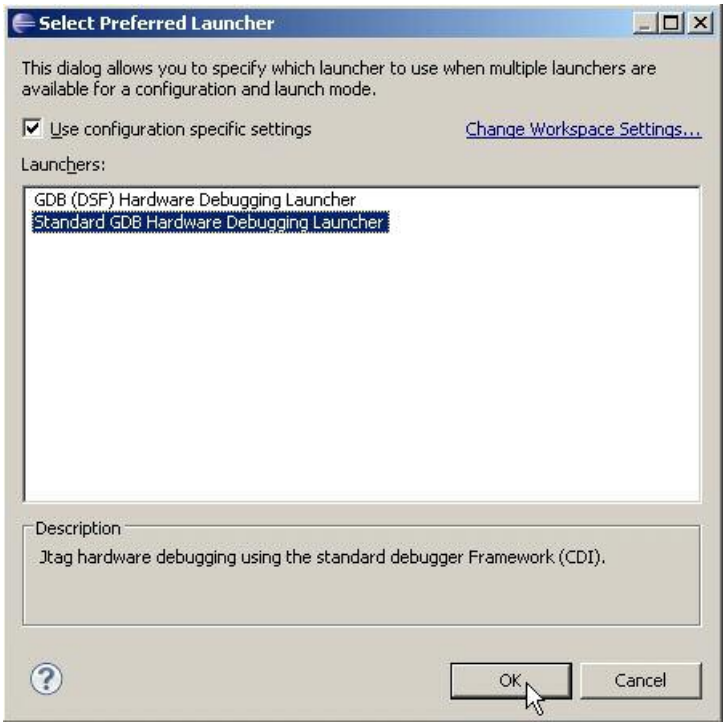


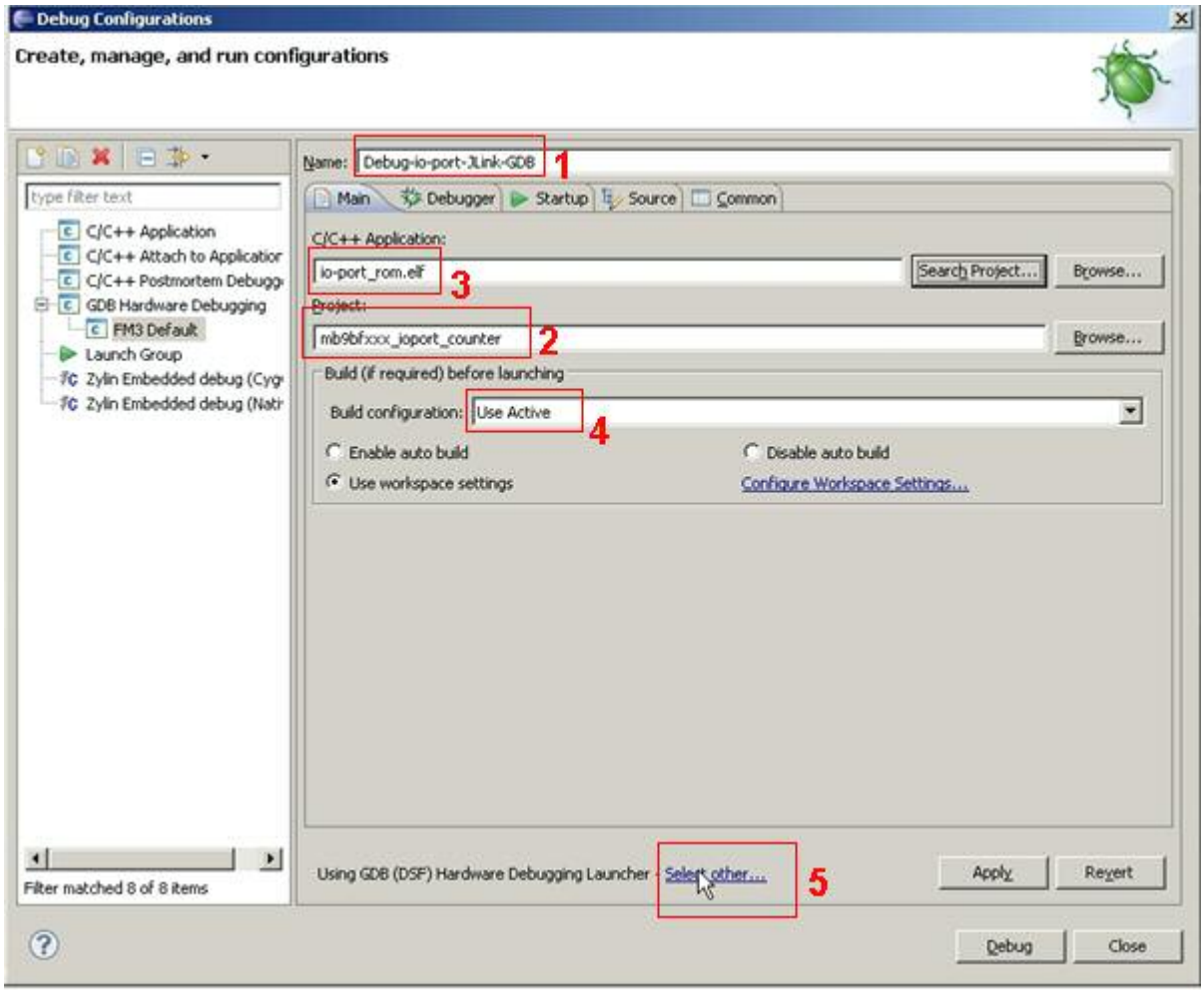
The debug configuration is shown below:



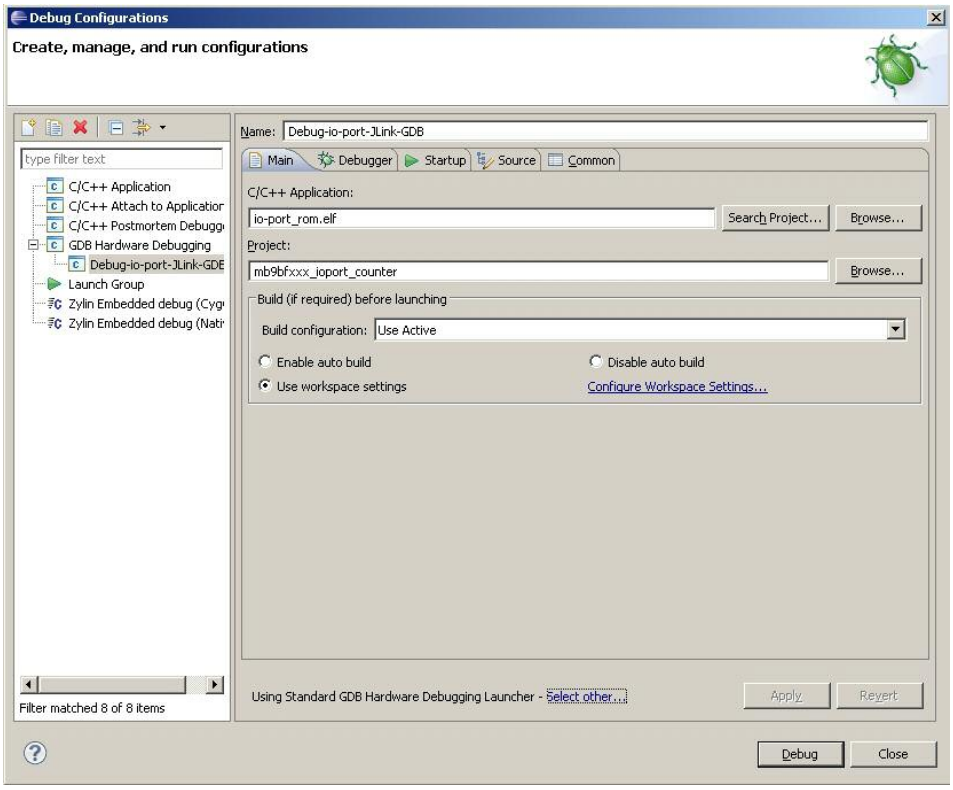
To create a new debug configuration, select “GDB Hardware Debugging” and click on *New*.

1. Rename the debug configuration. To avoid confusion with other debug configurations (using OpenOCD), it is recommended that the selected name a reference to the project name (io-port) and to the used external tool (e. g. “JLink-GDB” for J-Link GDB Server).
2. In the “Project” text box, use the *Browse* button to find the project *mb9bfxxx_ioport_counter*.
3. In the “C/C++ Application” text box, use the *Search Project…* button to find the application file *io-port_rom.elf*.
4. Set the “Build configuration” text box to *Use Active*.
5. Click on *Select other....* by “Using GDB (DSF) Hardware Debugging launcher” as shown below and select “Standart GDB Hardware Debugging launcher”. Click on *OK* finally.



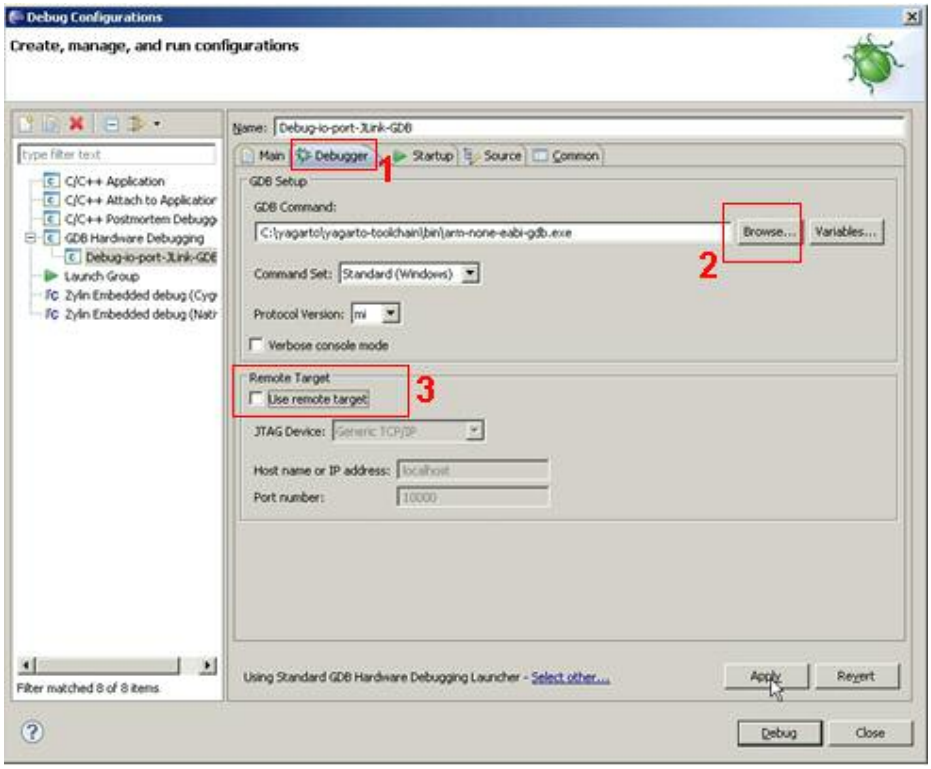


the change are now applied. The configuration window now shows:



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 in: *C:\yagarto\yagarto-toolchain\bin*.

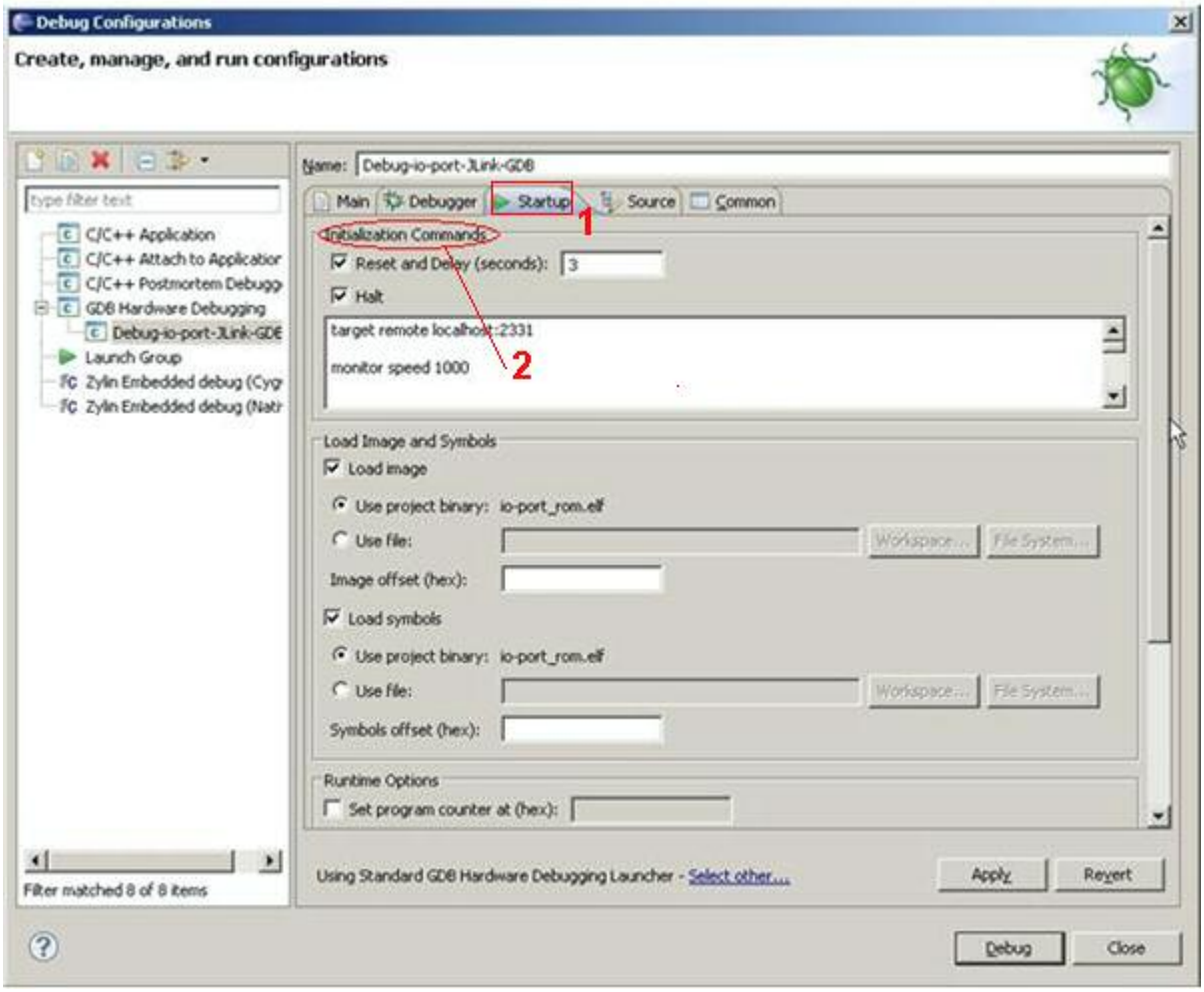
Uncheck *Use remote target*.



Now select the “Startup” tab as shown below. On the “Initialization Commands” panel type in or copy the following lines:

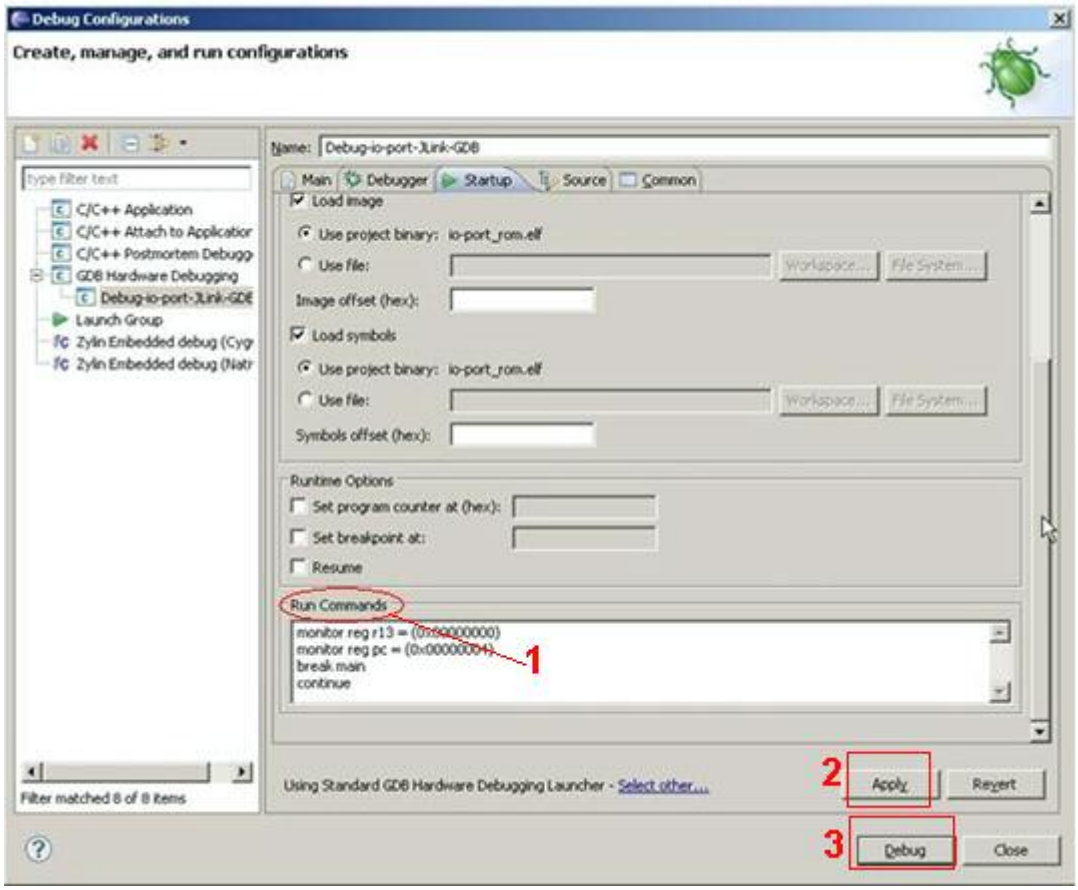
```
target remote localhost:2331
monitor speed 1000

monitor flash device = MB9BF506N
monitor flash download = 1

# Set gdb server to little endian
monitor endian little

# Set JTAG speed to 30 kHz
monitor speed 30

# Reset the chip to get to a known state.
monitor reset
monitor sleep 10

# Set JTAG speed in khz
monitor speed auto
load
monitor sleep 100

# Reset the chip to get to a known state.
monitor reset
monitor sleep 10
```

:

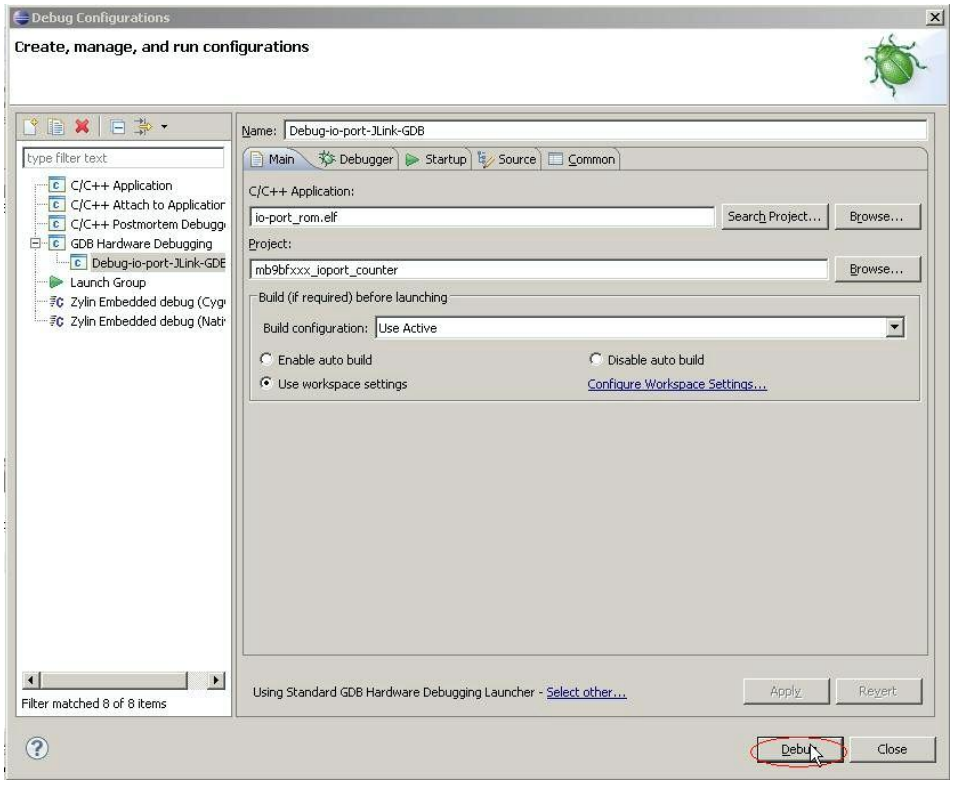


Scroll down in the “Startup” tab to locate the “Run Commands” panel and add the following lines

```
monitor reg r13 = (0x00000000)
monitor reg pc = (0x00000004)

break main
continue
```

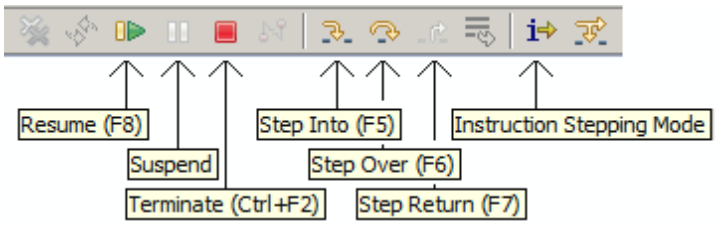


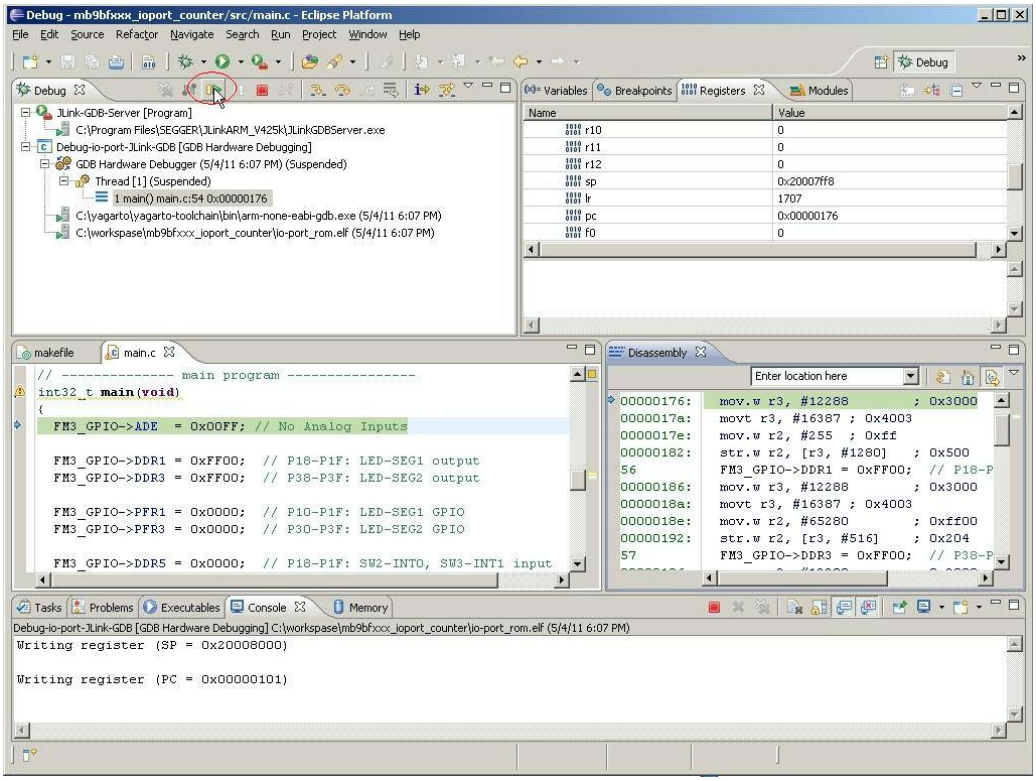
The rest of the configuration window can be left in its default settings. Click on *Apply* to confirm the debug configuration. The debug process can be started by clicking on the *Debug* button.



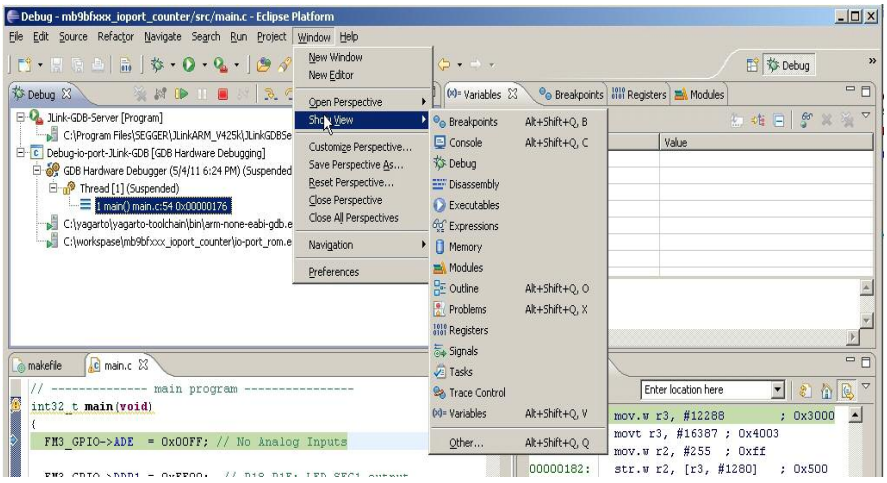
Note that the Flash erase and download will take some seconds.

The next figure shows that the application was downloaded to the Flash memory and the debug process was successfully started. To resume, simply click on the *Resume* button.





For other views, go to the menu *Window* and under *Show View* all views supported by Eclipse CDT are listed. Click on the respective view to get it.



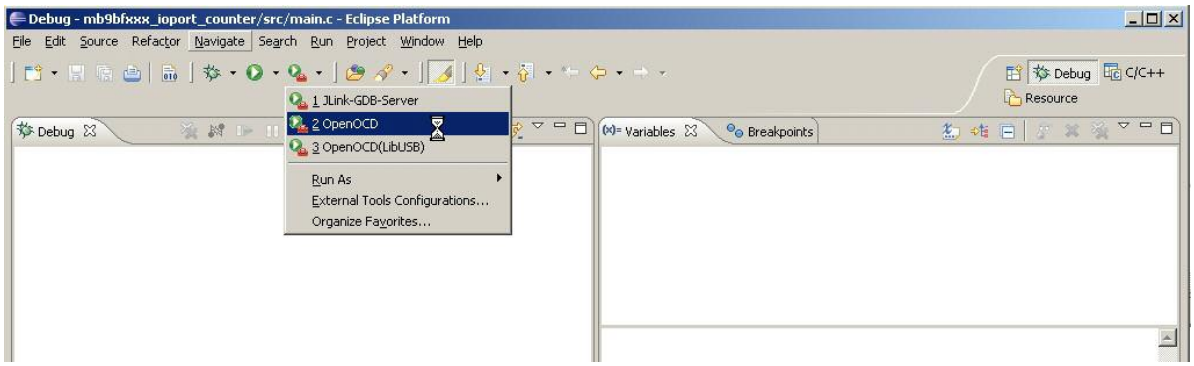
The “GNU GDB debugger” is fully integrated to the Eclipse IDE to perform IDE debugging with breakpoints, single stepping and sophisticated inspection of variables and data structures.

12.1.2 Using the OpenOCD Server to debug a Flash Application

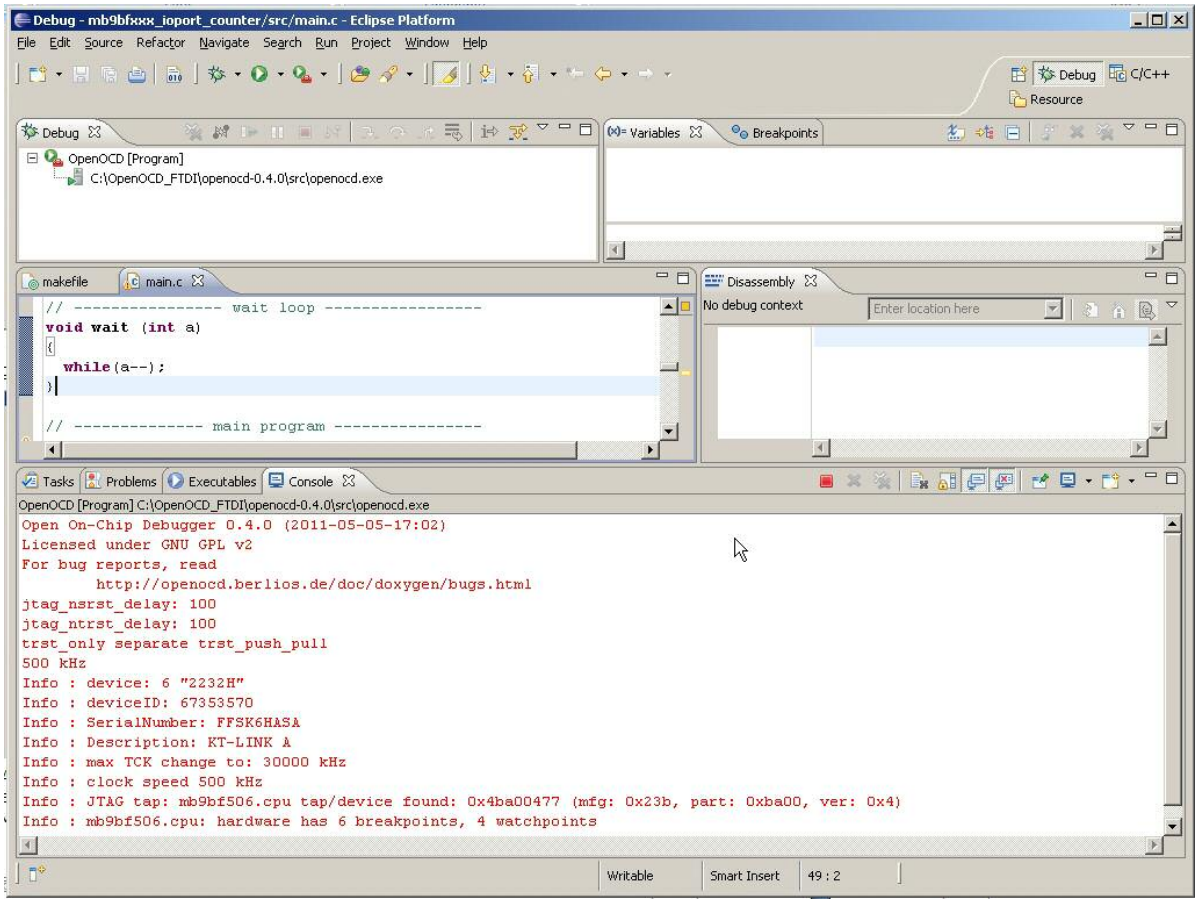
Connect the SK-FM3-100PMC board via JTAG interface to the USB interface of your computer. As the interface tool for this connection use e.g. the JTAG dongle “KT-Link”.

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.

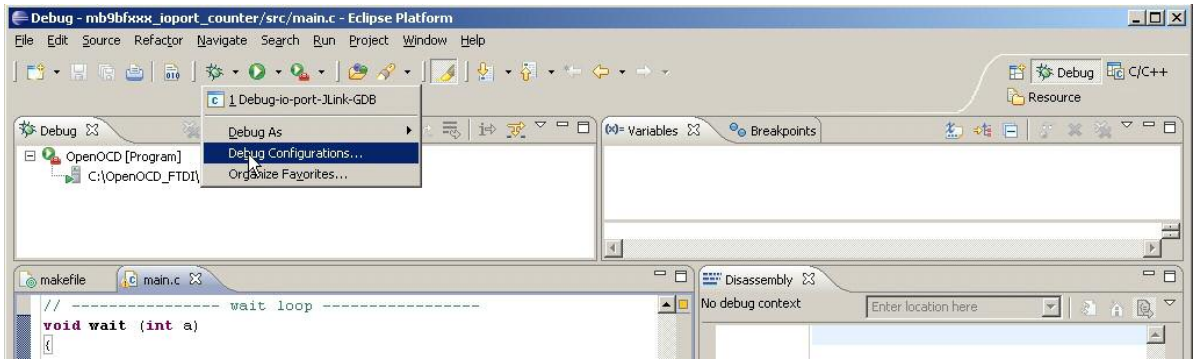
Click on *OpenOCD* and the external tool will be started as shown below.



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



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.

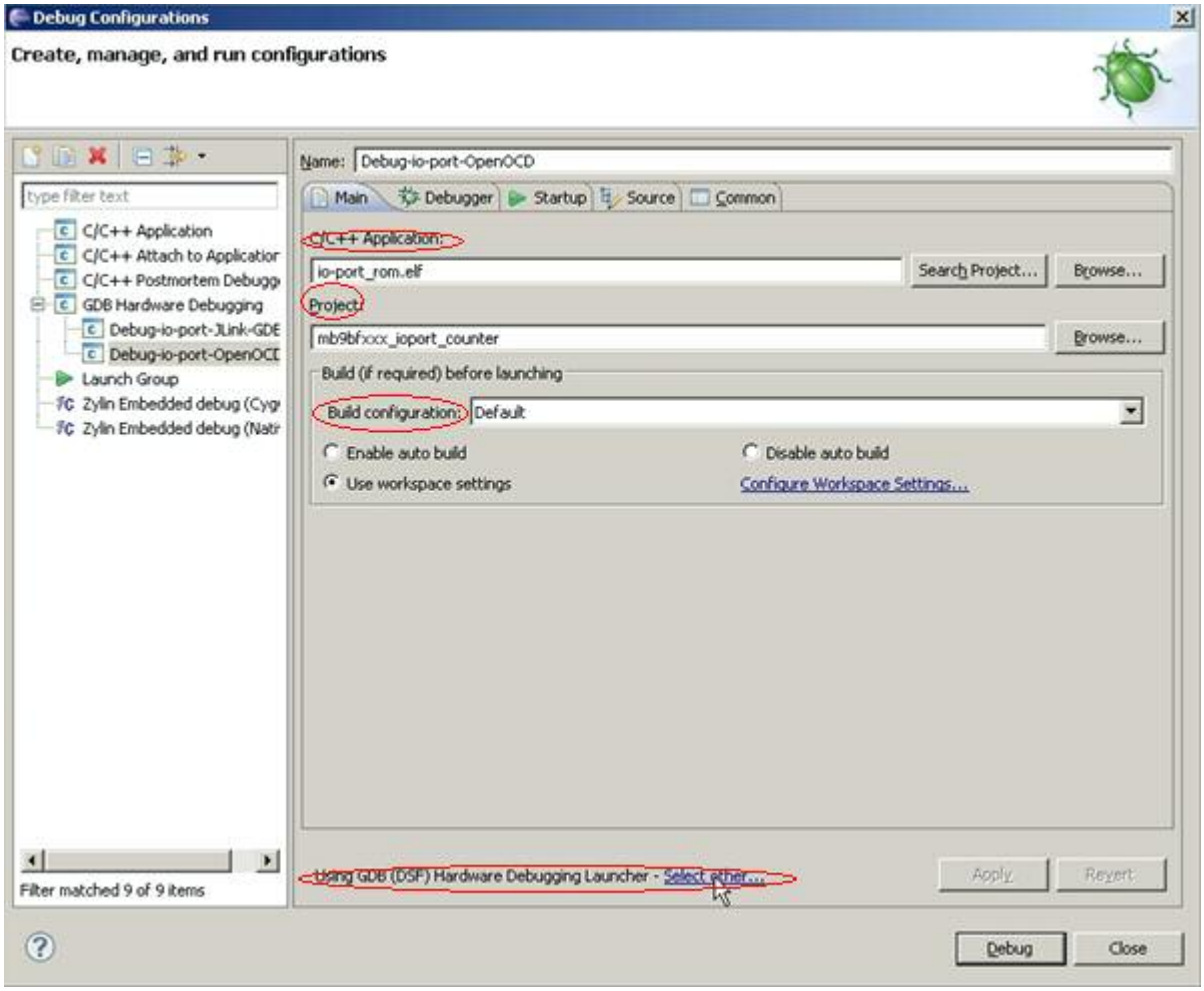
To create a new debug configuration select “GDB Hardware Debugging” and click on *New*.

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).

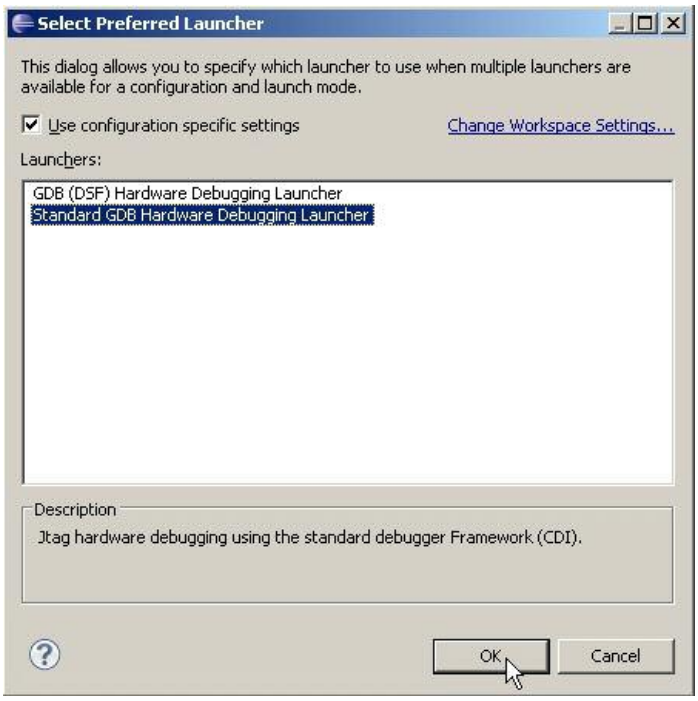
In the “Project” text box, use the *Browse* button to find the project *ioport_sk-fm3-****.

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”.

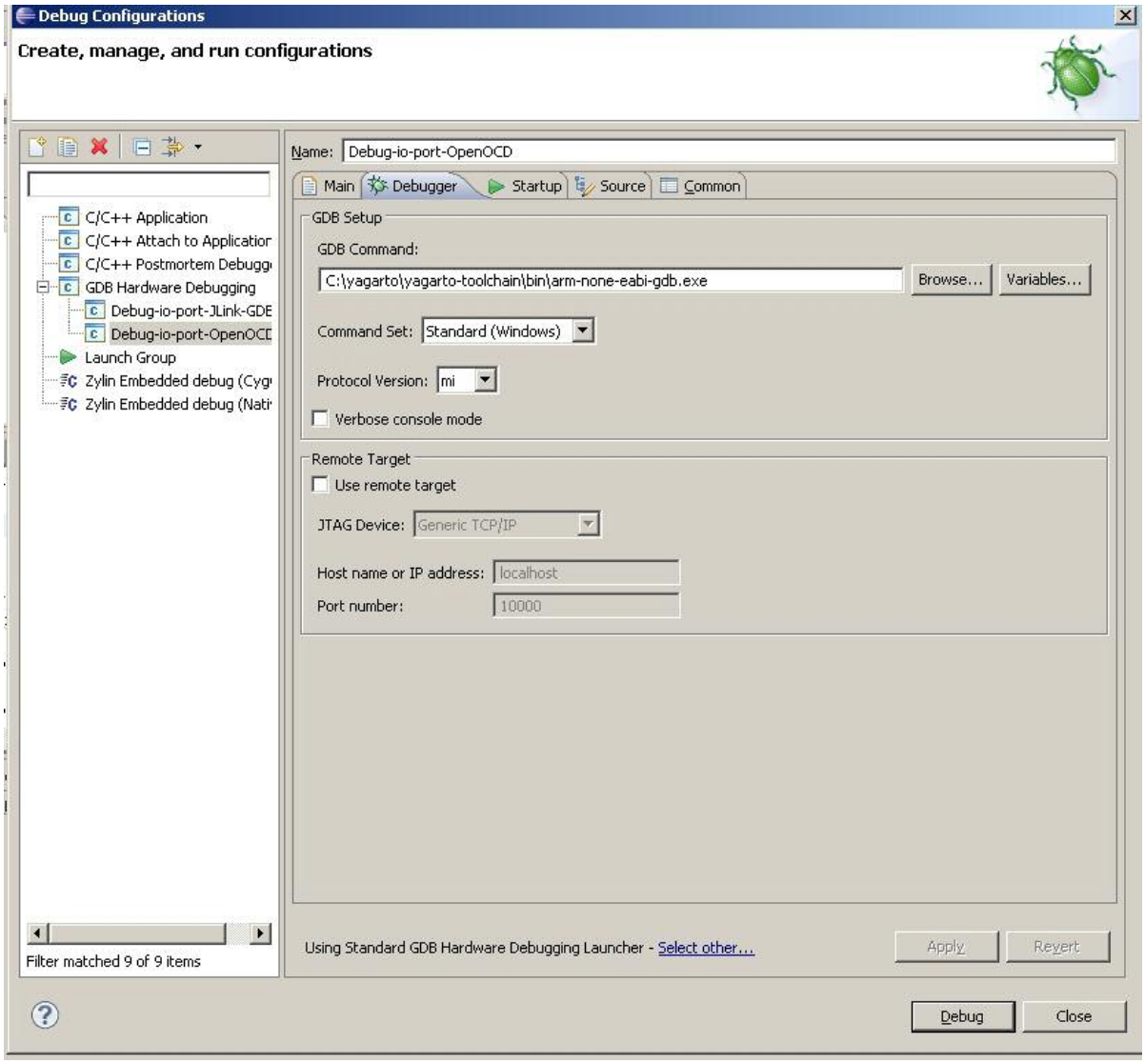


Click on *Select other....* by “Using GDB (DSF) Hardware Debugging launcher” as shown below and select “Standard GDB Hardware Debugging launcher”. Click on *OK*.



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*.



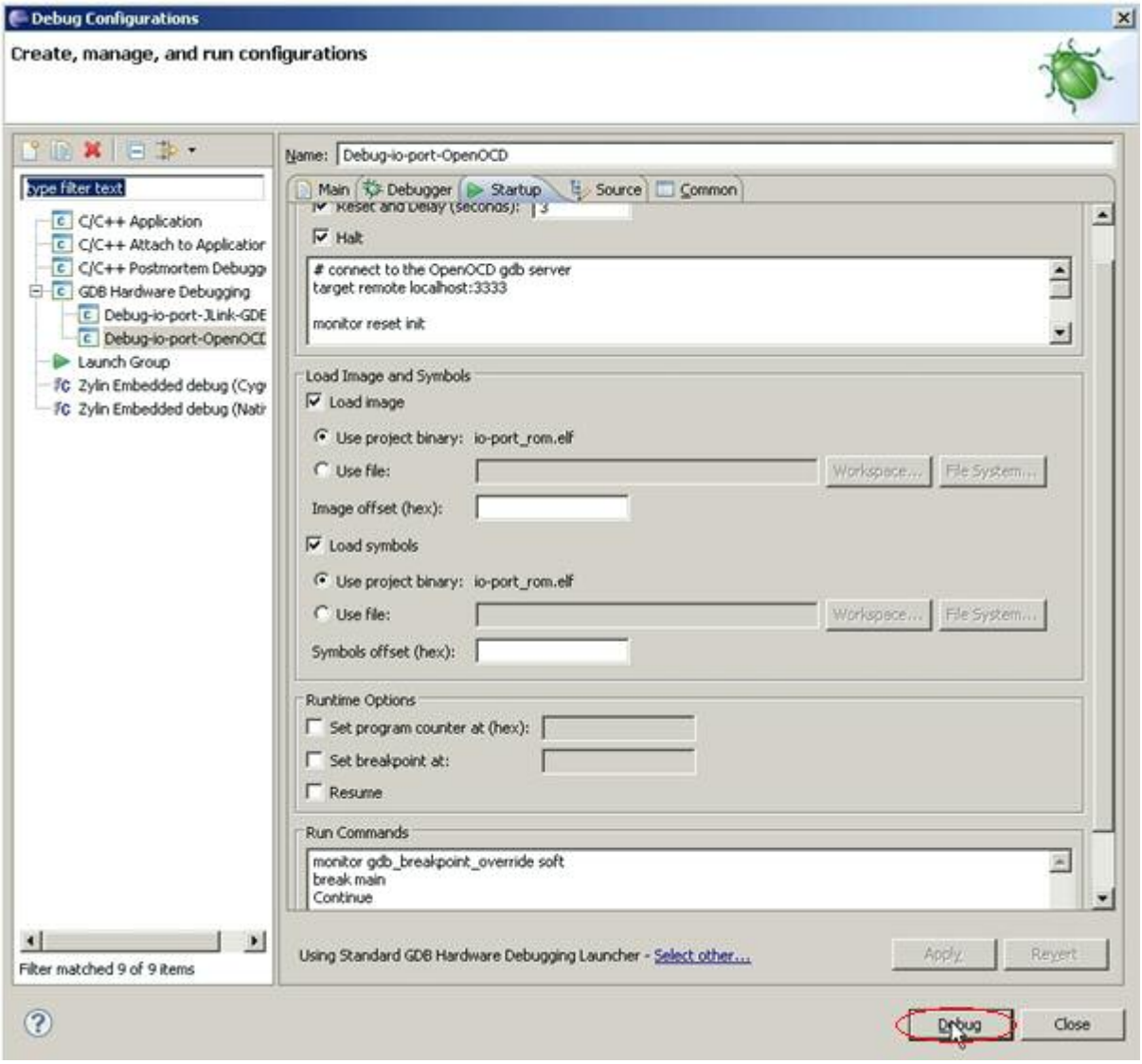
Now select the “Startup” tab as shown below.

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

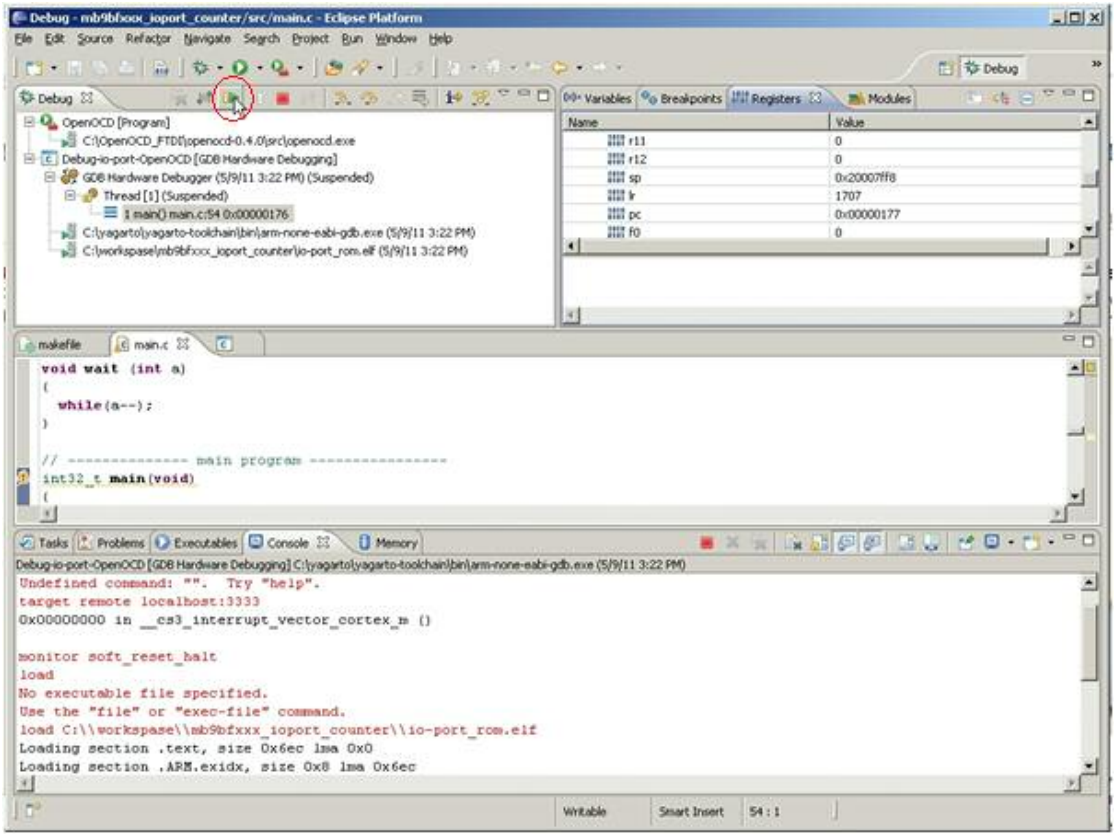
```
# connect to the OpenOCD gdb server
target remote localhost:3333

monitor reset init

monitor soft_reset_halt

load
```

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

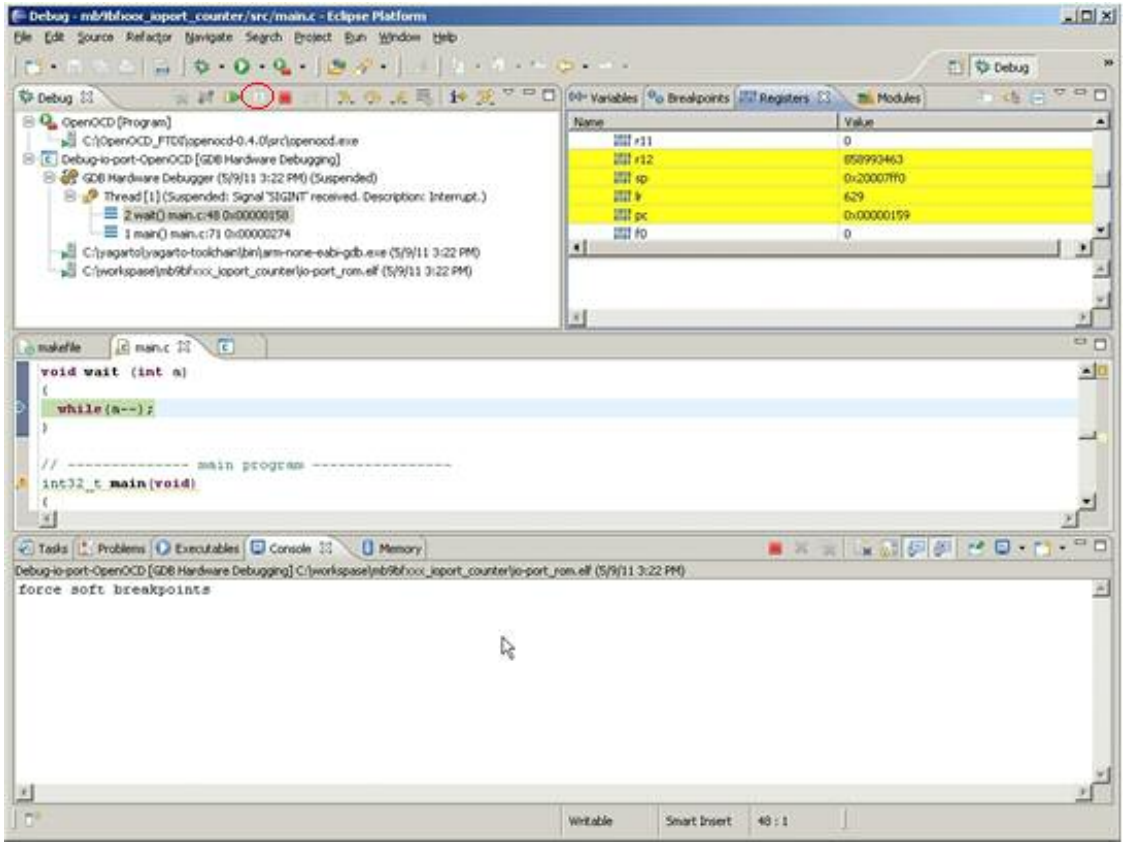
```
monitor gdb_breakpoint_override soft
break main
Continue
```



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.

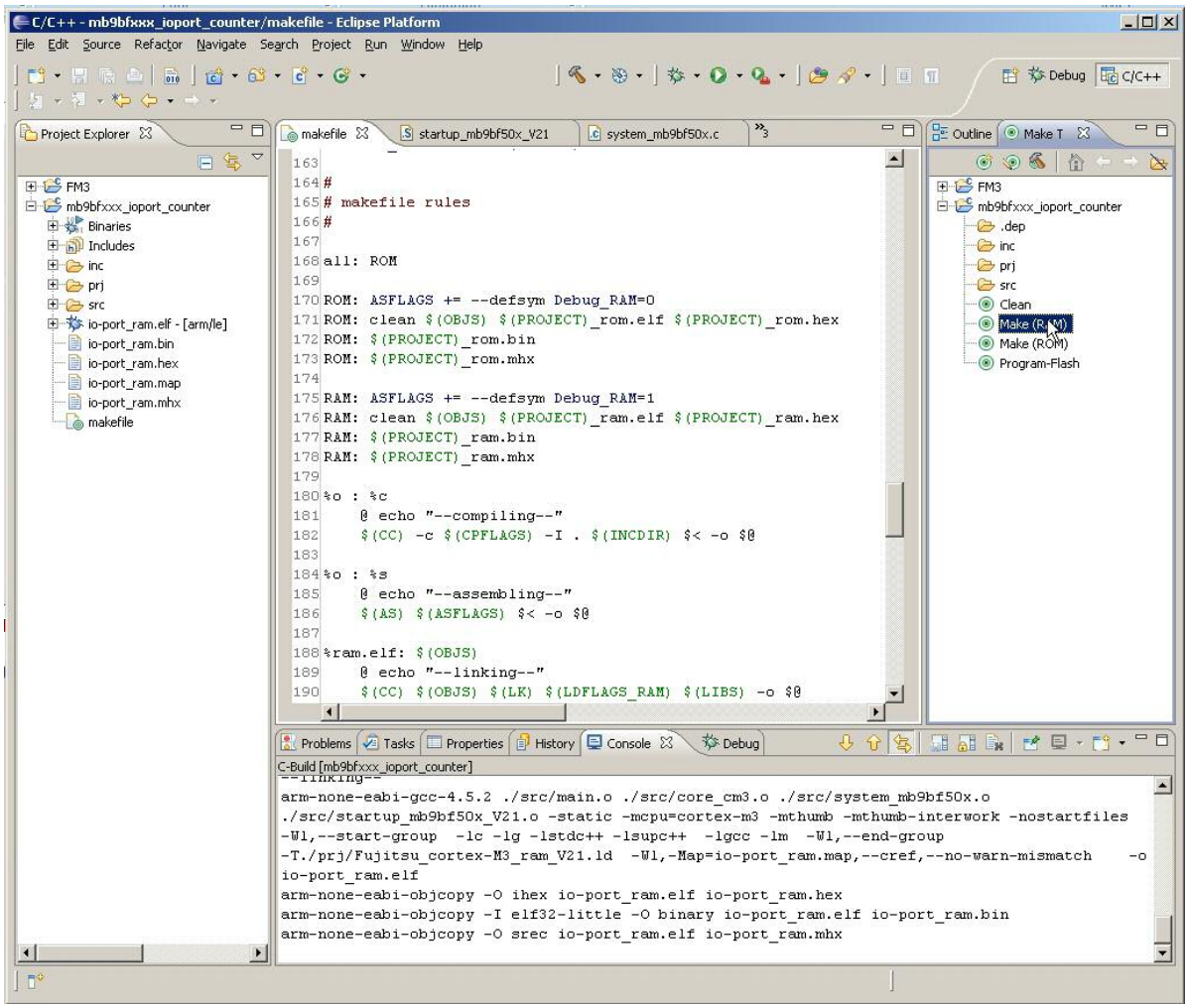


After starting the debug procedure, the debug process can be terminated at any time by clicking on the “Suspend” button.

12.2 Debug on the RAM

In the paragraph before the Flash debug was explained. 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).

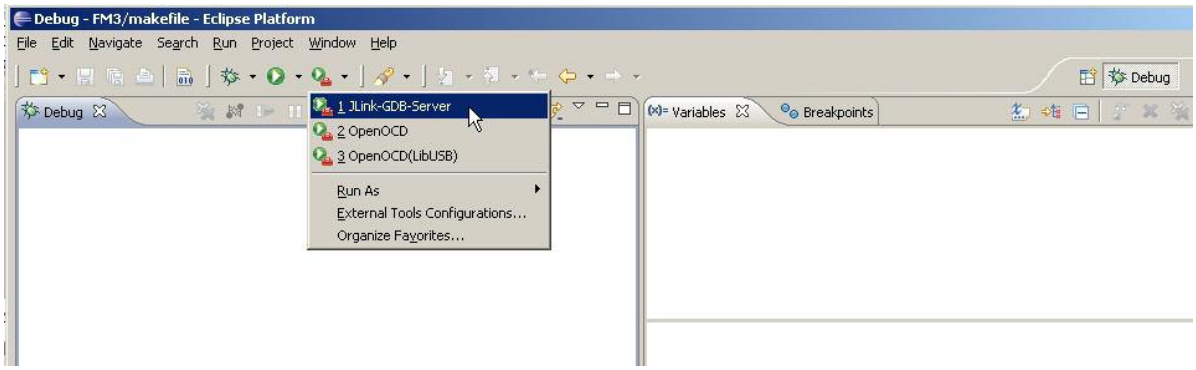


Now switch back to the *Debug perspective* to initiate the RAM debug process.

12.2.1 Using J-Link GDB Server to Debug the RAM Application

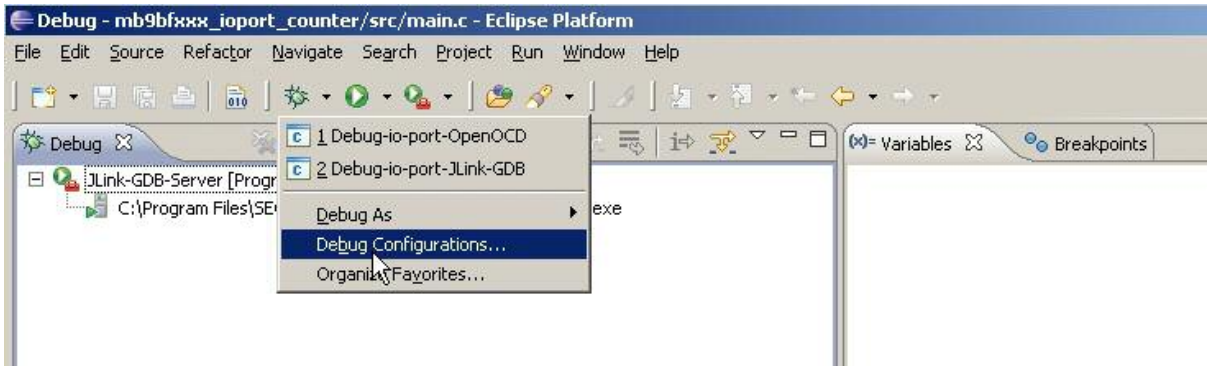
Reconnect the SK-FM3-100PMC board via the JTAG interface “J-Link” to the USB interface of your computer.

After this start the “J-Link GDB Server”. Click on “J-Link GDB Server” and the external tool will be started then as shown below.



The GDB server requires a license before starting. See chapter 4.3 for terms of usage.

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



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

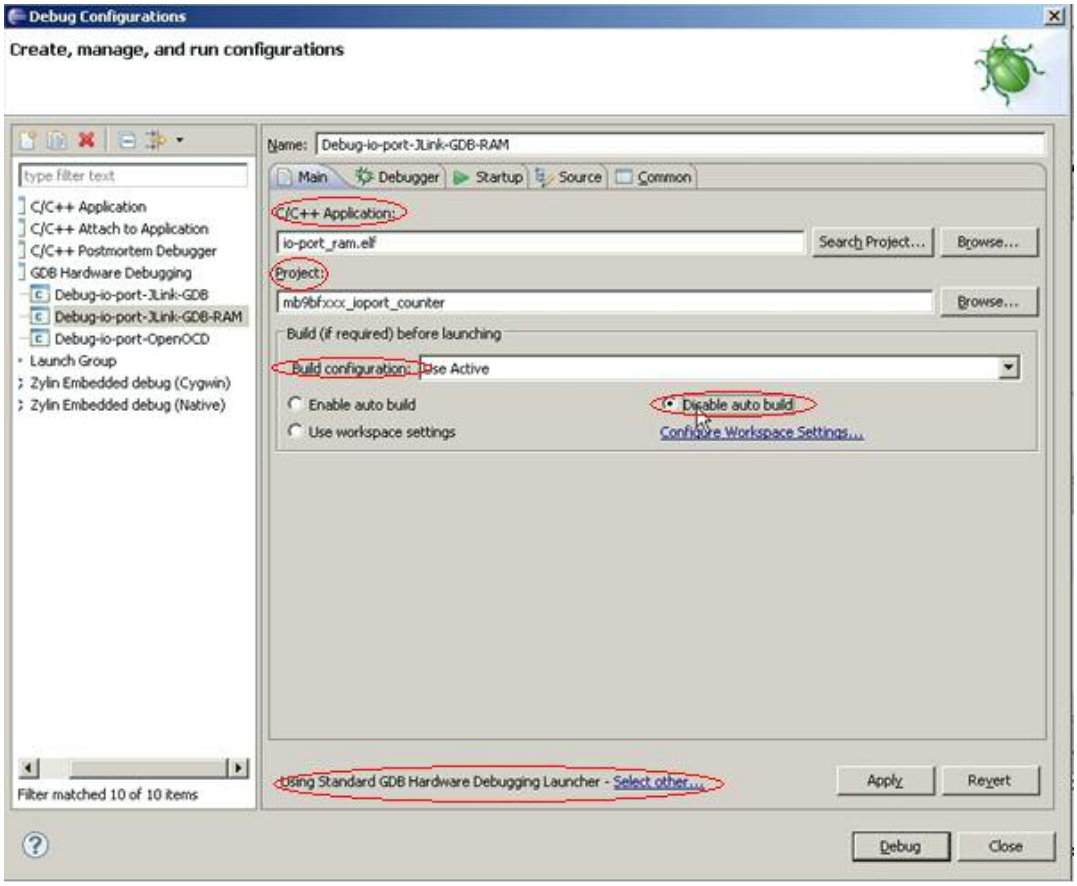
Rename the debug configuration. For differencing the RAM debug from the Flash debug, give the name also an suffix “_RAM” to avoid confusions with the configurations already saved.

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

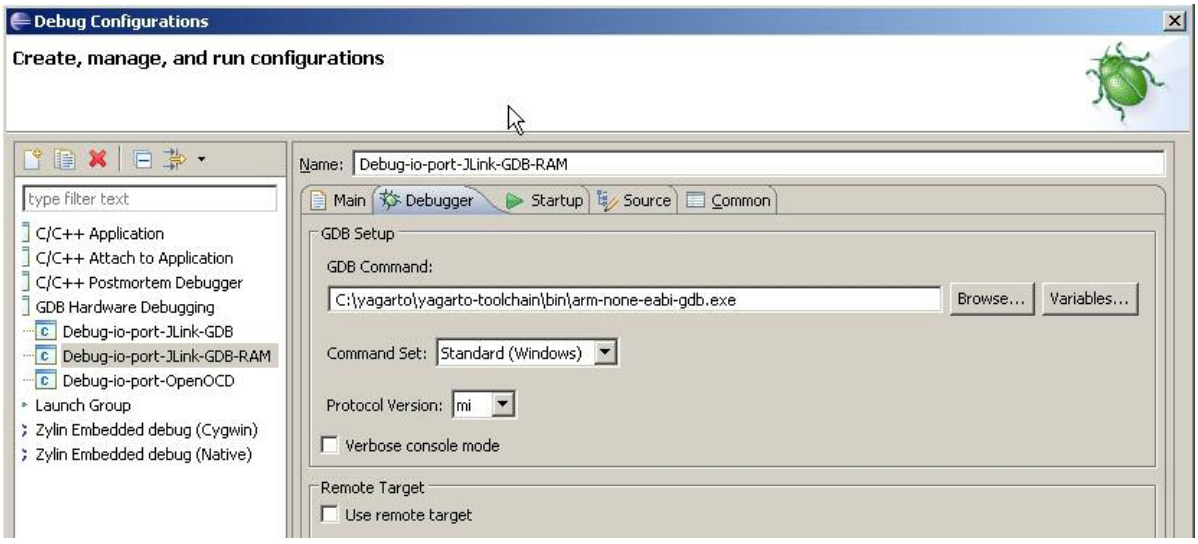
In the “C/C++ Application” text box, use the *Search Project…* button to find the application file *io-port_ram.elf*.

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*.



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



In the “Startup” tab copy on “Initialization Commands” panel the following commands:

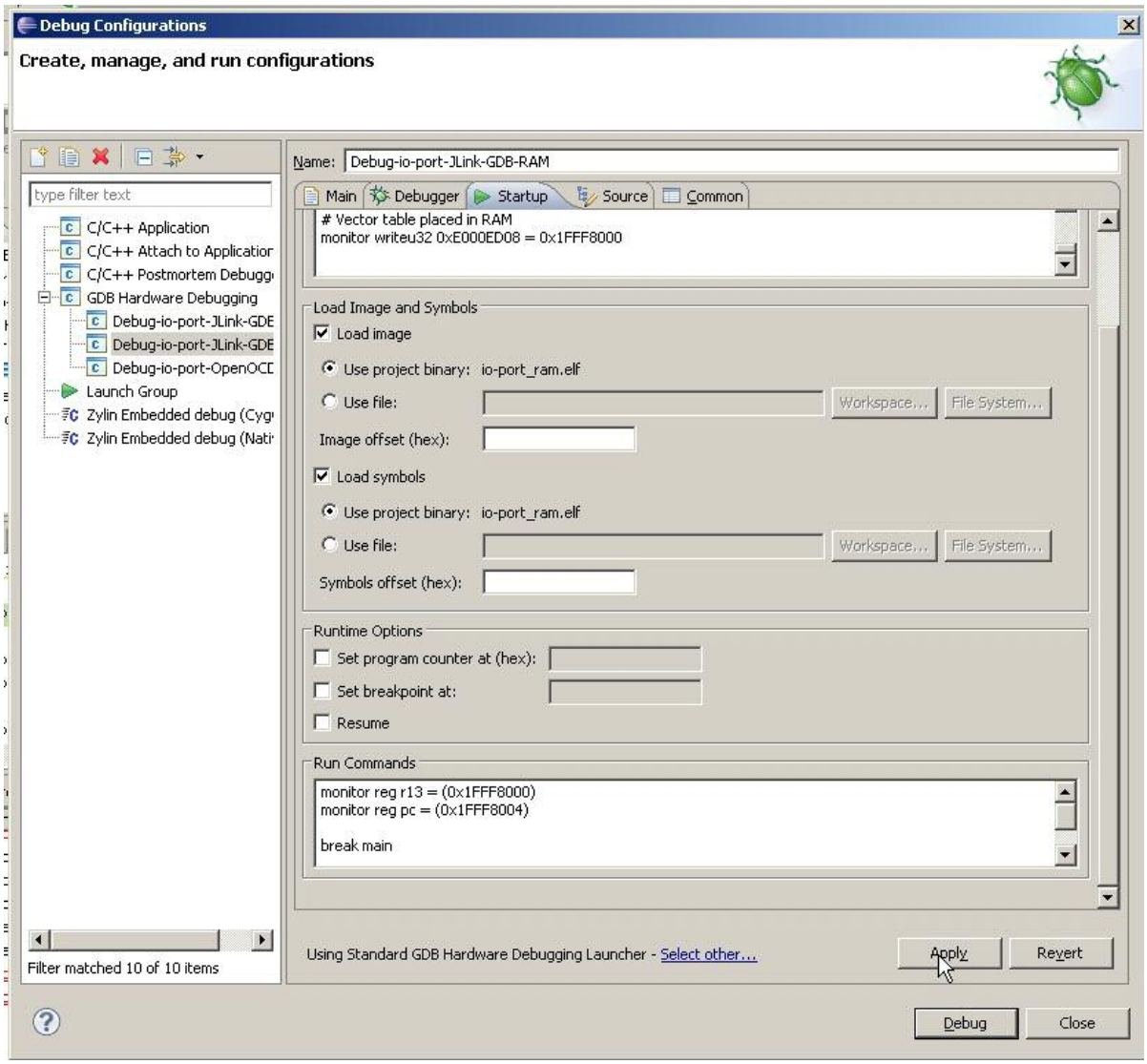
```
# connect to the J-Link gdb server
target remote localhost:2331

# Set gdb server to little endian
monitor endian little

# Set JTAG speed to 5 kHz
monitor speed 5

# Reset the target
monitor reset
monitor sleep 100

# Set JTAG speed in khz
monitor speed auto

# Vector table placed in RAM
monitor writeu32 0xE000ED08 = 0x1FFF8000
```

Use RAM start (Vector table start) for address!

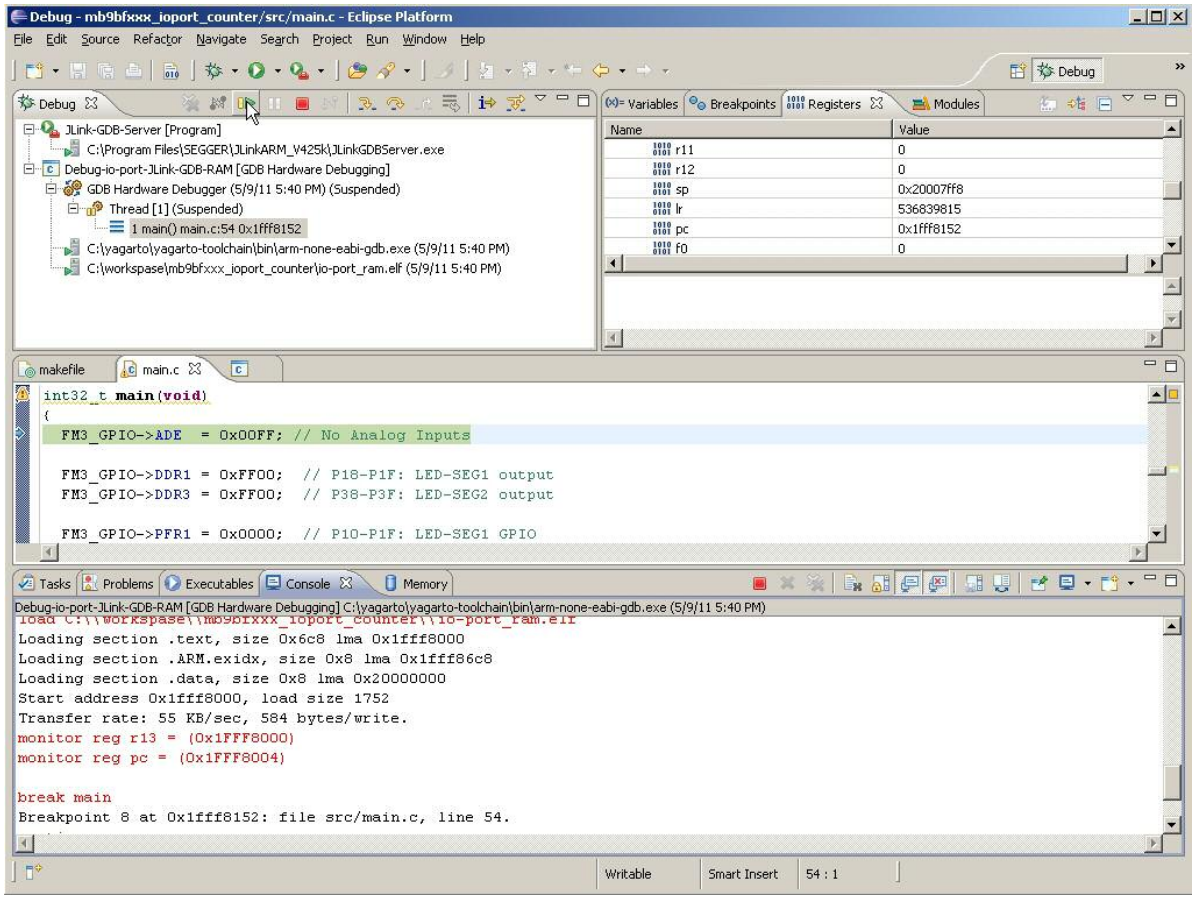
In the “Startup” tab copy on “Run Commands” panel the following commands:

```
monitor reg r13 = (0x1FFF8000)
monitor reg pc = (0x1FFF8004)

break main
continue
```

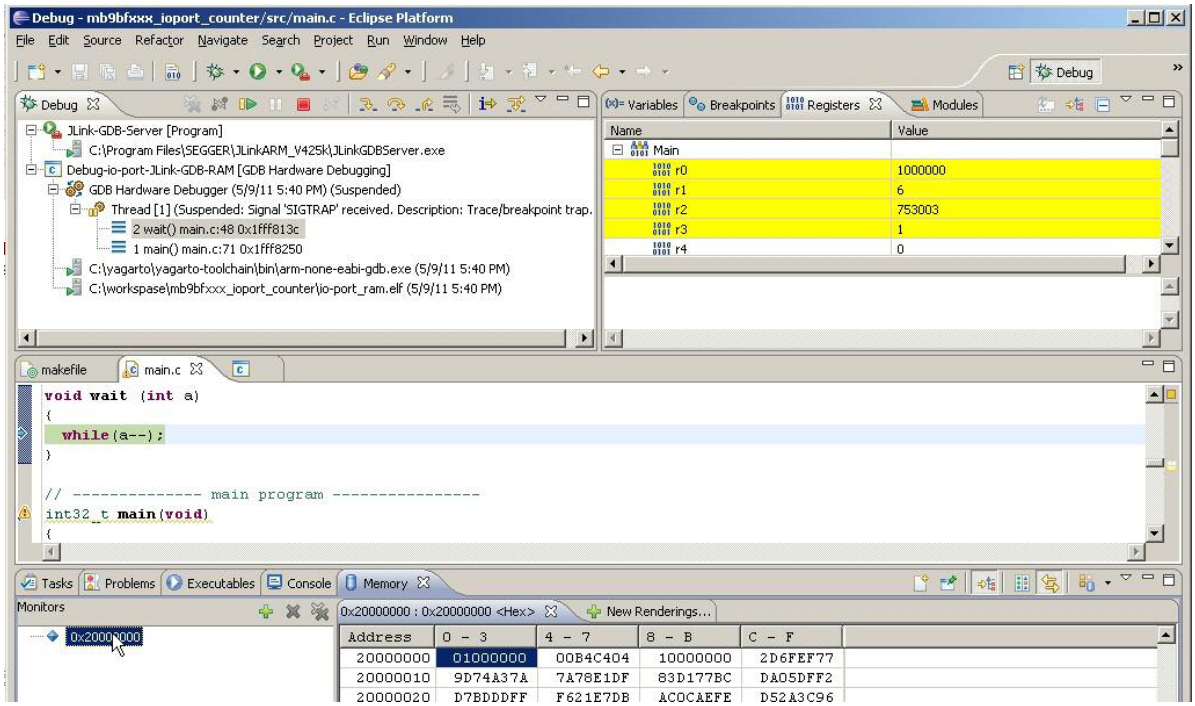
Use RAM start (Vector table start) + 4 Bytes for address!



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



The figure shows that the successful RAM debug process start. To resume, simply click on the *Resume* button.

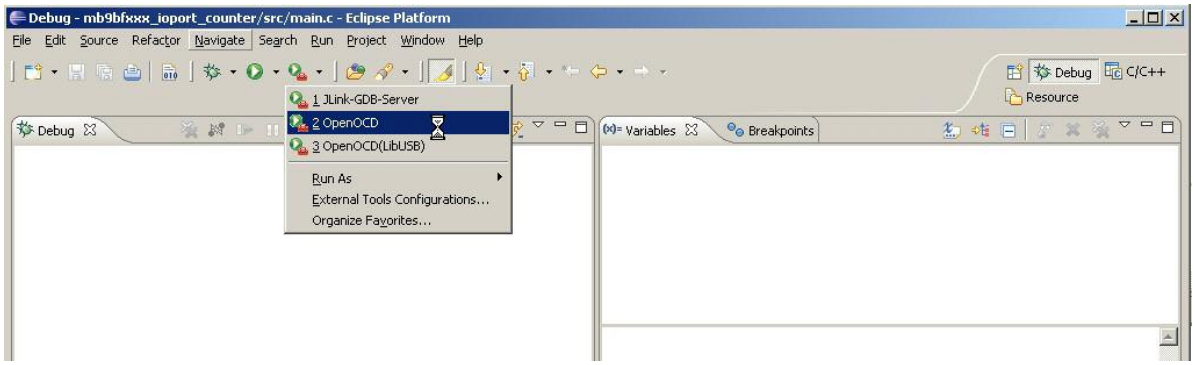


To get e.g. a view of the target memory, select the *Memory* view. Therefore choose from the Eclipse menu select *Window* and under *Show View*.

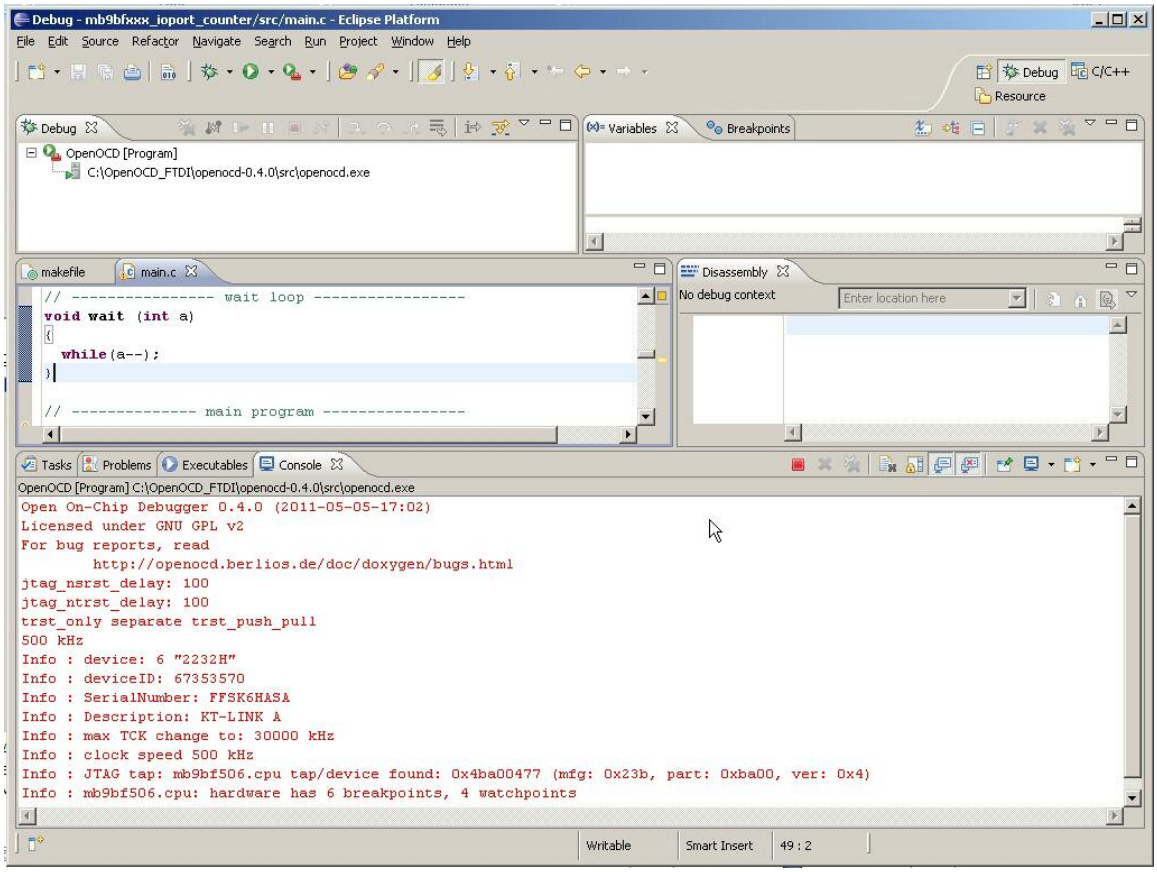
12.2.2 Use OpenOCD to debug the RAM application

Reconnect the SK-FM3-100PMC board via the JTAG interface “KT-Link” to the USB interface of your computer.

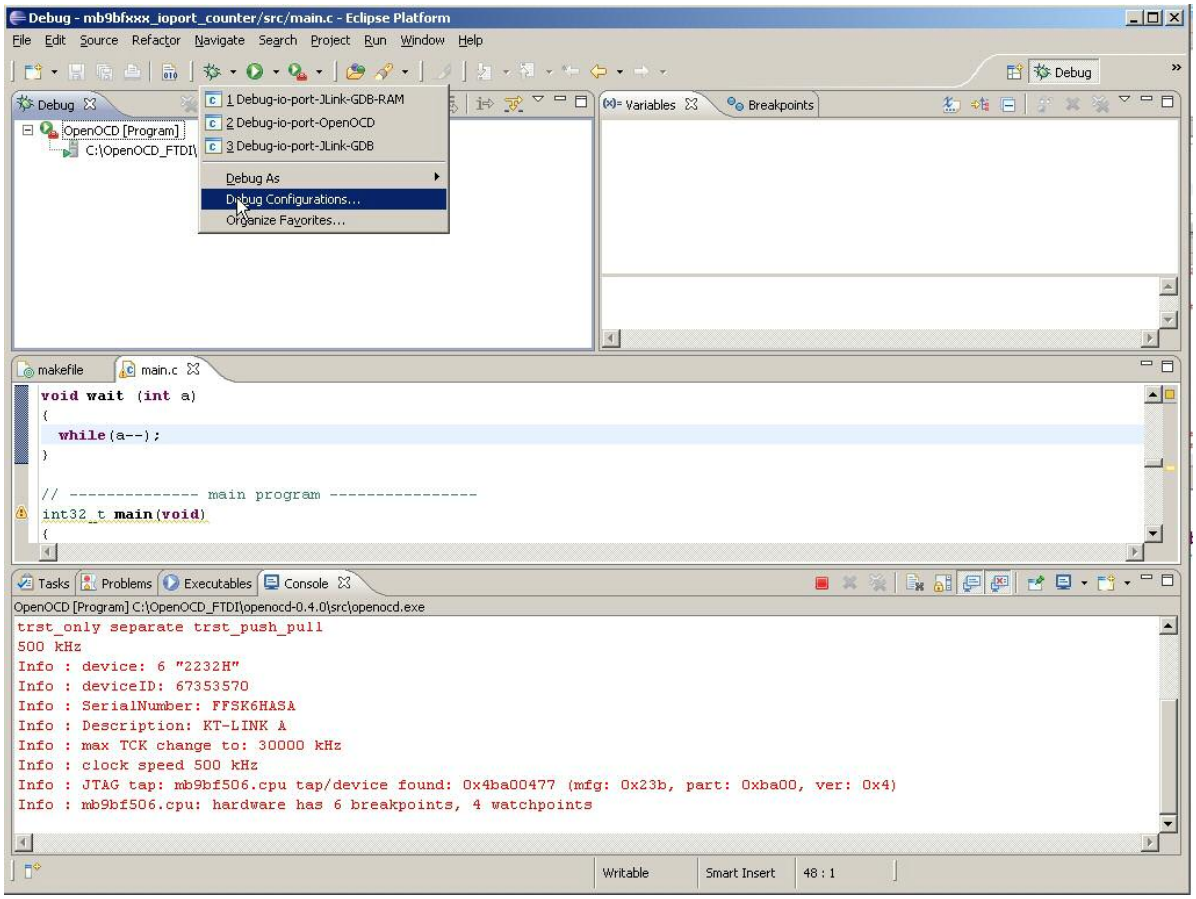
After this, start the “OpenOCD”. Therefore click on *OpenOCD* and the external tool will be then started as shown below.



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



Create a new “Debug Configuration”. For this click on *Debug Configurations...* as shown below.



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

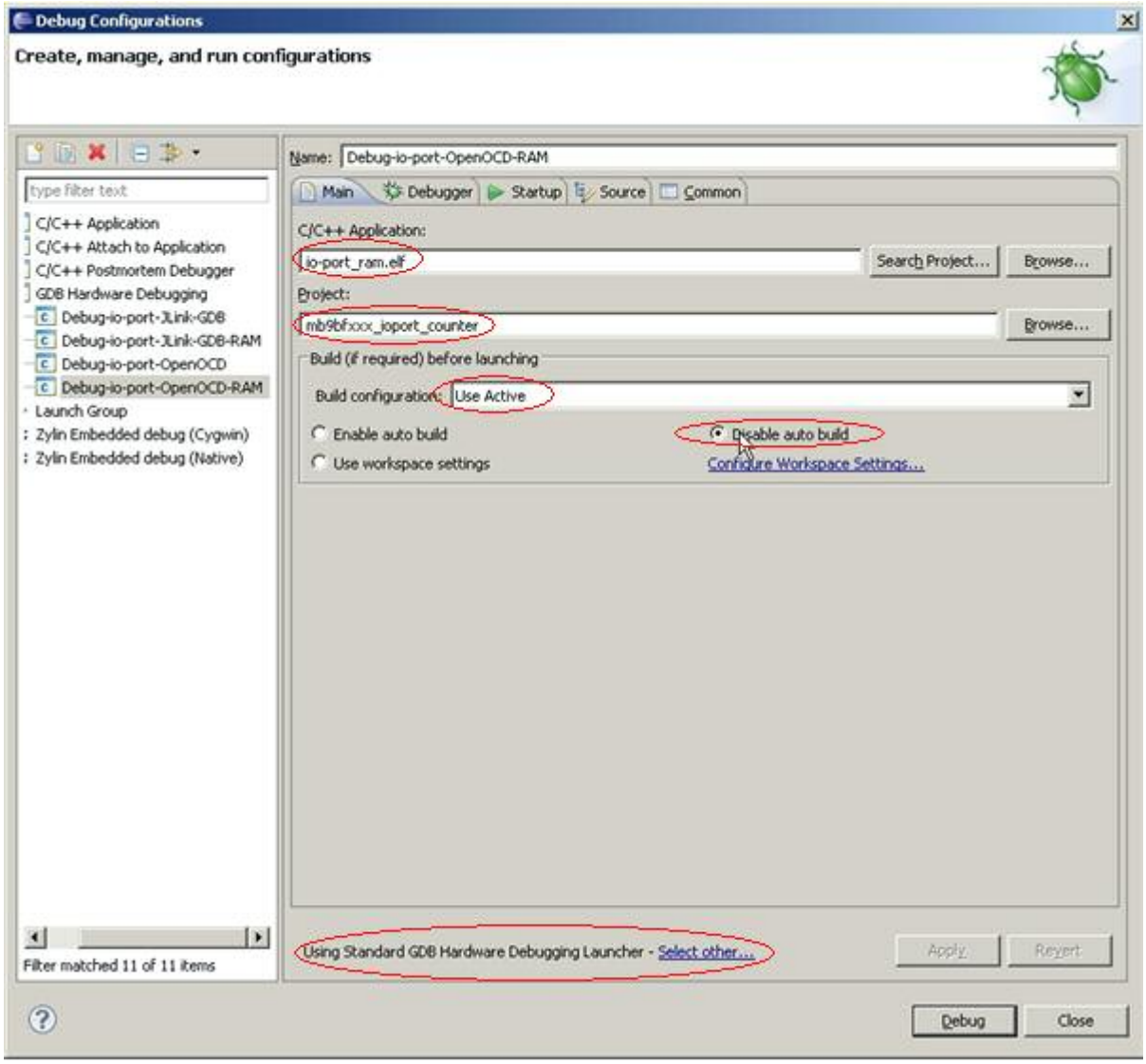
Rename the debug configuration. For distinguishing between RAM debug and Flash debug, give the name a suffix “_RAM” to avoid all confusion with the configurations already saved.

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 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* then.



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 0x1fff8000

load
```

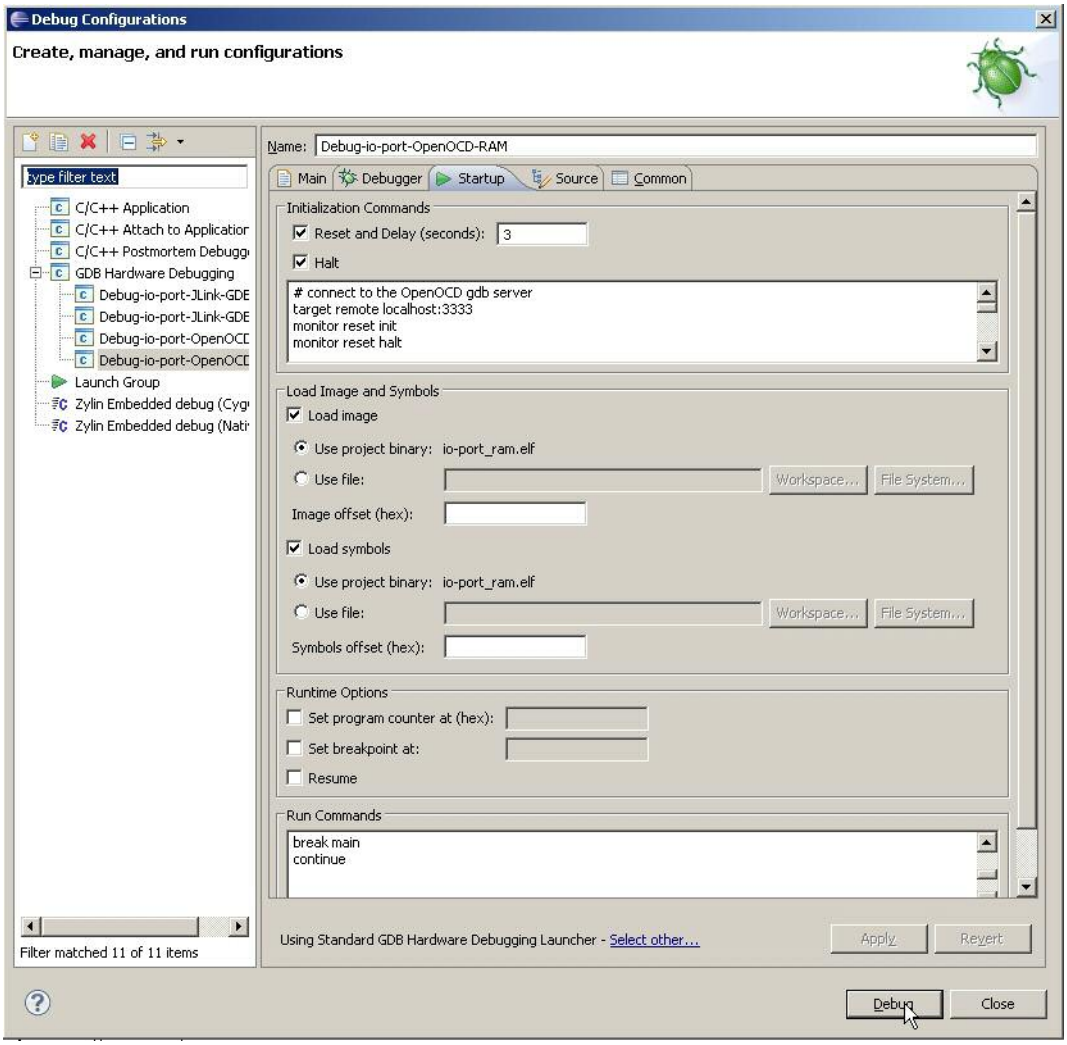
Use RAM start (Vector table start) for address!

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

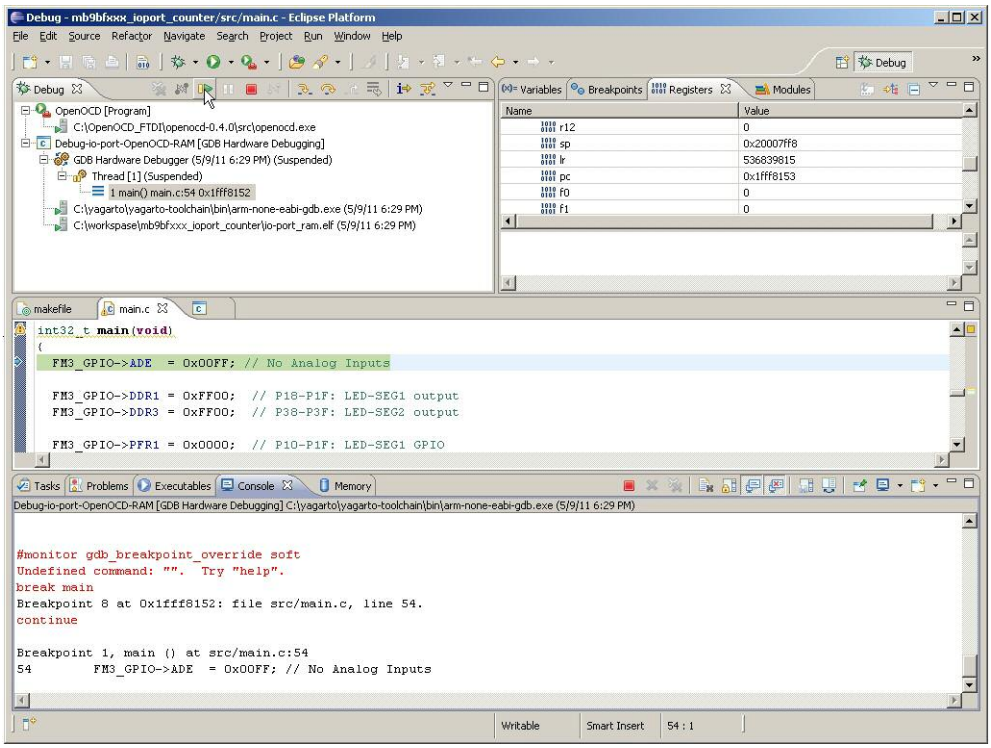
```
break main
set $r13 = *(int*)0x1fffE000
set $pc = *(int*)0x1fff8004
continue
```

Use RAM start (Vector table start) for address!

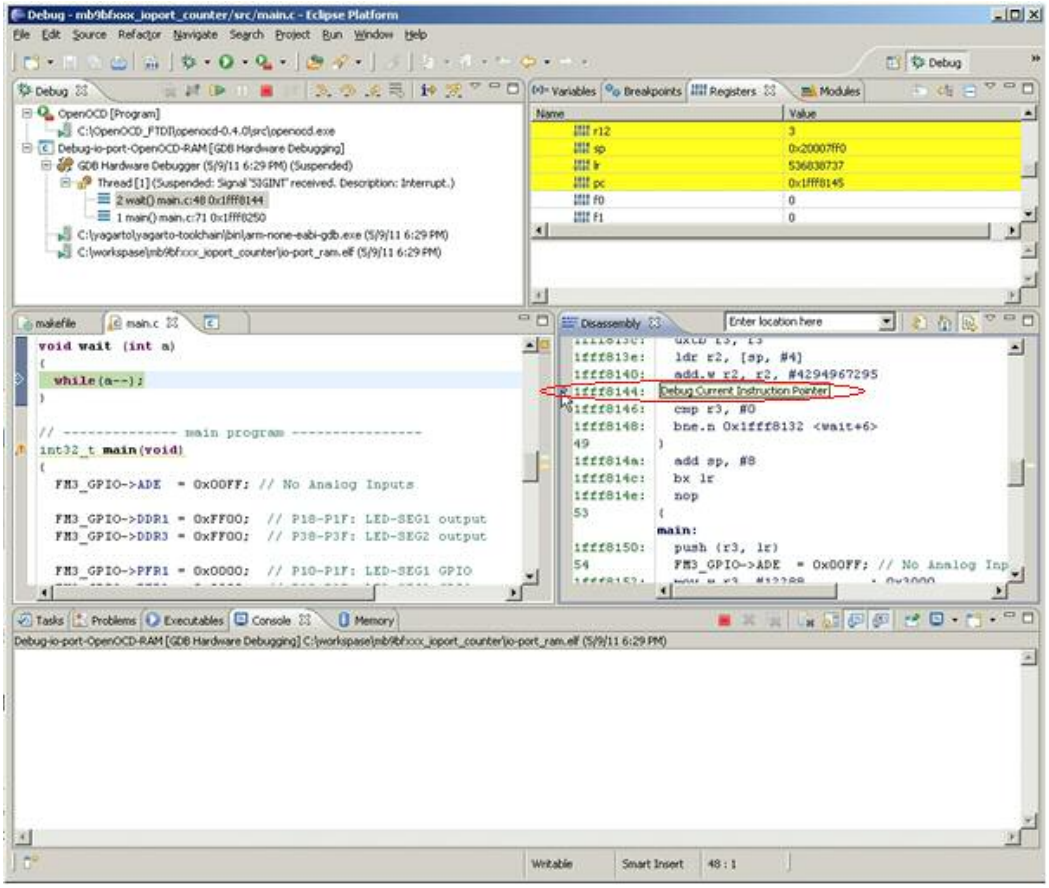
Use RAM start (Vector table start) + 4 Bytes for address!



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*.

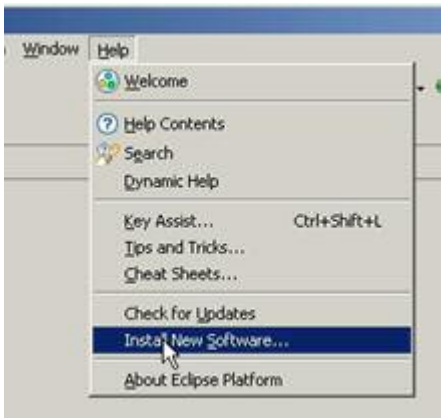
13 Eclipse Embedded Systems Register View Plug-In

HOW TO ADD A 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.

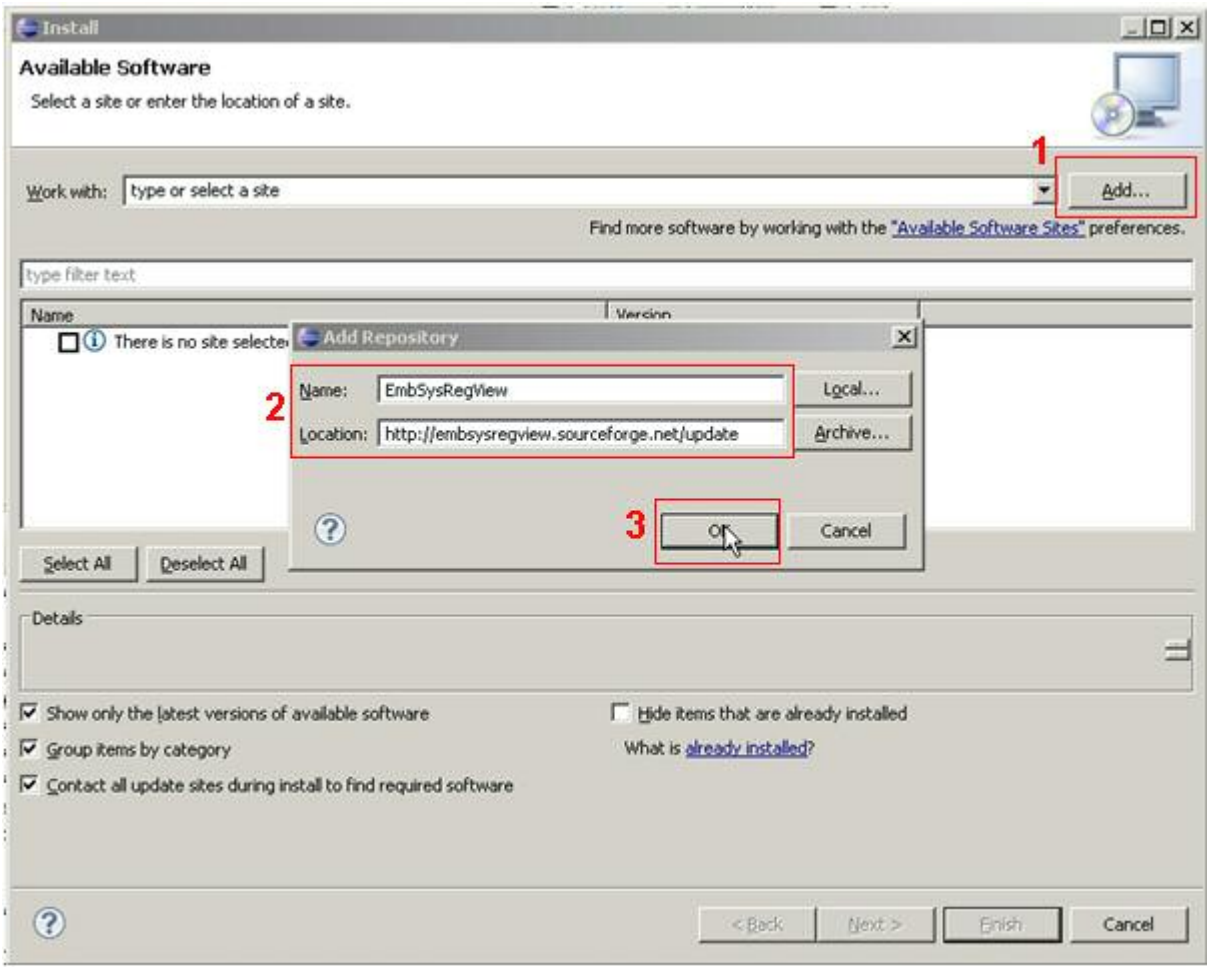
13.1 Plug-in installation

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

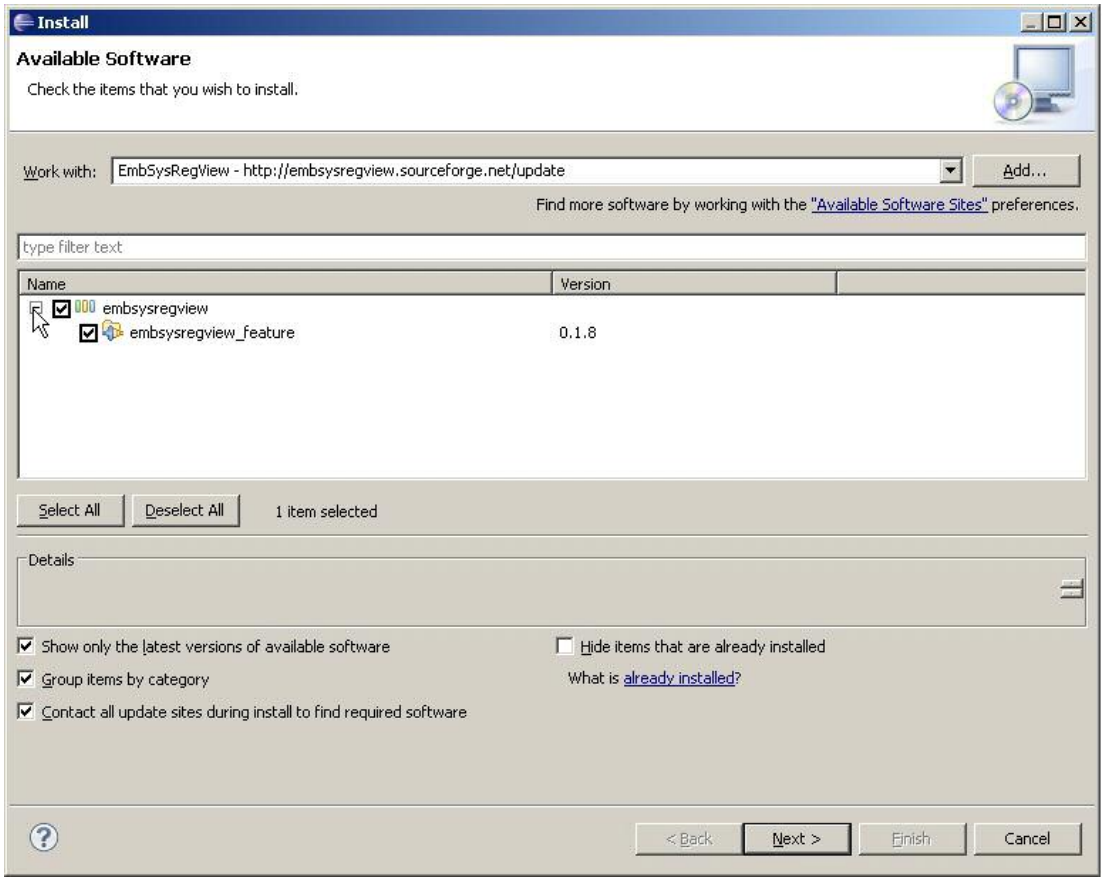


Click on the *Add* button. Enter, e.g. “EmbSysRegView” as name and in the location text box the following link: http://embsysregview.sourceforge.net/update

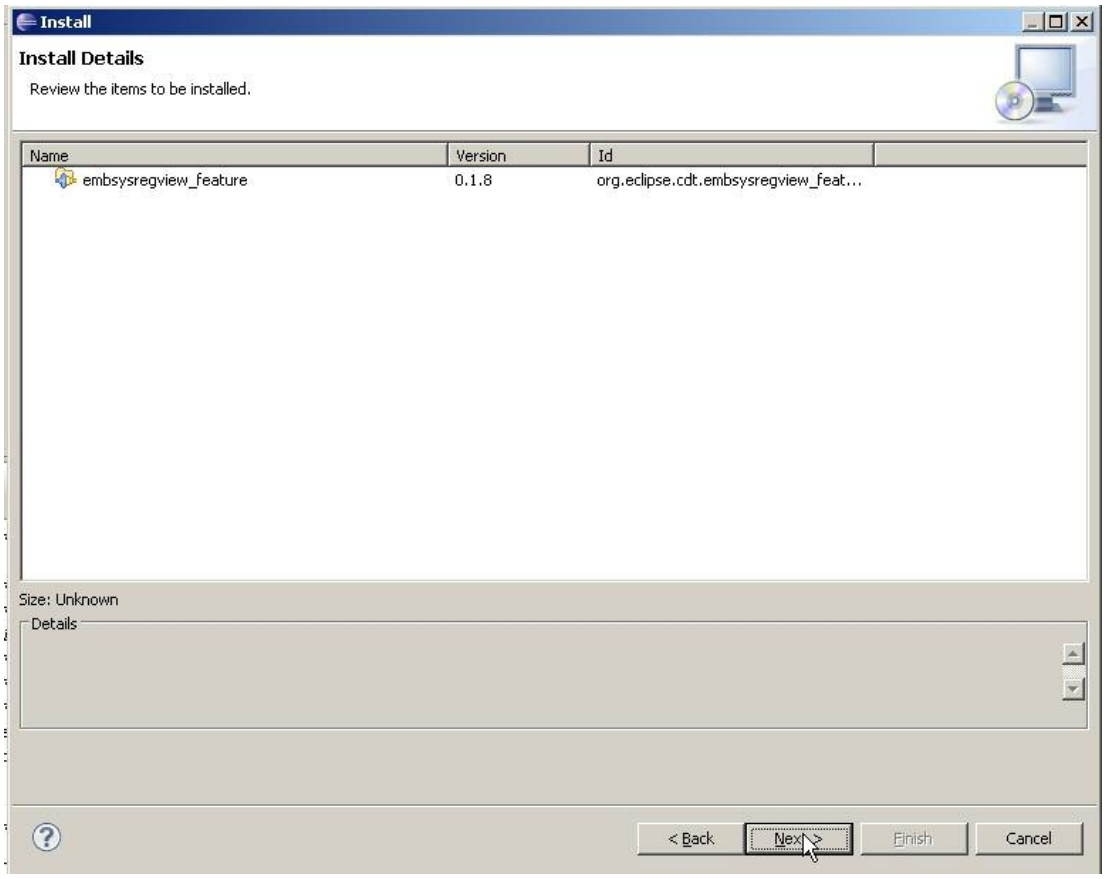
Confirm the repository with *OK*.



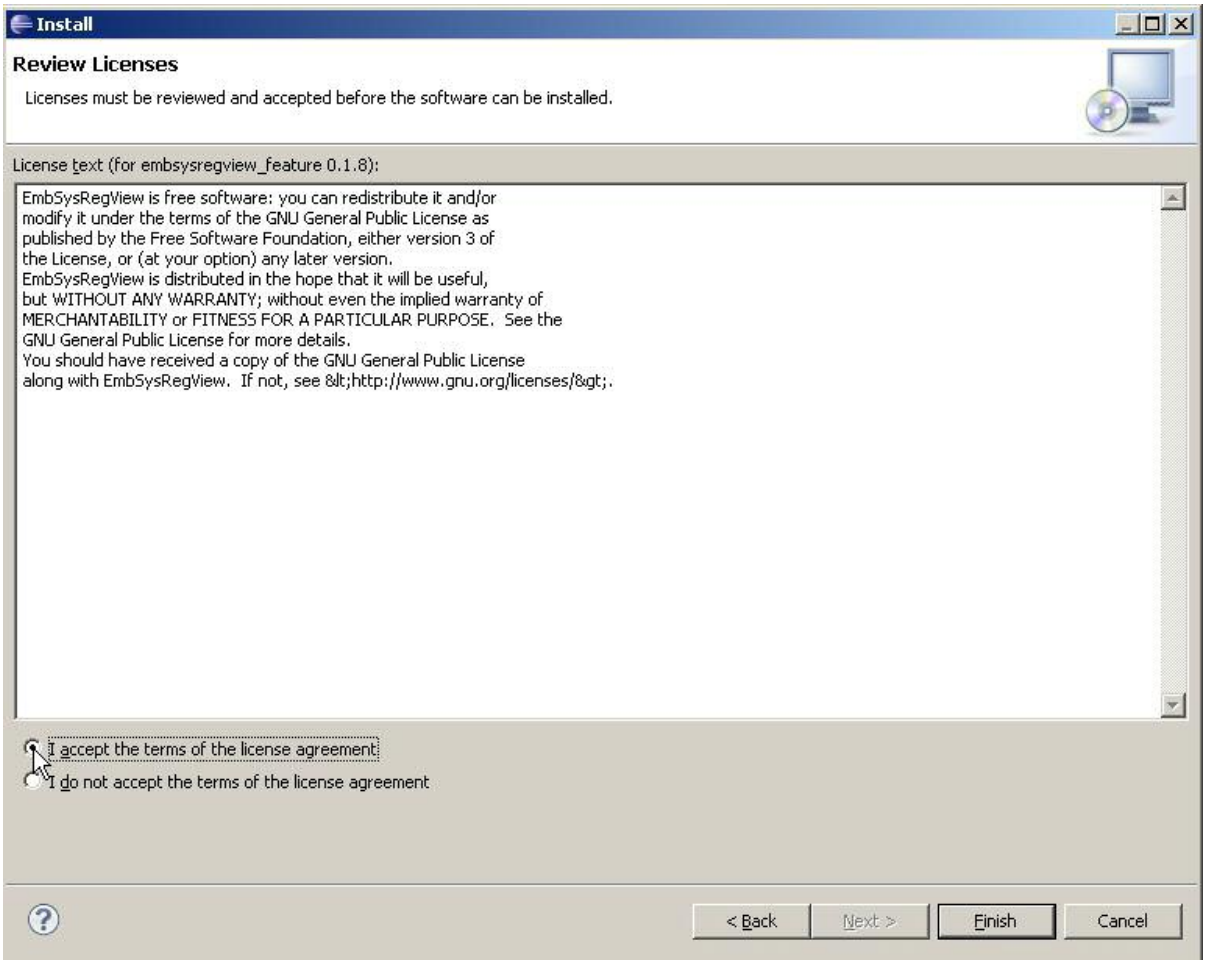
After the confirmation select all plug-in feature and click on *Next*.



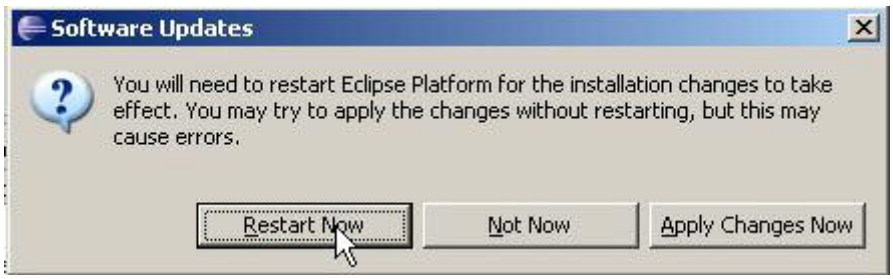
Click on *Next* to confirm the installation detail.



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*.

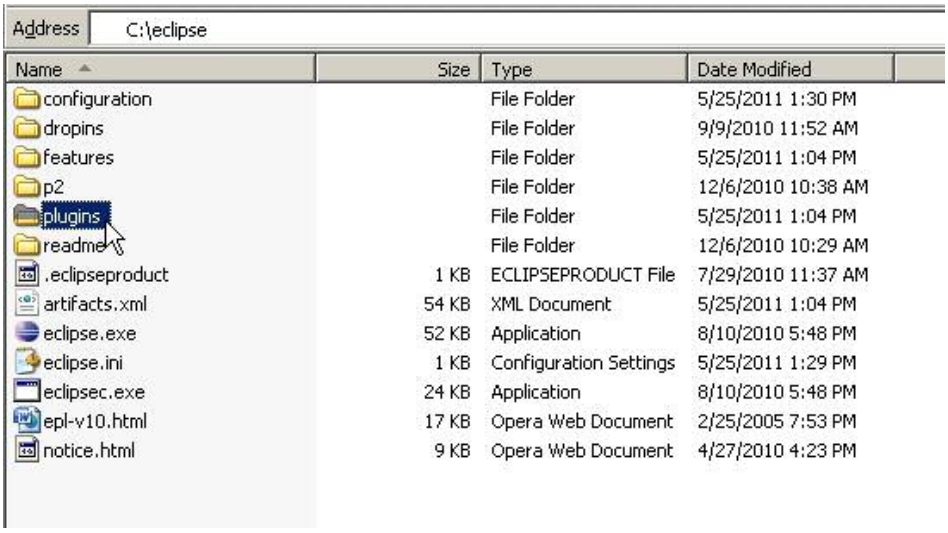


The Eclipse software are now up-to-date and the “EmbSysRegView” is also installed.

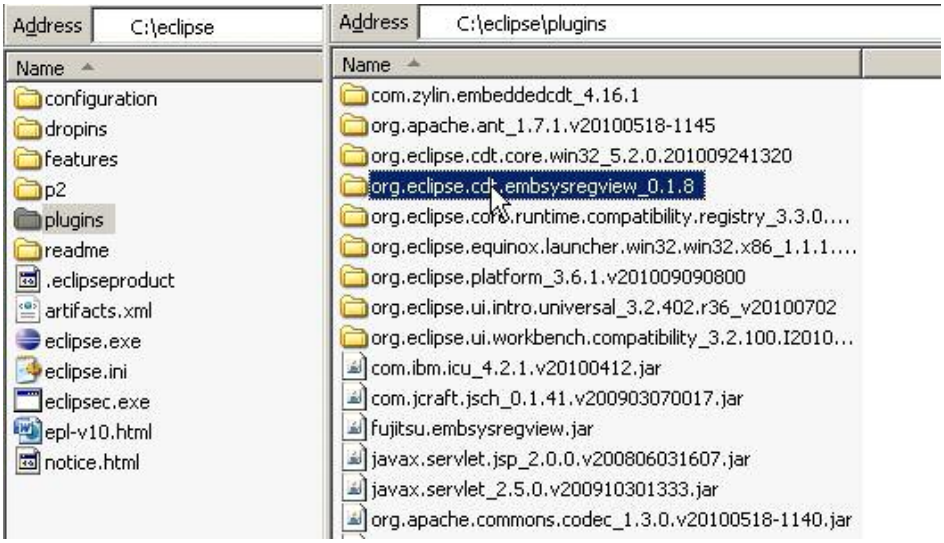
13.2 Using the Eclipse Register View

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 Fujitsu, 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:

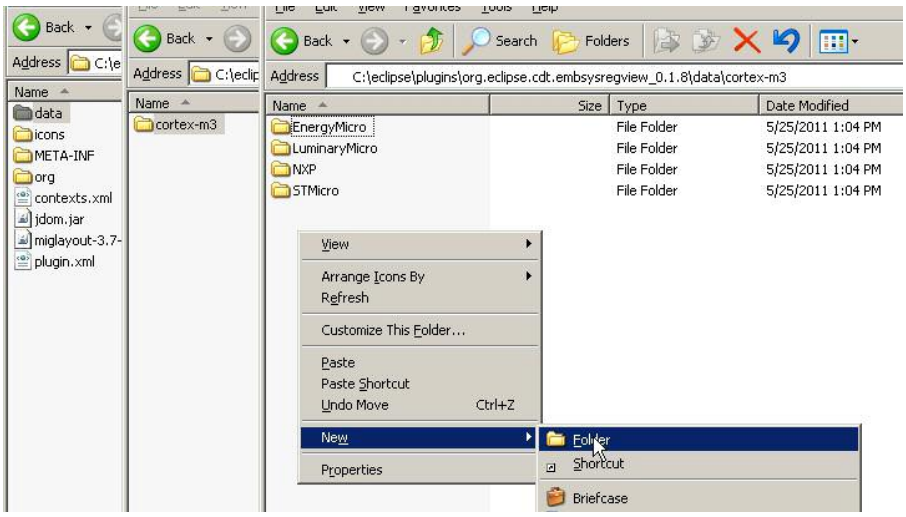


Open the directory *\plugins* and look for the installation directory for the installed plug-in "EmbSysRegView".

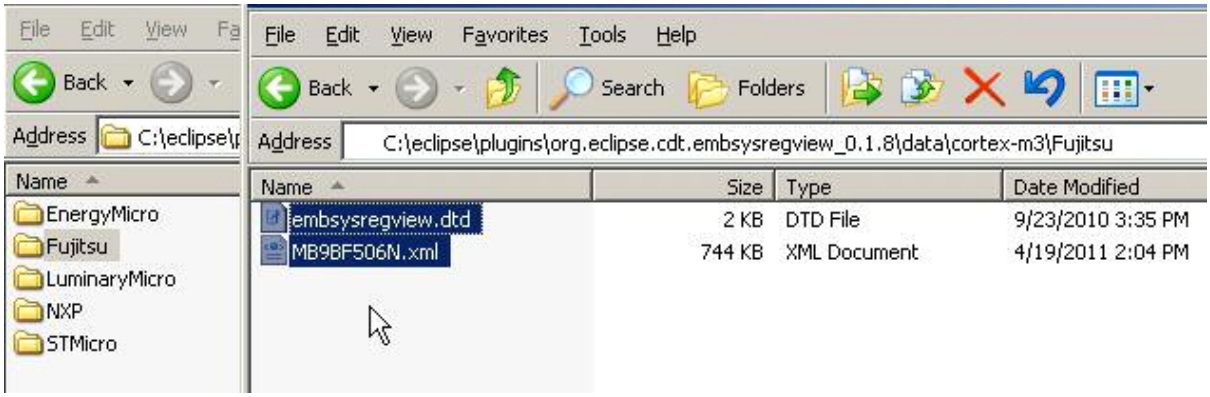


Open the selected directory and create a new folder with the name e.g. *Fujitsu* to directory:

\data\cortex-m3

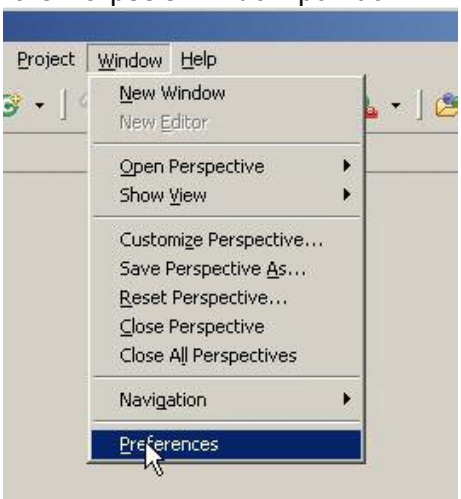


When the folder *Fujitsu* is created, add both description files *embsysregview.dtd* and *MB9BF506N.xml* to it.

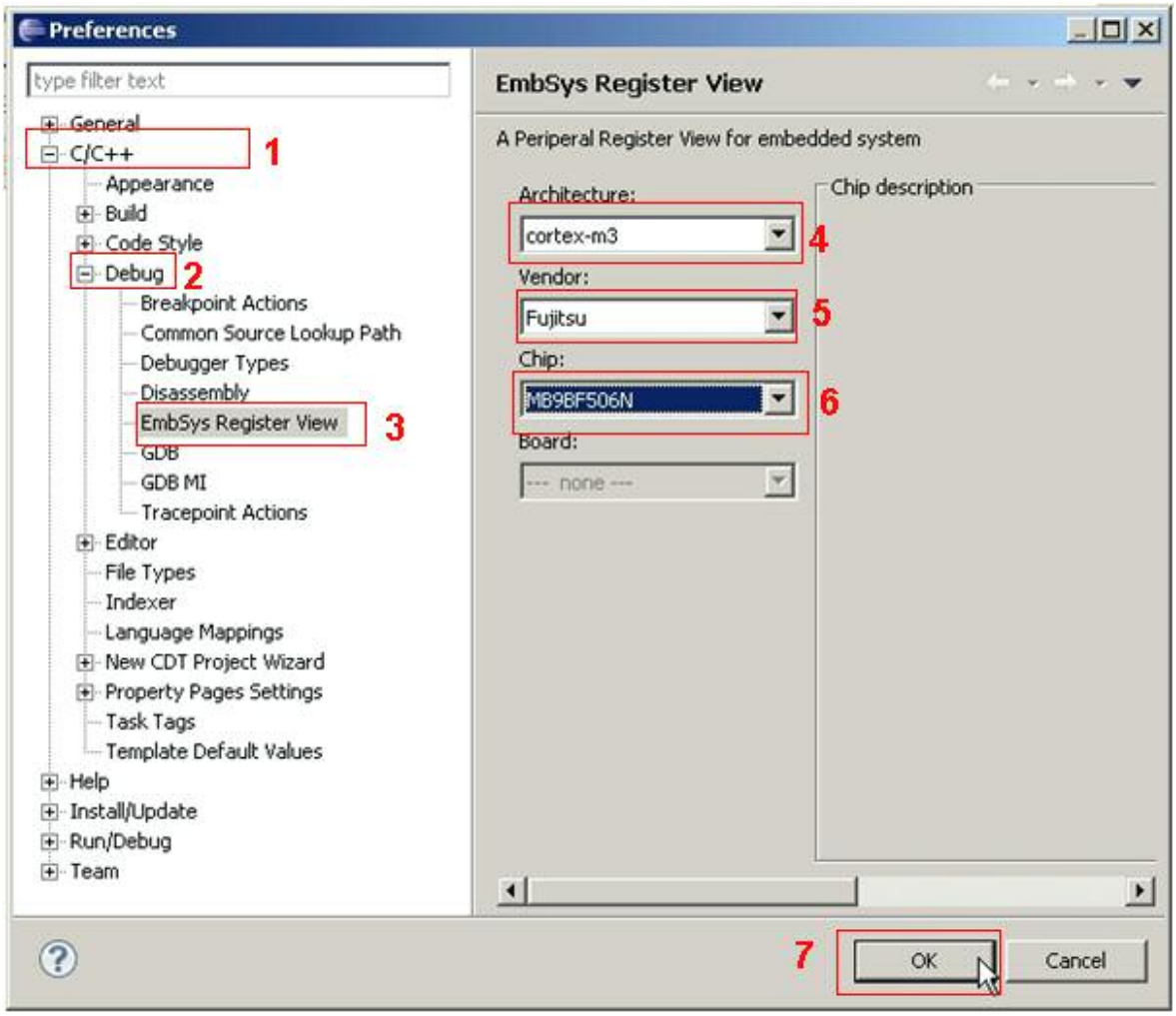


Now go back to Eclipse IDE and use the installed Register view.

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

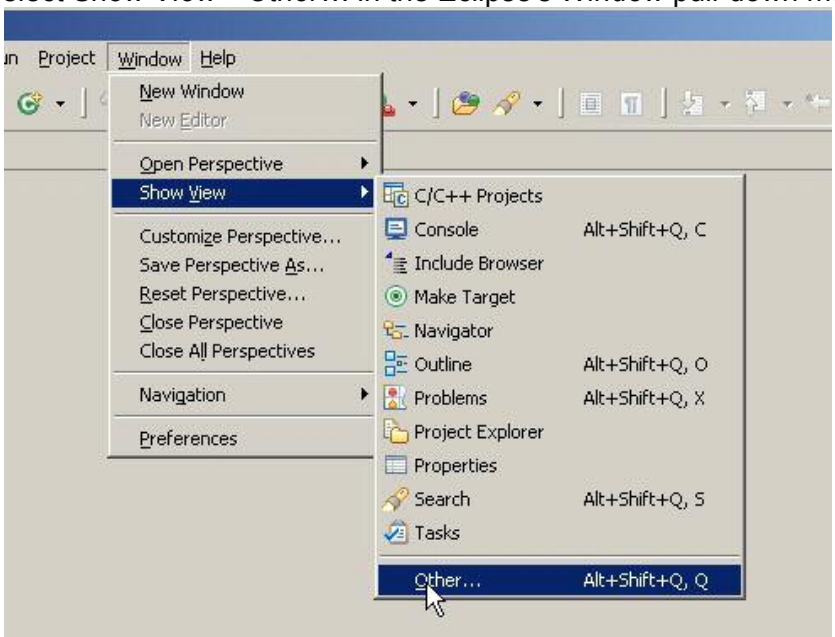


Select the correct device as shown in the figure below.

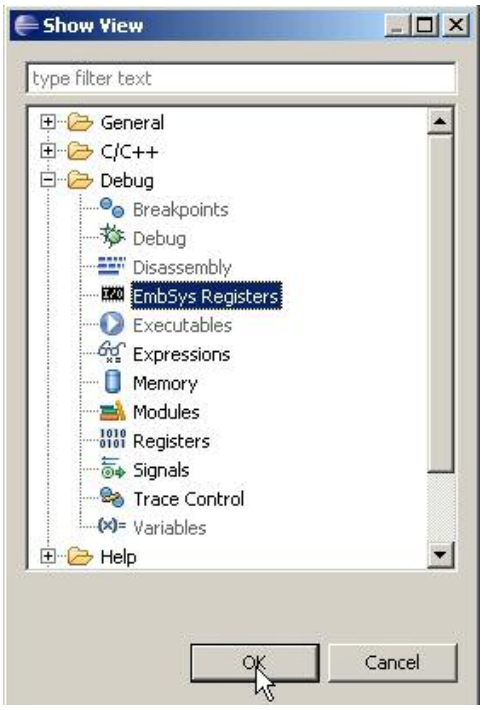


After Confirming the Register view configuration, the tool can be now used.

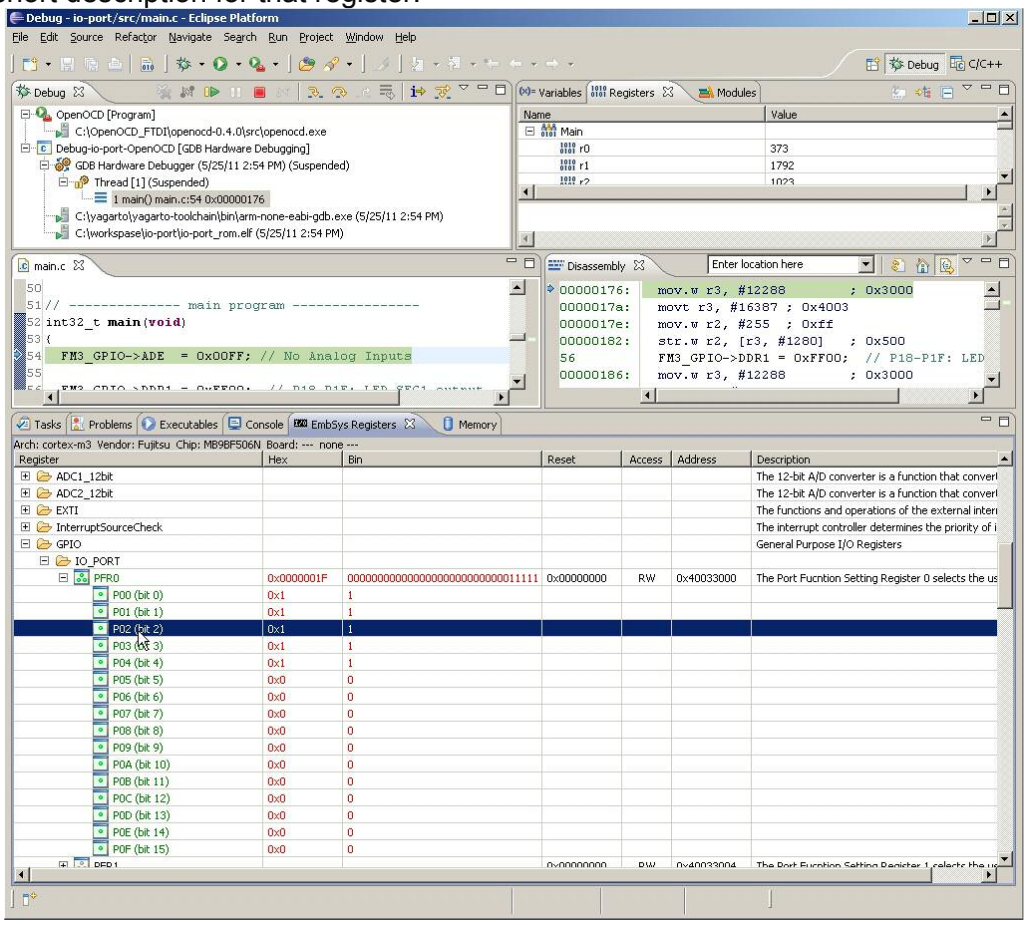
To open a register viewer in the CDT debug perspective (see chapter 12 for detailed information), select *Show View→Other…* in the Eclipse's *Window* pull-down menu.



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

HERE IS A SHORT COLLECTION OF OTHER ECLIPSE FEATURES GIVEN

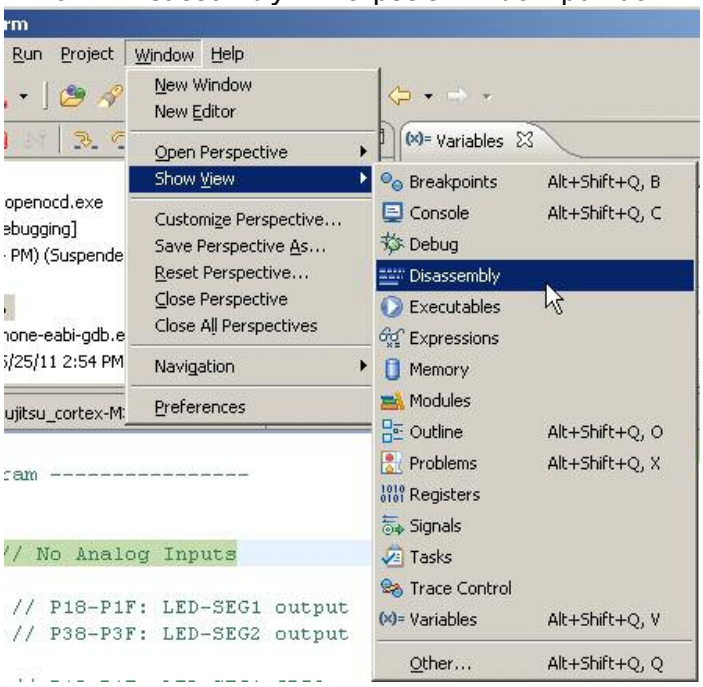
14.1 Overview

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 this features of the debug perspective are discussed.

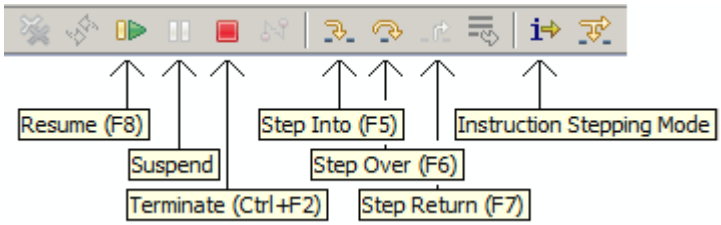
14.2 Disassembly view

To display the “Disassembly” view in the CDT debug perspective (see chapter 12 for details), select *Show View→Disassembly* in Eclipse's *Window* pull-down menu.



The view will be then displayed as shown below.

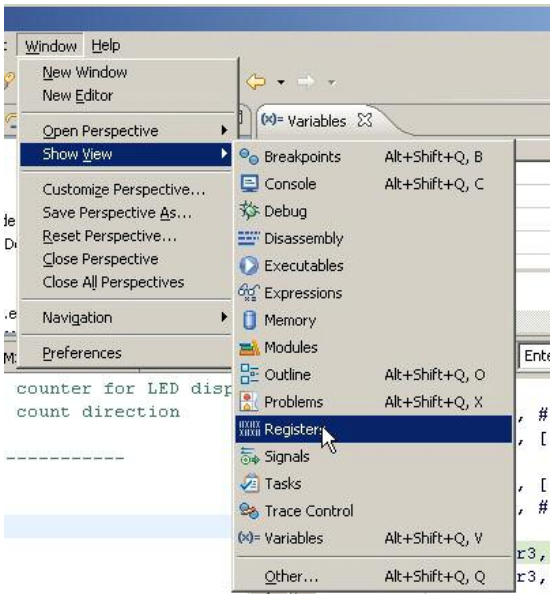
On this view a pointer to the current instruction will be set, so that the user can break the debugging process any time by clicking on the button *Suspend*. Do not mix it up with *Terminate*, which will end the debug session!



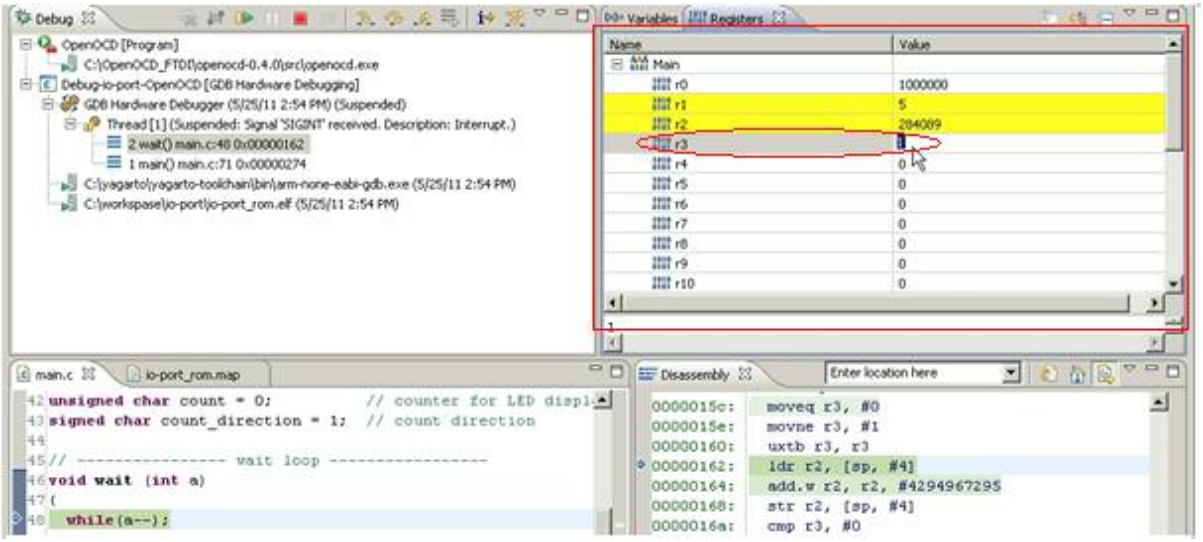
14.3 CPU Register View

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.



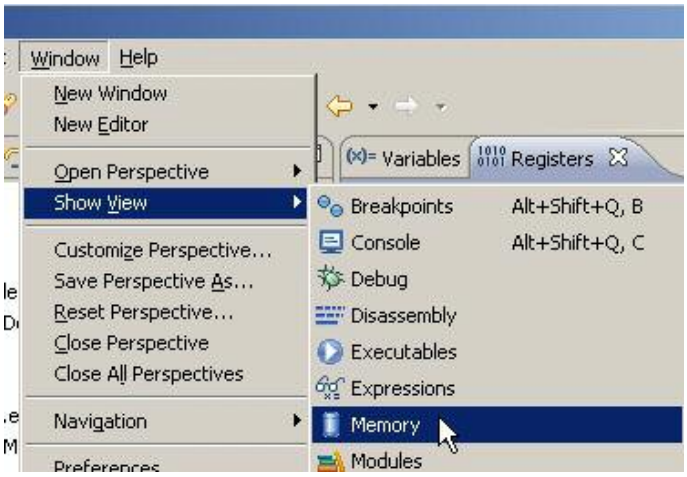
The selected view displays all core registers and their contents. Open the tree “Main” to get a CPU registers overview.



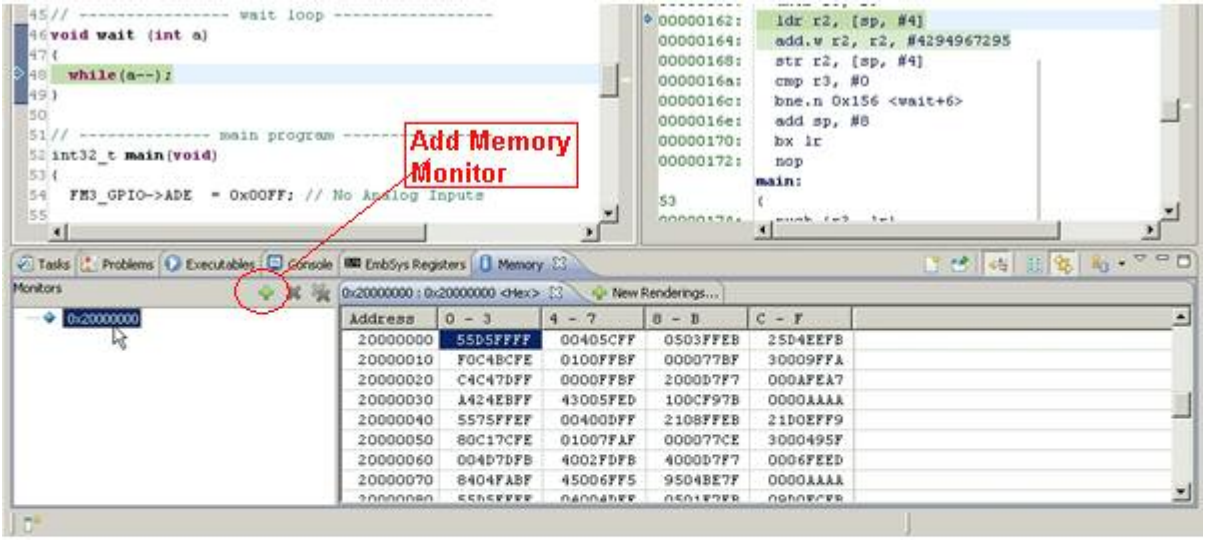
To edit the content of a register, select the register and double click on it.

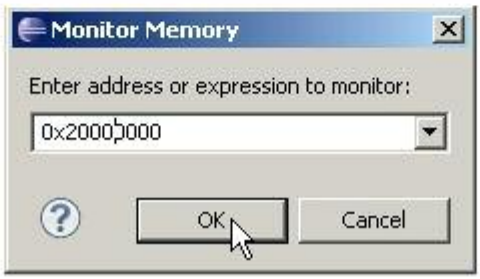
14.4 Memory view

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, clicki to the green plus sign in the Monitor pane.
The figure below shows the active memory monitors at address 0x20000000.

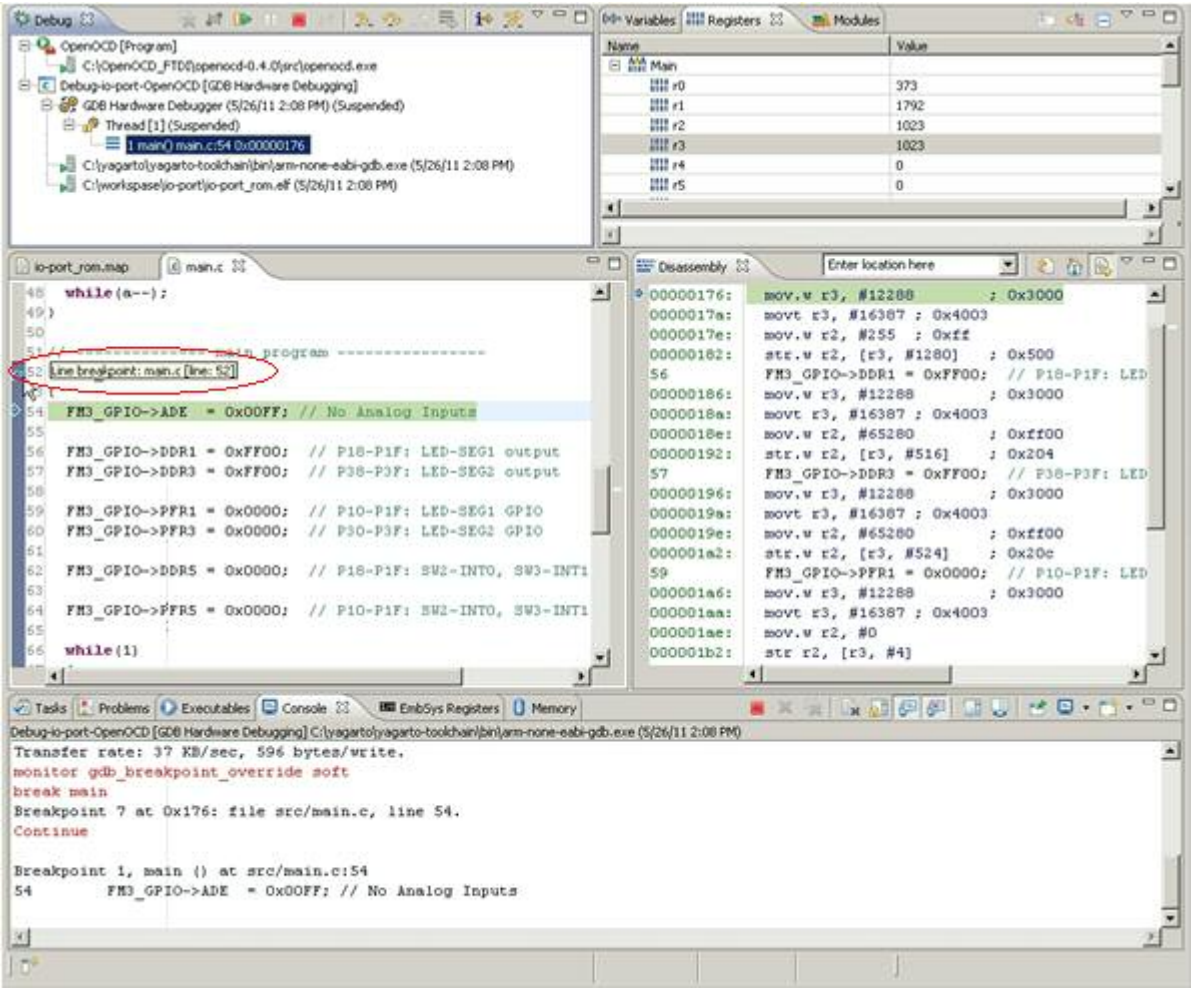




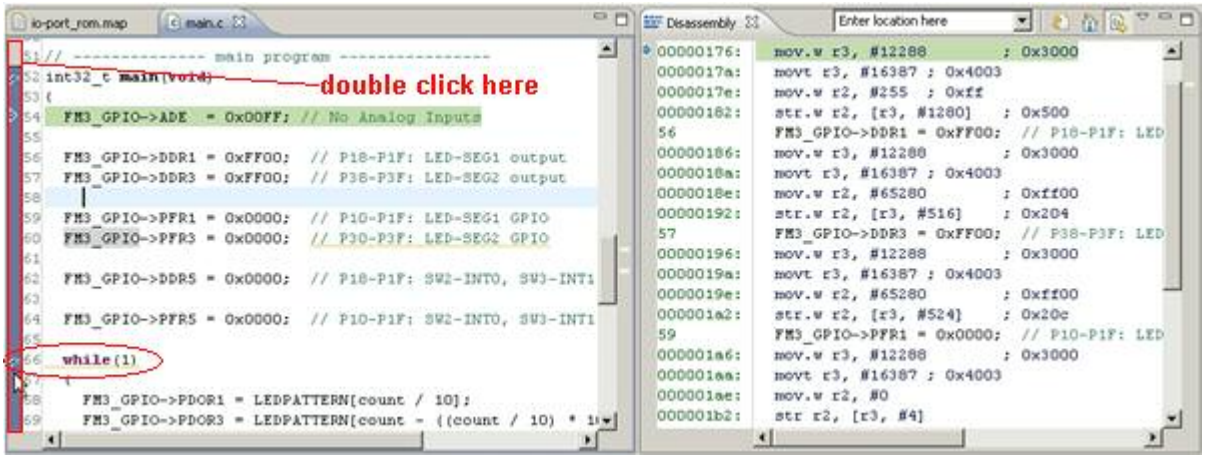
The content of a selected memory address (RAM and some I/O resources) can be edited and changed by double clicking on the respective address.

14.5 Using Breakpoints on Eclipse Debug Perspective

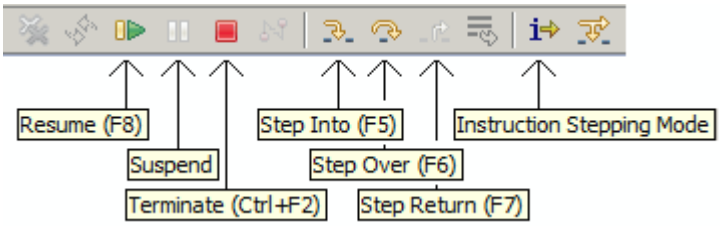
After starting a debug session, the debugger will set a breakpoint at the main function.



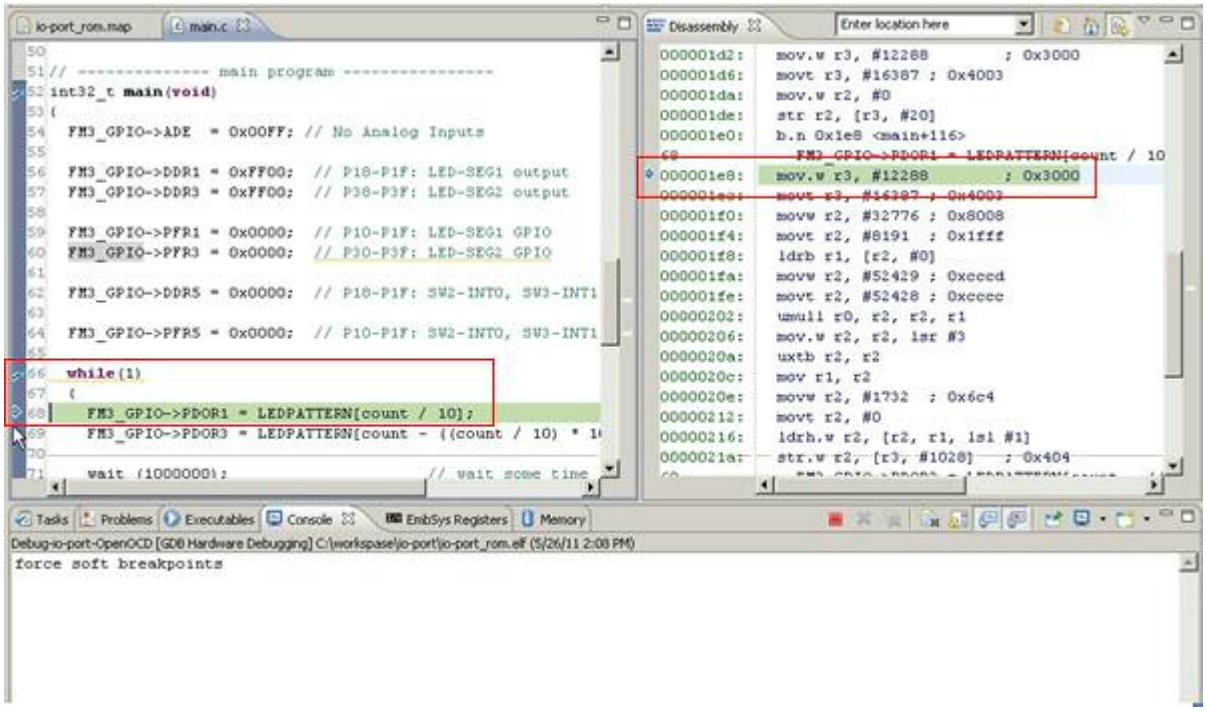
Other breakpoints can be set by double clicking in the left pane in the source code tab beside the line numers.



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
Cygwin	Cygnus Solutions for Windows	Compatibility layer for porting Unix/Linux based programs to Windows operating systems
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://prdownload.berlios/openocd/openocd-0.4.0.zip

FDTI driver:
www.ftdichip.com/D2XX.html

OpenOCD with LibUSB:
http://www.freddiechopin.info/index.php/en/download/category/4-openocd

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

Cygwin:
http://cygwin.com/install.html

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

Java JRE:
http://java.com/

15.2.2 Hardware and belonging Software (if needed)

J-Link from Segger:
http://www.segger.com/cms/jlink.html

KT-Link from Kris-Tech:
www.shop.kristech.eu

OpenOCD-USB (FTDI JTAG-to-USB adapter):
http://shop.embedded-projects.net/index.php?module=artikel&action=gruppe&id=16
→ OpenOCD USB Adapter

16 Additional Information

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

http://mcu.emea.fujitsu.com/

The software examples' and configuration files' archive related to this application note is:

Eclipse_configurations_software_package

It can be found on the following Internet page (attached to this application note):

http://mcu.emea.fujitsu.com/mcu_product/mcu_all_appnotes.htm

