

Tutorial: Arm Cross Development with Eclipse

/*Date: 04.03.2013 Version: 1.1 by Liwa Wu*/

1. Introduction

This tutorial will show you how to install Eclipse on Windows, how to setup, compile and debug a simple project using a MCB2100 board based on an ARM7TDMI-S microcontroller and Segger J-Link Debugger. All the software components used in this tutorial are free of charge.

2. Installing the Necessary Components

To set up an ARM cross-development environment using Eclipse, you need to download and install several components. The required parts of the Eclipse/ARM cross development system are:

1. Java Runtime
2. Eclipse CDT for C++/C Development
3. Java development plug-in
4. Minimalist GNU for Windows (MinGW)
5. Yagarto
6. Segger J-Link Debugger/Emulator for JTAG debugging

2.1 JAVA Runtime

As Eclipse is a Java application you must have Java Runtime Environment (JRE) installed on your computer before installing Eclipse. Open a command prompt and type:

```
C:\>java -version
```

The result should look like:

```
java version "1.7.0_10"  
Java(TM) SE Runtime Environment (build 1.7.0_10-b18)  
Java HotSpot(TM) Client VM (build 23.6-b04, mixed mode, sharing)
```

If the Java Runtime Environment (JRE) is not installed on your PC, you can download the JRE from the following Oracle website:

[Java SE Downloads](#) (about 30 MB)

Here you will find the JRE download button. Download and install it.

2.2 Eclipse CDT

Once you have Java installed, head over to <http://eclipse.org/>. The following Eclipse welcome page will display. Expect some differences from my example below since the Eclipse web site is very dynamic.

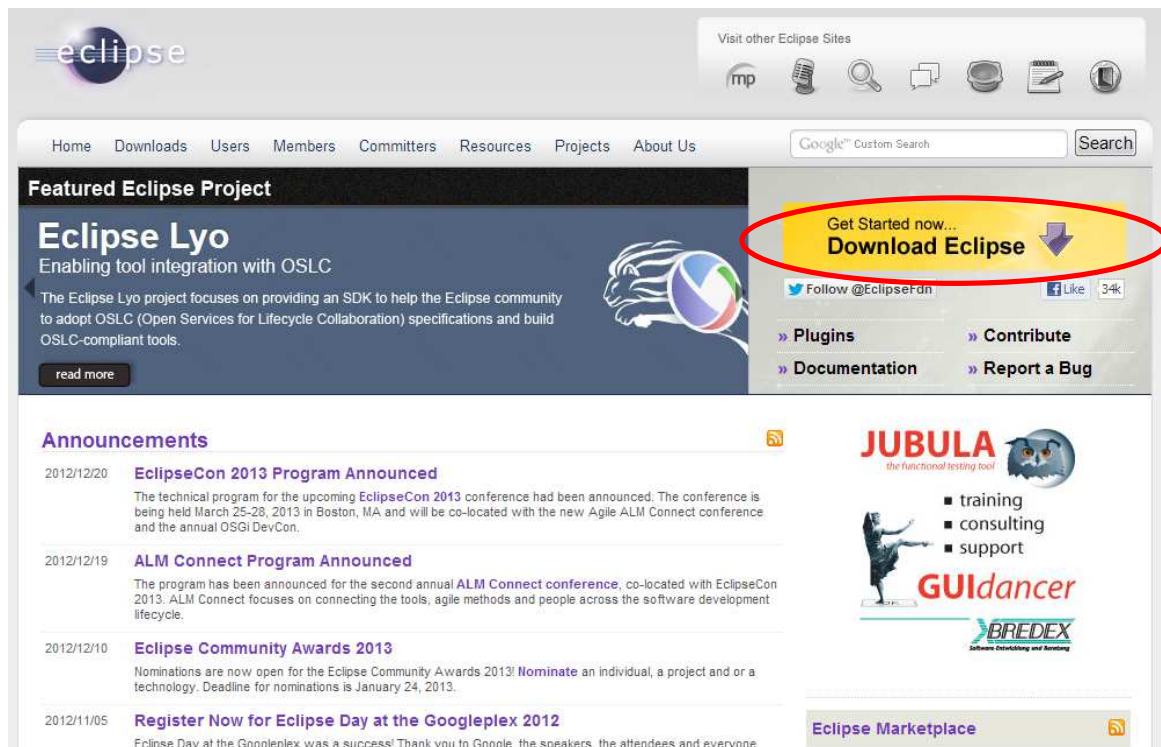


Fig. 2.1: Display of the eclipse welcome website

Click on **Download Eclipse** to get things started.

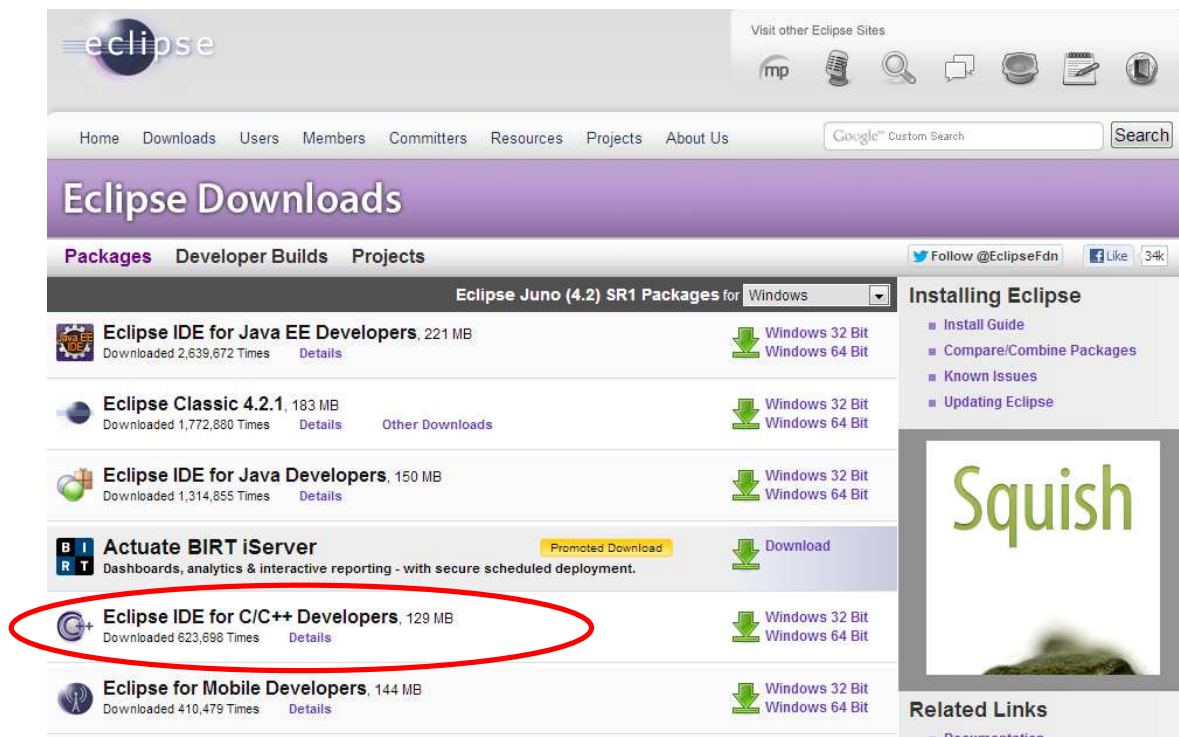


Fig. 2.2: Display of the eclipse downloads. Select of the Eclipse IDE for C/C++ developers

Click on “Eclipse IDE for C/C++ Developers” to setup the Eclipse CDT

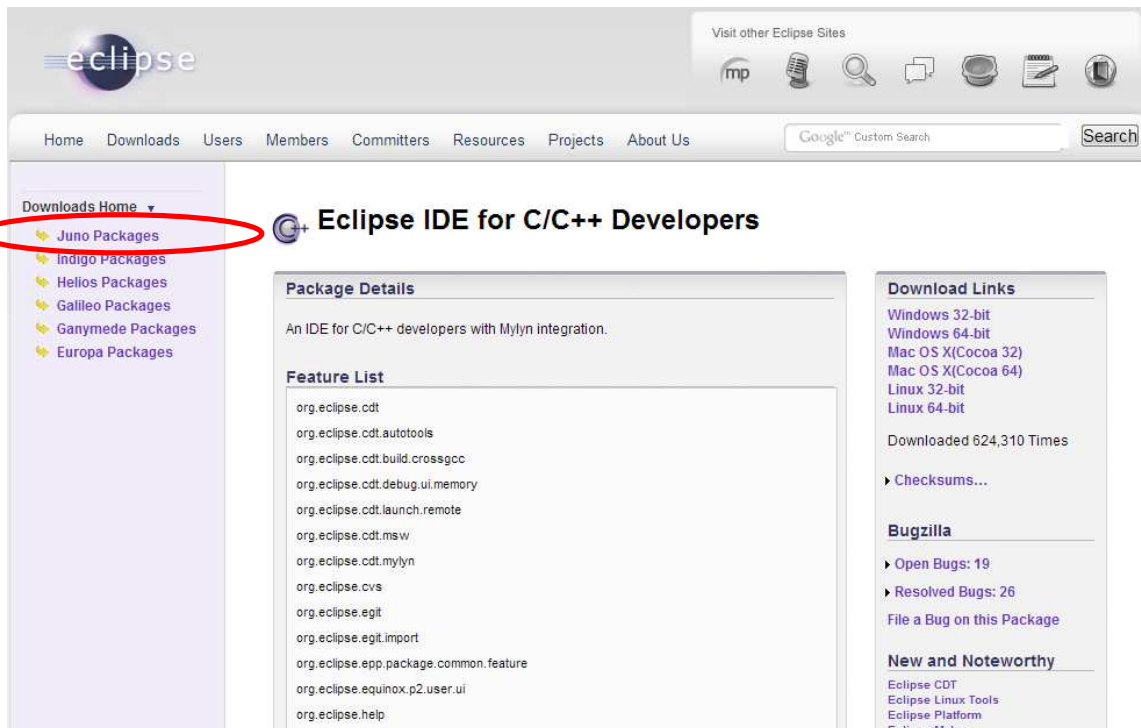


Fig. 2.3: Display of the eclipse IDE for C/C++ developers. Select the Indigo packages.

At the left edge of the window, we can see that there are several different packages refers to different versions, here “Indigo Package” is selected.

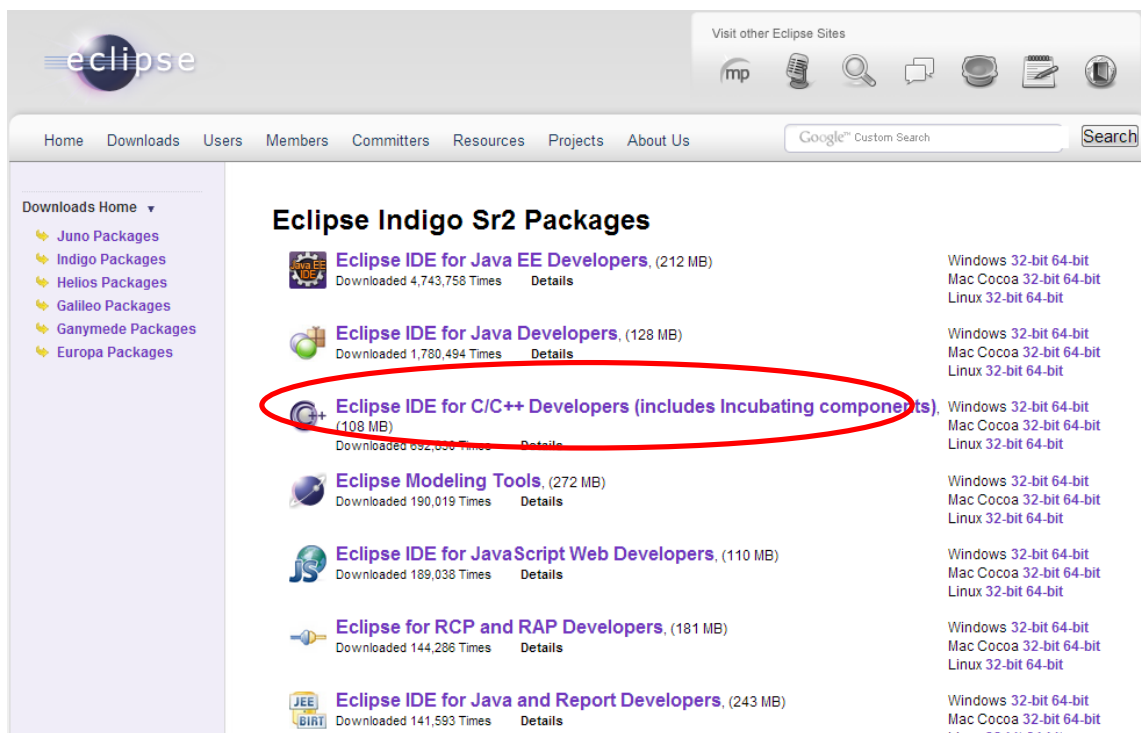


Fig. 2.4: Display of the eclipse indigo packages download. Select the eclipse IDE for C/C++ developers.

Select the “Eclipse IDE for C/C++ Developers” and then choose the suitable version, here “Windows

32-bit” is selected.

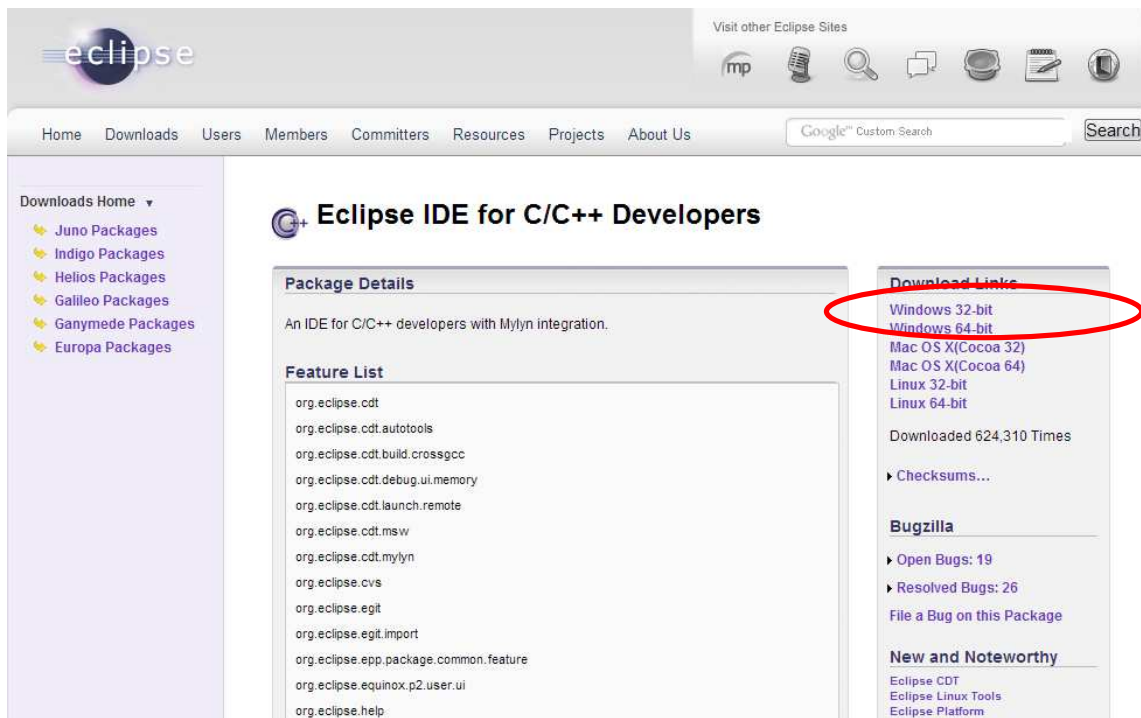


Fig. 2.5: Display of the eclipse IDE for C/C++ developers under the indigo packages. Select the windows 32-bit version.

What appears next is a list of download mirror sites that host the Eclipse components.

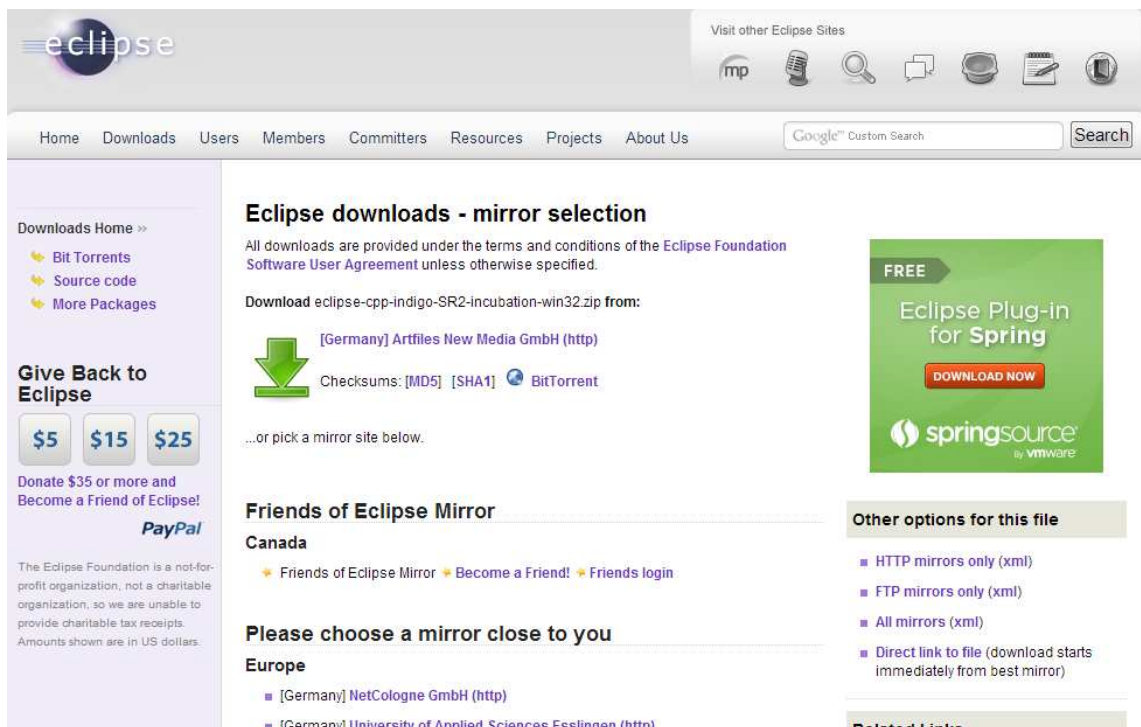


Fig. 2.6: Layout of the eclipse downloads – mirror selection.

Select one and then download the Eclipse zip file.

Once Eclipse is downloaded follow the procedure provided below:

- Extract the “eclipse” folder from the downloaded zipped Eclipse classic file; which in my case was called “eclipse-cpp-indigo-SR2-incubation-win32.zip” into the “D:\” directory such that the full path of the eclipse directory is: “D:\eclipse”. Inside the eclipse directory you will find the eclipse binary (.exe) file. Click on it.
- A workspace launcher window will appear asking you of the location of the eclipse workspace.

I choose to place the workspace within the Eclipse directory, namely “D:\workspace”. You are free to place this anywhere; you can have multiple workspaces; here is where you make that choice.

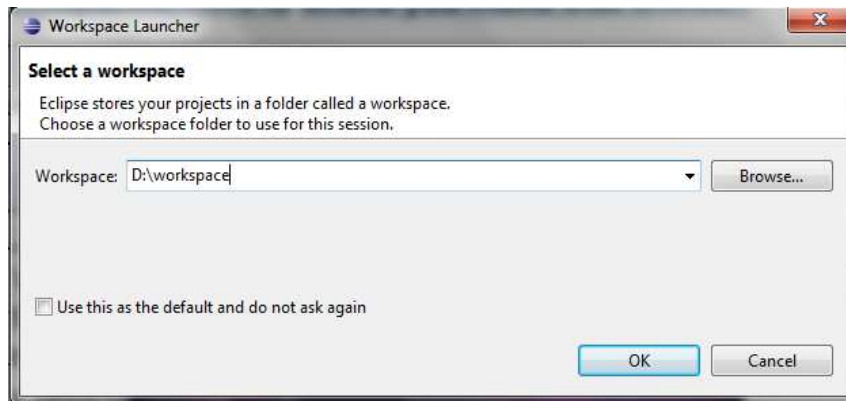


Fig. 2.7: Dialog box of the setting of the workspace.

When you click **OK**, the Eclipse main screen will start up.

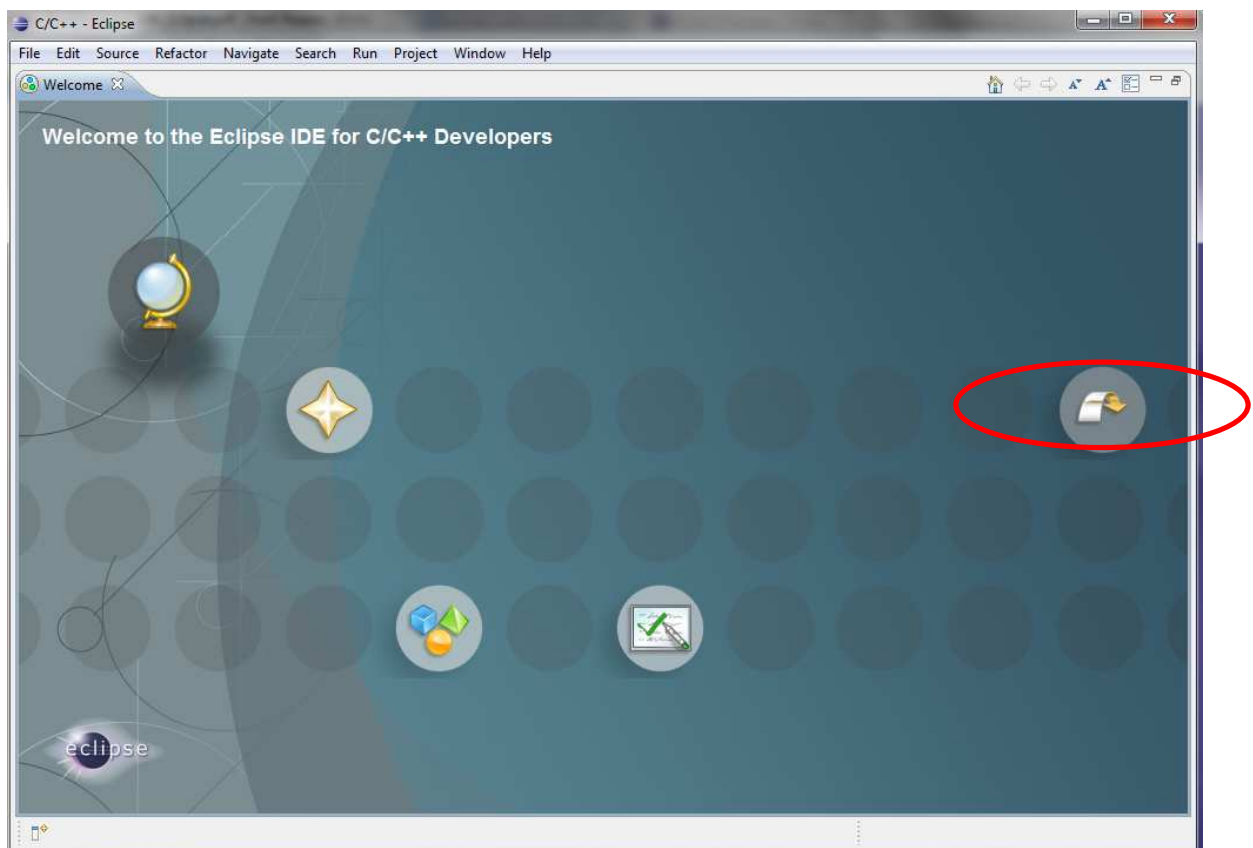


Fig. 2.8: Welcome page of the eclipse Indigo for C/C++ developers

If you made it this far, you now have a complete Eclipse system capable of developing C/C++ programs.

Now click at the **WorkBench** icon at the right edge of the window.

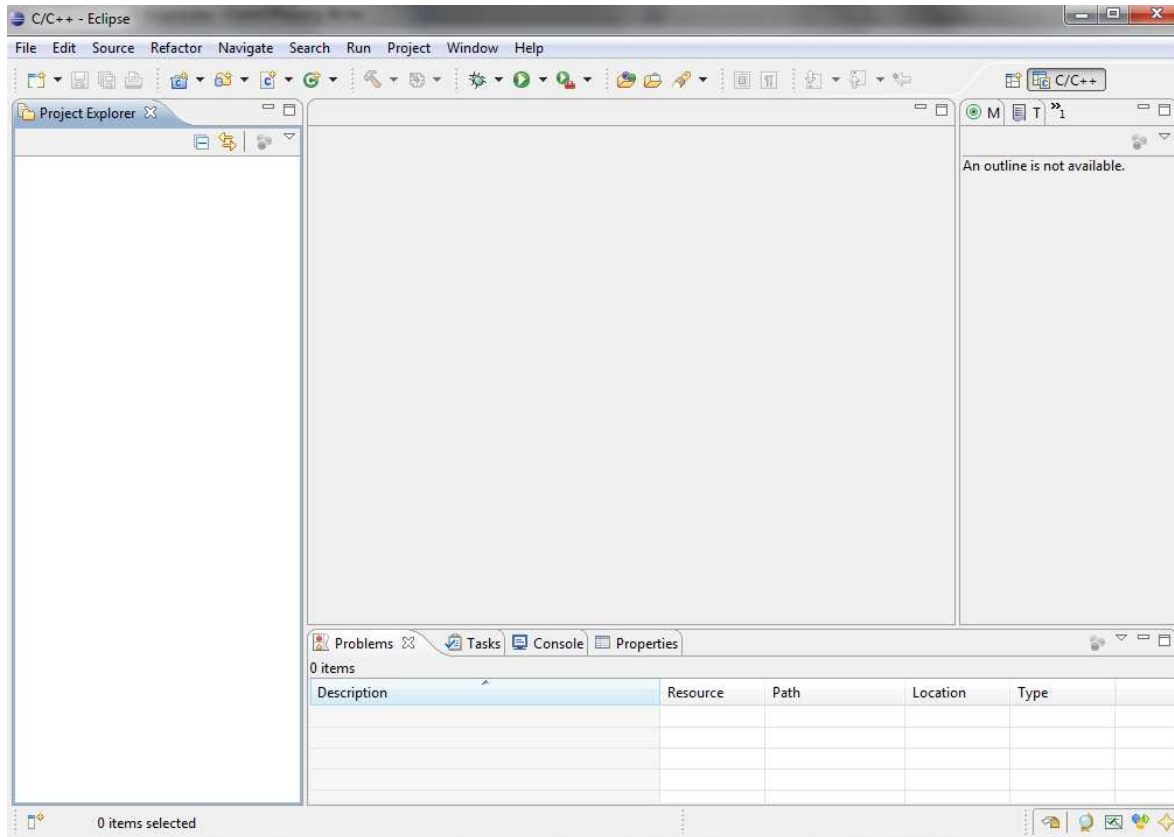


Fig. 2.9: Layout of the eclipse Indigo

2.2.1 Plug-ins

Eclipse provides a powerful and easy to use mechanism to install additional features. This is a three step process:

- To add the update site providing the plug-in (tell Eclipse where to find the software).
- To select the plug-in to install.
- To start the installation.

To install new components open the **Install** dialog by clicking the **Install New Software** command from the *Help* menu.

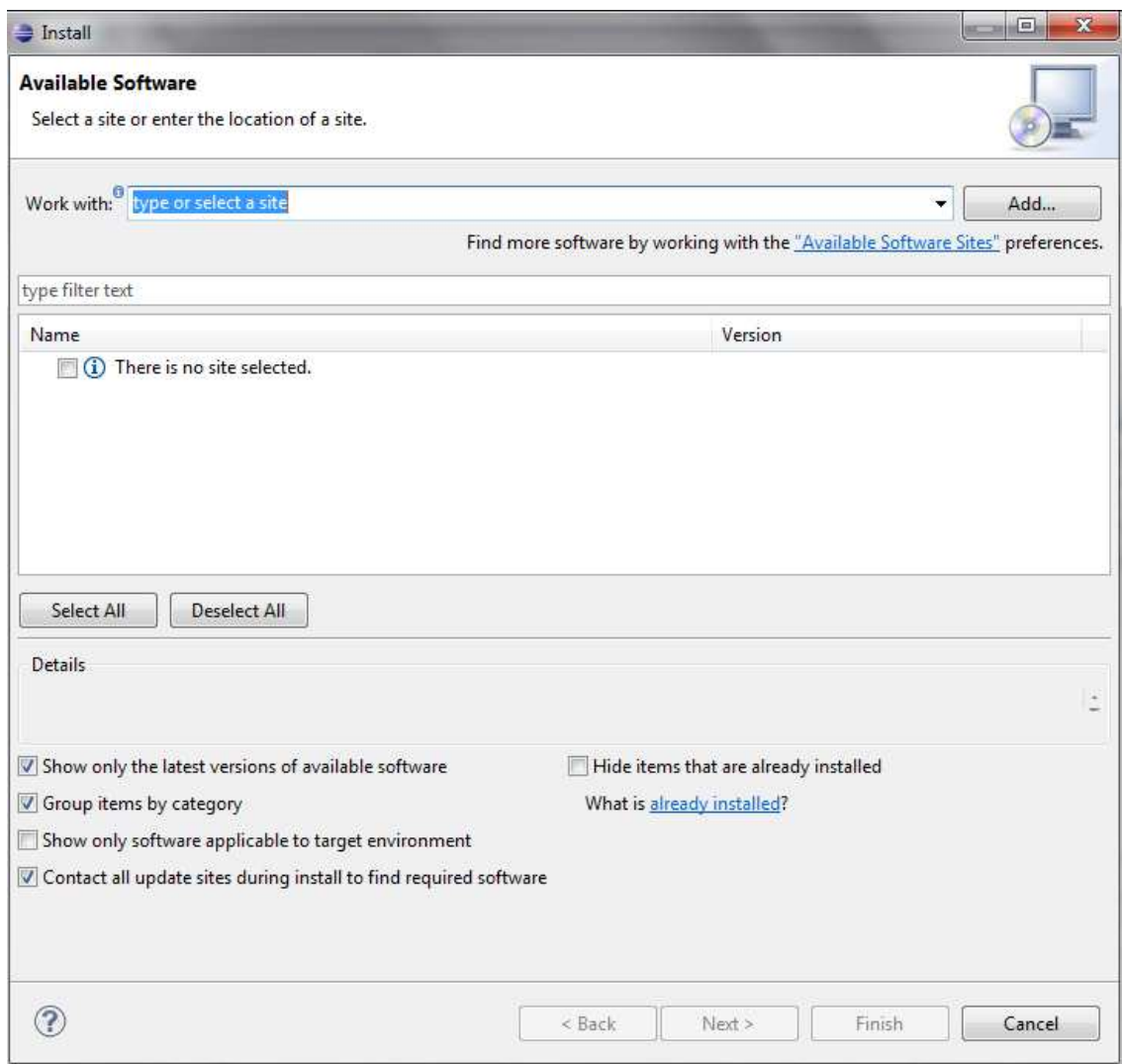


Fig. 2.10: Dialog of the install new software

2.2.1.1 Install GDB Hardware Debugging Plug-in

- Select the **--All Available Sites--** in the **Work with** field.
- Type “GDB Hardware Debugging” and press **Enter** or left click mouse.
- After a while the **Install** dialog displays the list of the available Plug-ins.
- Select **C/C++ GDB Hardware Debugging** and press **Next** as shown in the Fig. 2.11.
- Follow the onscreen instruction, don’t worry about (but read ;-) the warning, and restart Eclipse when required to complete the installation.

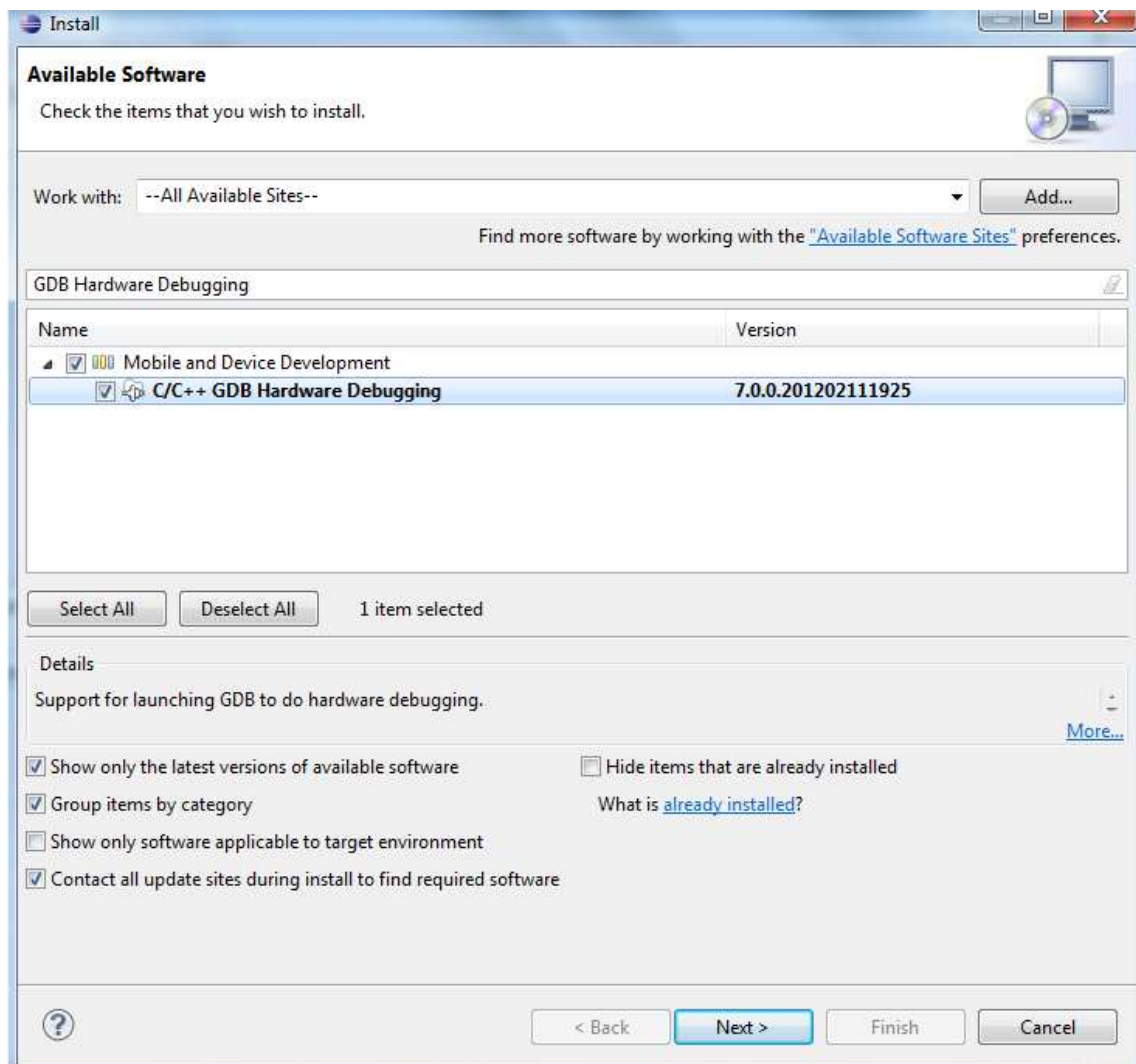


Fig. 2.11: Dialog of install the GDB Hardware Debugging plug-in

2.2.1.2 GNU ARM Eclipse Plug-in

This plug-in automate the Makefile generation and management.

- Click the **Add** button on the top right of the **Install** dialog to open the **Add Repository** dialog.
- Type **GNU ARM Eclipse Plug-in** in the **Name** field (this is only a name, so choose the one you prefer).
- Insert the following web address in the **Location** field:
<http://sourceforge.net/projects/gnuarmeclipse/files/Eclipse/updates/>

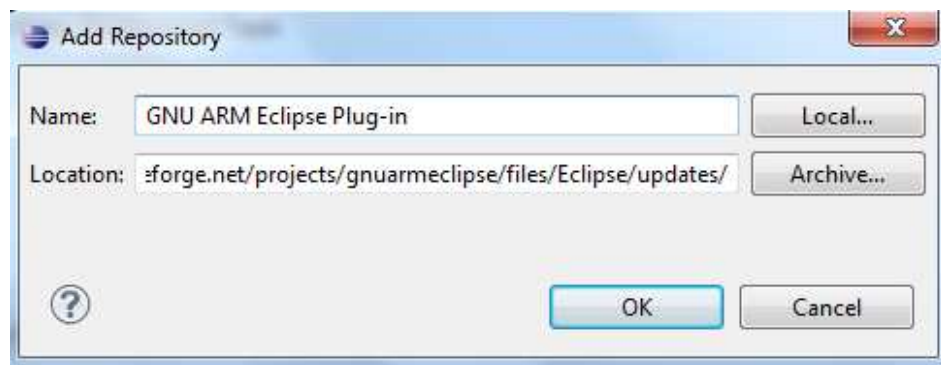


Fig. 2.12: Dialog of add the GNU ARM Eclipse Plug-in

- Press the **OK** button to confirm your choice.
- After a while the **Install** dialog displays the list of the available Plug-ins.
- Select **CDT GNU Cross Development Tools** and press **Next**.

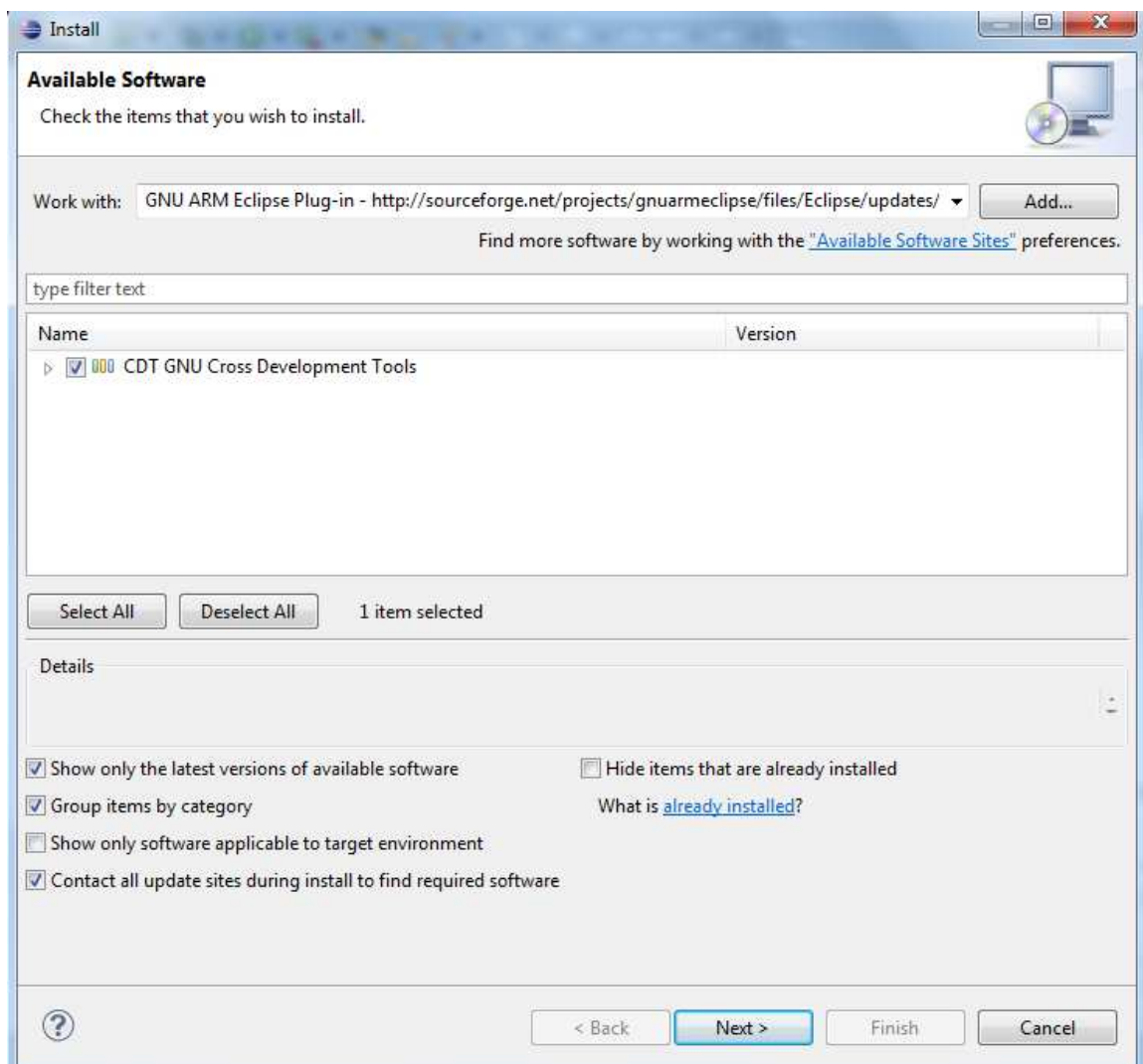


Fig. 2.13: Dialog of install the GNU ARM Eclipse Plug-in

- Follow the onscreen instruction, don't worry about (but read ;-) the warning, and restart Eclipse

when required to complete the installation.

2.3 Install IDE for Java Development

To get Eclipse ready for java development, we need to add the java development tools.

- From the *Help* menu, select **Install New Software**.
- Choose “Indigo - <http://download.eclipse.org/releases/indigo>”.
- Under **Programming Languages**, select “Eclipse Java Development Tools”
- Click **Next**.
- Follow the onscreen instruction, and restart Eclipse when required to complete the installation.

2.4 Install MinGW for Native Microsoft Windows Applications

Open the folder “Y:\public Studenten\Langen\Grundlagen Informatik I\Entwicklungsumgebungen”. Double click the “MinGW-3.1.0-1.exe”, follow the onscreen instruction and then setup the MinGW.

2.5 Yagarto Tools and Yagarto GNU ARM Toolchain

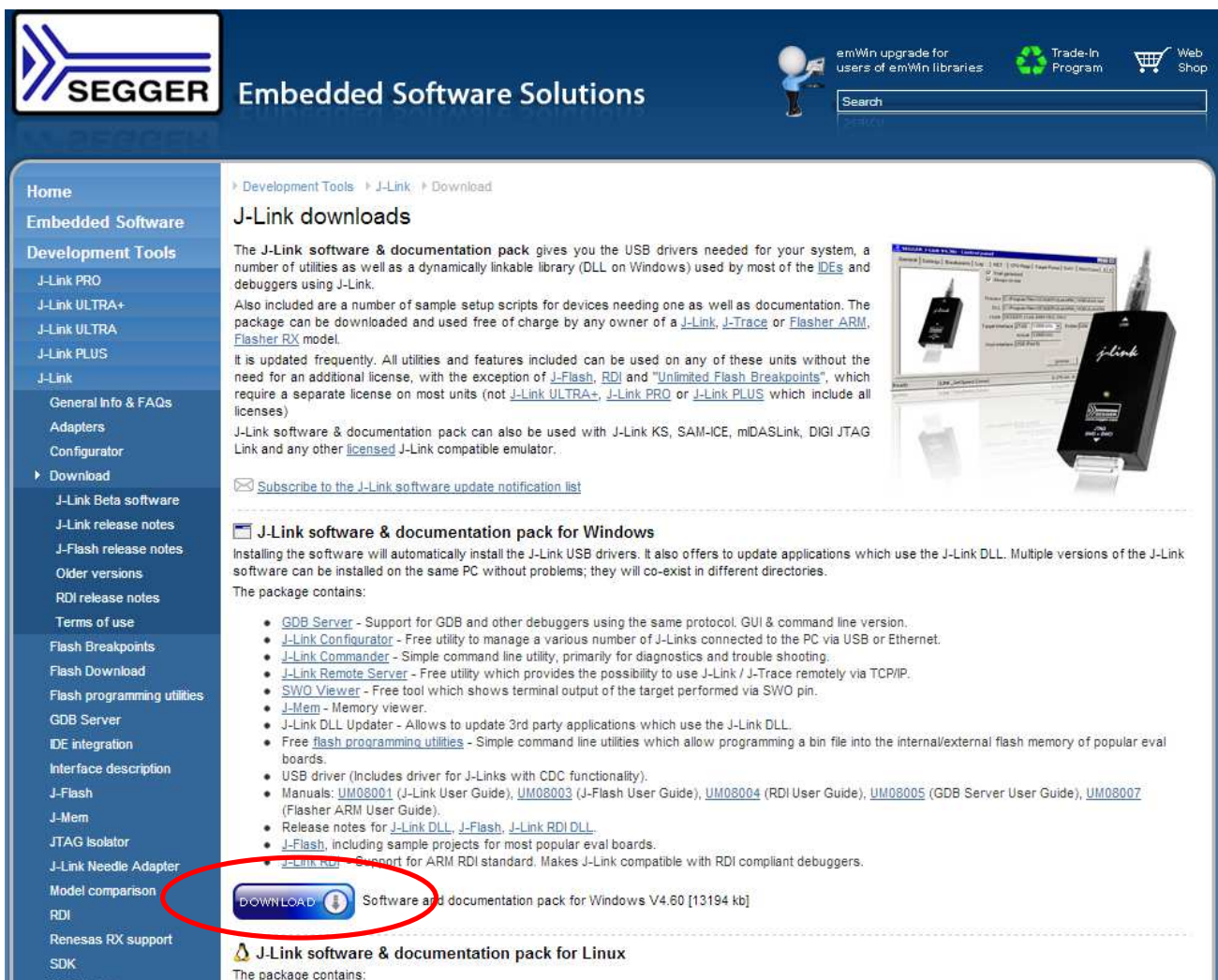
Download the two packages namely YAGARTO Tools and YAGARTO GNU ARM toolchain respectively from <http://www.yagarto.org/>.

If you are the Windows user, please download the two packages namely “YAGARTO Tools” and “YAGARTO GNU ARM toolchain”. Download them and then follow the onscreen instruction and select all the default settings click **Next** until **Finish**.

If you are the Mac OS X user, only the “YAGARTO GNU ARM toolchain” package is needed. Download it and then follow the onscreen instruction and select all the default settings click **Next** until **Finish**.

2.6 Segger J-Link Debugger/Emulator

To use the debugger, download and install the J-Link drivers from the SEGGER website (<http://www.segger.com/jlink-software.html>). Download the proper package depends on the different systems. For the Windows user, click the **DOWNLOAD** button as shown in the Fig. 2.14.



SEGGER Embedded Software Solutions

emWin upgrade for users of emWin libraries | Trade-In Program | Web Shop

Search

Home

Embedded Software

Development Tools

J-Link PRO

J-Link ULTRA+

J-Link ULTRA

J-Link PLUS

J-Link

General Info & FAQs

Adapters

Configurator

Download

J-Link Beta software

J-Link release notes

J-Flash release notes

Older versions

RDI release notes

Terms of use

Flash Breakpoints

Flash Download

Flash programming utilities

GDB Server

IDE integration

Interface description

J-Flash

J-Mem

JTAG Isolator

J-Link Needle Adapter

Model comparison

RDI

Renesas RX support

SDK

Development Tools > J-Link > Download

J-Link downloads

The J-Link software & documentation pack gives you the USB drivers needed for your system, a number of utilities as well as a dynamically linkable library (DLL on Windows) used by most of the IDEs and debuggers using J-Link.

Also included are a number of sample setup scripts for devices needing one as well as documentation. The package can be downloaded and used free of charge by any owner of a J-Link, J-Trace or Flasher ARM, Flasher RX model.

It is updated frequently. All utilities and features included can be used on any of these units without the need for an additional license, with the exception of J-Flash, RDI and "Unlimited Flash Breakpoints", which require a separate license on most units (not J-Link ULTRA+, J-Link PRO or J-Link PLUS which include all licenses)

J-Link software & documentation pack can also be used with J-Link KS, SAM-ICE, mDASLink, DIGI JTAG Link and any other licensed J-Link compatible emulator.

[Subscribe to the J-Link software update notification list](#)

J-Link software & documentation pack for Windows

Installing the software will automatically install the J-Link USB drivers. It also offers to update applications which use the J-Link DLL. Multiple versions of the J-Link software can be installed on the same PC without problems; they will co-exist in different directories.

The package contains:

- [GDB Server](#) - Support for GDB and other debuggers using the same protocol. GUI & command line version.
- [J-Link Configurator](#) - Free utility to manage a various number of J-Links connected to the PC via USB or Ethernet.
- [J-Link Commander](#) - Simple command line utility, primarily for diagnostics and trouble shooting.
- [J-Link Remote Server](#) - Free utility which provides the possibility to use J-Link / J-Trace remotely via TCP/IP.
- [SWO Viewer](#) - Free tool which shows terminal output of the target performed via SWO pin.
- [J-Mem](#) - Memory viewer.
- [J-Link DLL Updater](#) - Allows to update 3rd party applications which use the J-Link DLL.
- [Free flash programming utilities](#) - Simple command line utilities which allow programming a bin file into the internal/external flash memory of popular eval boards.
- [USB driver](#) (Includes driver for J-Links with CDC functionality).
- [Manuals](#): [UM08001](#) (J-Link User Guide), [UM08003](#) (J-Flash User Guide), [UM08004](#) (RDI User Guide), [UM08005](#) (GDB Server User Guide), [UM08007](#) (Flasher ARM User Guide).
- [Release notes for J-Link DLL, J-Flash, J-Link RDI DLL](#).
- [J-Flash](#), including sample projects for most popular eval boards.
- [ARMv7-M](#) - Support for ARM RDI standard. Makes J-Link compatible with RDI compliant debuggers.

DOWNLOAD Software and documentation pack for Windows V4.60 [13194 kb]

J-Link software & documentation pack for Linux

The package contains:

Fig. 2.14: J-Link downloads for Windows user

Here you have to enter the serial number of the emulator, which can be found on the bottom side of the J-Link. Enter the serial number and then download the zip file "Setup_JLinkARM_V460.zip". It is important that before installing the software, please connect the board to your PC via the supplied USB cable and the J-Link device. Unzip the downloaded file and then follow the onscreen instruction and select all the default settings click **Next** until **Finish**.

Now let's check whether the J-Link device driver was properly installed or not.

- From the *Start* menu , in **All Programs** → **SEGGER** → **J-Link ARM V4.56d** (that was the current version when I created this tutorial)
- Launch the program called **J-Link GDB Server via JTAG**.
- The program starts and the GUI should look something like:

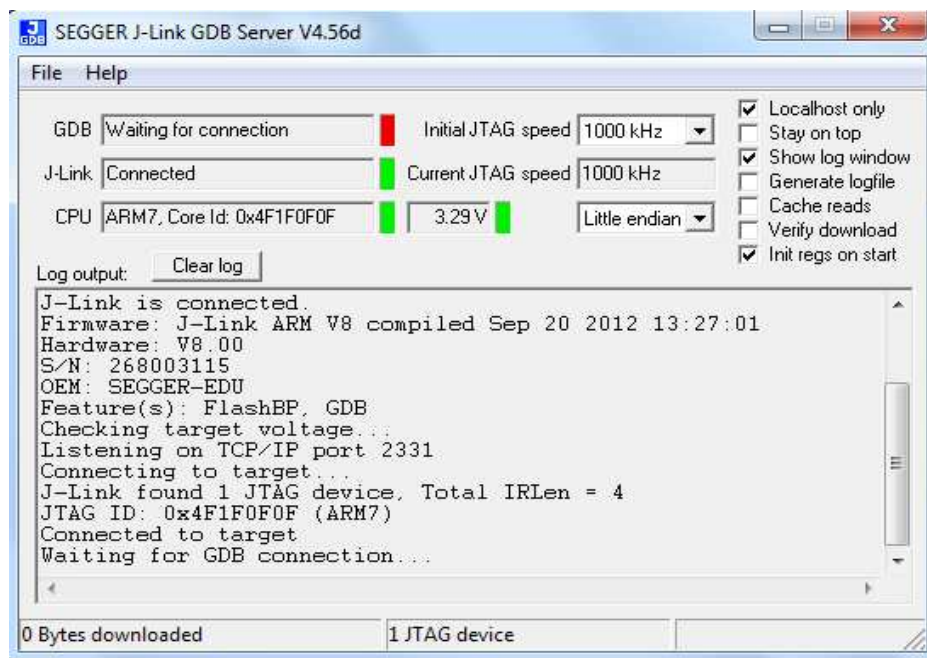


Fig. 2.15: J-Link GDB Server GUI

Fig. 2.15 shows that SEGGER GDB server is ready and connected to the device.
Note: Do not exit this program before continuing.

3. Working with Eclipse

3.1 Customize Eclipse

At this point, you can make any tweaks to the IDE that you like. There are two in particular that I prefer. I like to turn on line numbering in the text editor by going to **Window -> Preferences -> General -> Editors -> Text Editors** and checking “Show line numbers.”

The second change that I prefer is to have the IDE automatically save any changed files before building when I build a project. (There’s nothing more frustrating than troubleshooting a bug you thought you’d just fixed, only to find out that your edits weren’t saved before the file was compiled.) This can be done in **Window -> Preferences -> General -> Workspace**, check “Save automatically before build.” I also like to uncheck “Build automatically” here.

Note that these are per-workspace settings. If you change your workspace you’ll likely have to edit these settings again.

3.2 Import an Existing Project

The easiest way to get up and running with a project of your own is to start with someone else’s work and build upon it.

From the **File** menu, click **Import** and a dialog will be shown. In the treeview, expand **General** and highlight the **Existing Project into Workspace**.

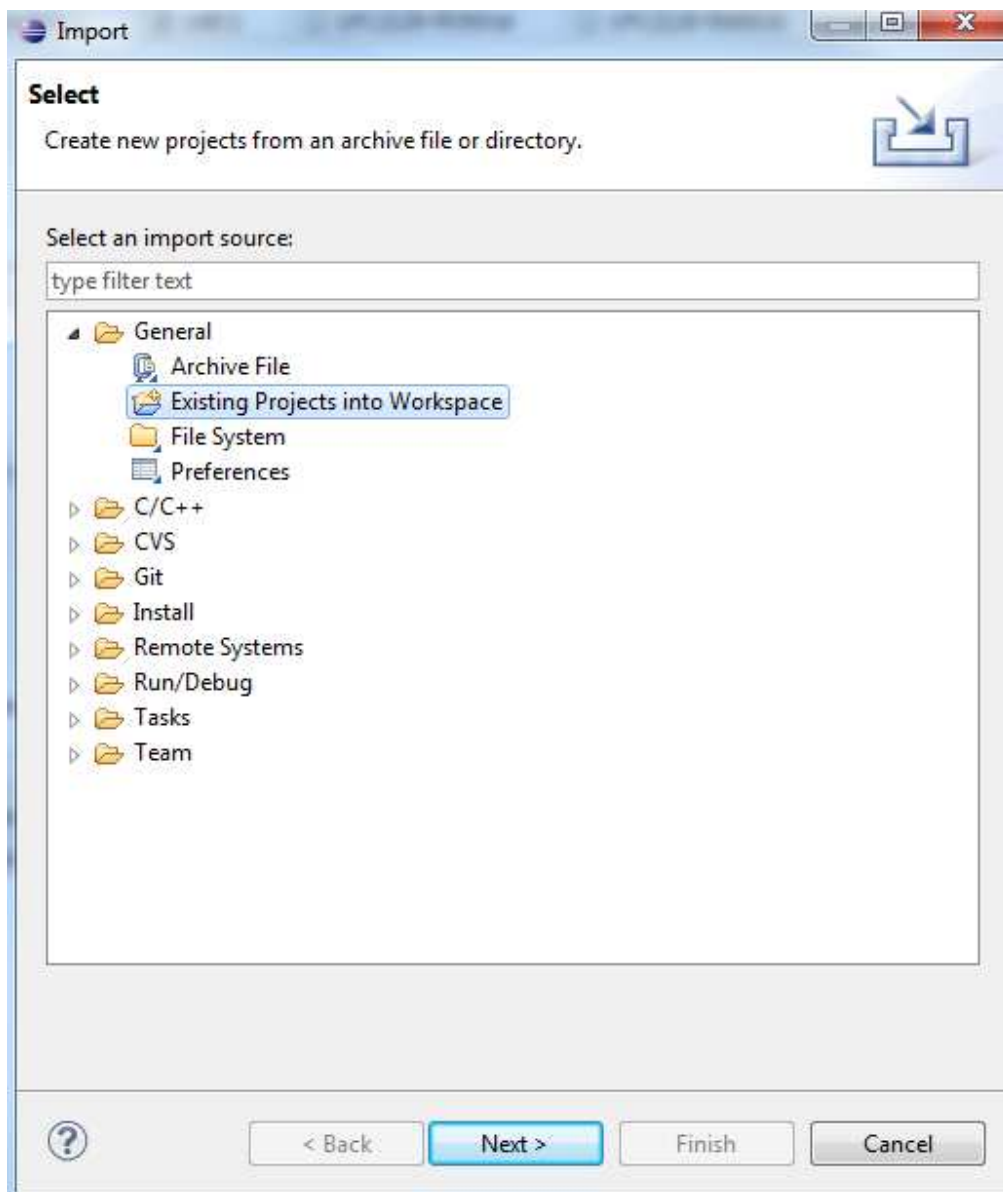


Fig. 3.1: Dialog of import action

Click **Next** and select the proper root directory of the project you want to imported. Here the “MCB2100_Blinky_IRQ” project is selected as shown in the Fig. 3.2. Make sure that the “Copy projects into workspace” is selected.

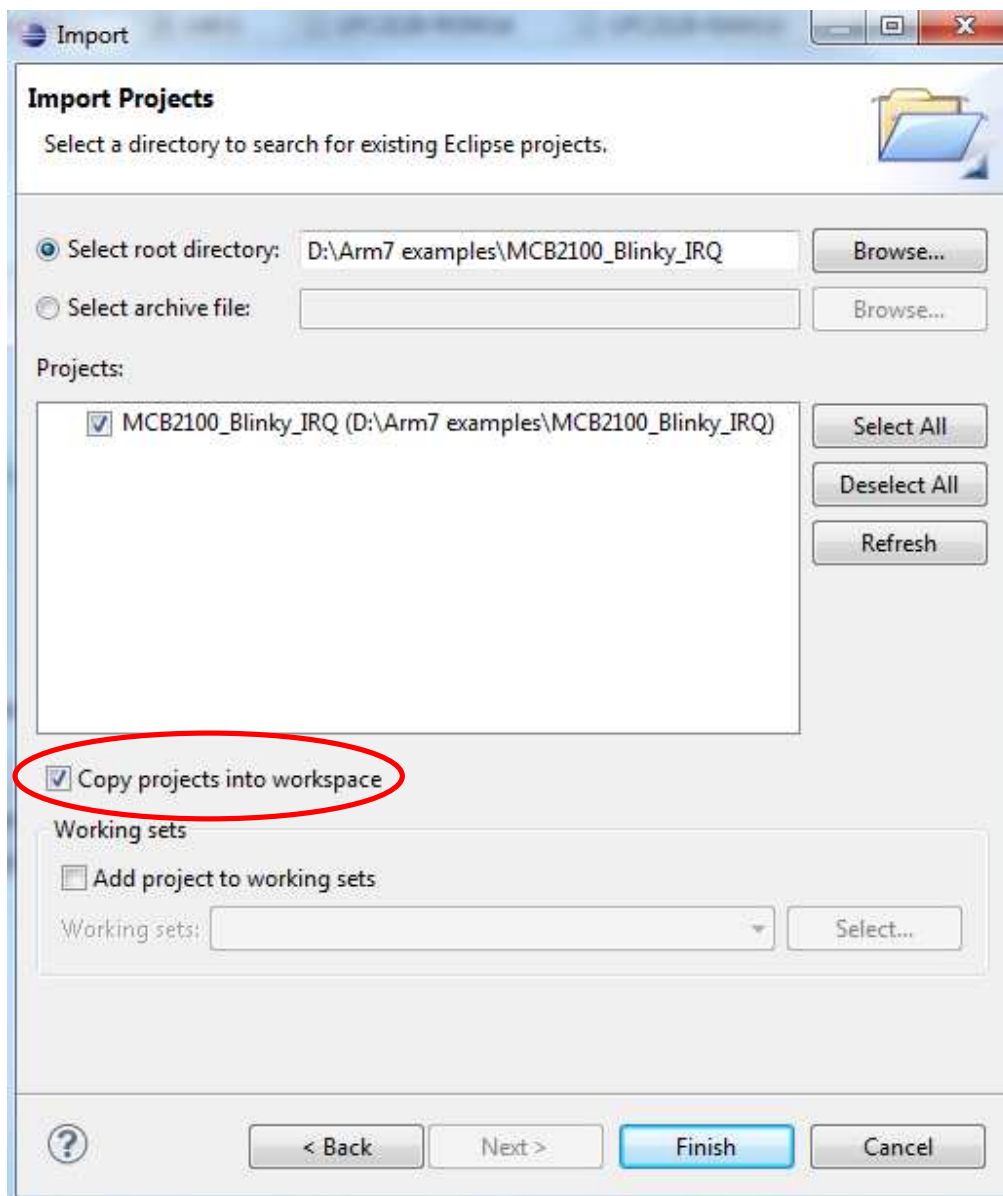


Fig. 3.2: Dialog of import projects

When you click **Finish**, the project will be imported and you can see it in the **Project Explorer** on the left side of your screen. Highlight **MCB2100_Blinky_IRQ**, the whole project contents will be shown.

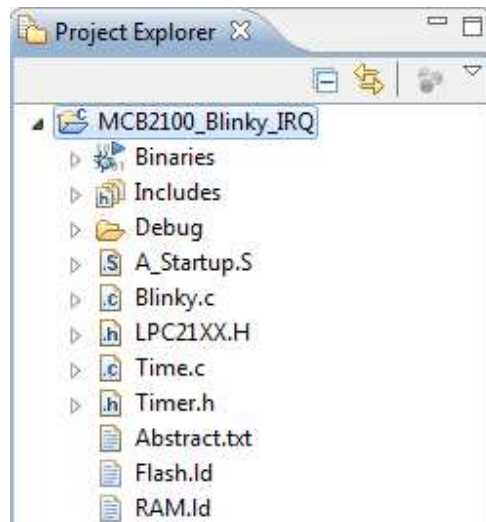


Fig. 3.3: Project structure in the project explorer

3.3 Configure the GCC Toolchain in Eclipse and Build the Project

As an exercise, let's attempt to build our project. Ensure that the **MCB2100_Blinky_IRQ** folder is selected in **Project Explorer**, and from the *Project* menu, choose "Build Project." (You may have to first uncheck the "Build Automatically" option in this menu.)

If the project builds successfully (no error happens), then skip this part, directly jump to Chapter 3.4.

Most likely, your project didn't build. Check the **Problems** tab for details:

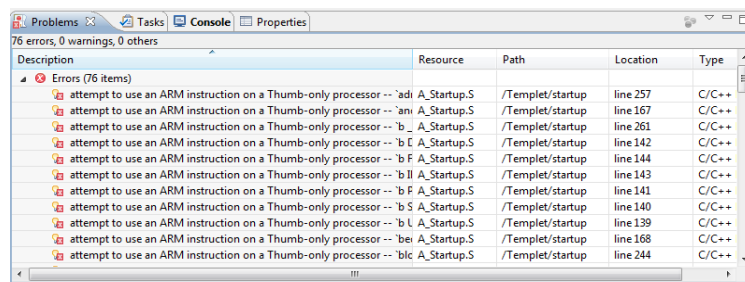


Fig. 3.4: Error messages in the Problems window.

Eclipse has given us a pretty clear error message here, that an ARM instruction can't be used on a Thumb-only processor.

From the *Project* menu, select **Properties**. In the treeview on the left, select **C/C++ Build-> Settings**. In the **Tool Settings** tab, select **Target Processor** and on the right side, select the proper processor, here the **arm7tdmi-s** is selected for our board. Select the **Thumb interwork** checkbox instead of the **Thumb** checkbox.

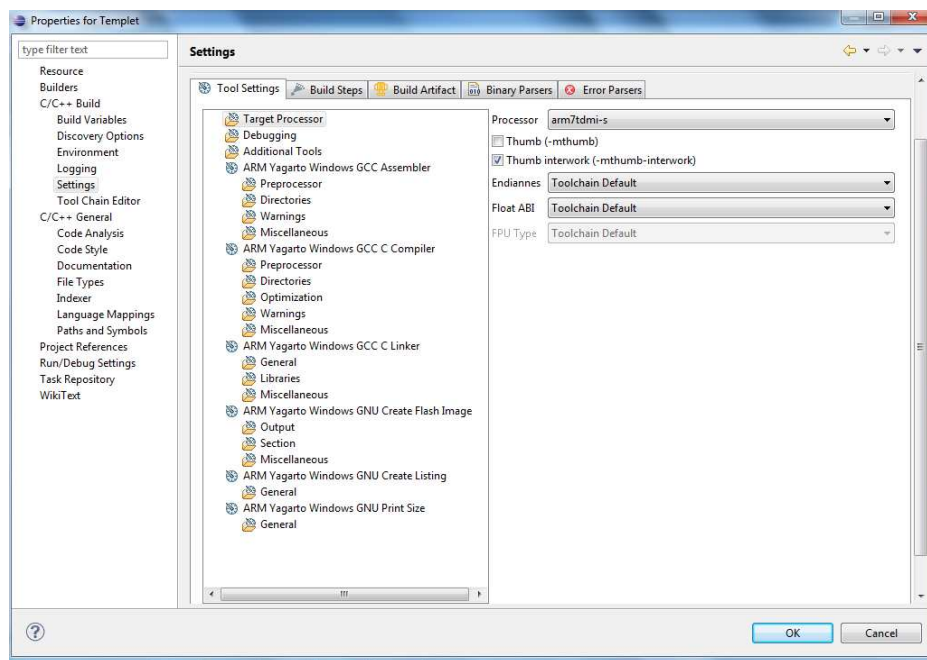


Fig. 3.5: Dialog of setting the processor.

Click **OK** and build the project again. (Highlight the project in the **Project Explorer**, from the *Project* menu, select **Build Project**.) Note that this time we get a different error as shown in the Fig. 3.6.

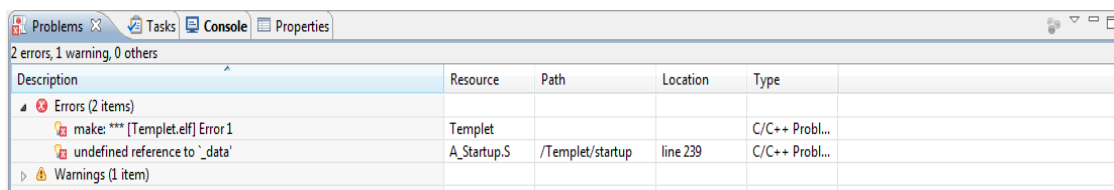


Fig. 3.6: Error messages after setting the processor.

Microcontrollers have a very limited amount of ROM and RAM. In addition, peripherals are mapped into the same address space as memory. Certain pieces of code, such as the reset vector and interrupt vectors, need to be placed at predefined locations in memory. In order to ensure that all the parts of our compiled project land in the correct spots in the memory map, the linker needs a resource, called a linker script that tells it how to arrange the memory map.

From the *Project* menu, select **Properties**. In the tree view in the left pane, expand **C/C++ Build**, and highlight the **Settings** node. Ensure the **Tool Settings** tab is selected. Click on **General** in the **ARM Windows GCC Linker** list on the left. In the **Script file** field, type the “D:\workspace\MCB2100_Blinky_IRQ\ RAM.ld” or click the **Browse** button, find the RAM.ld file under the project root and click **Open** to add it. Make sure to select the checkbox of “Do not use standard start files (-nostartfiles)” and then click **OK**.

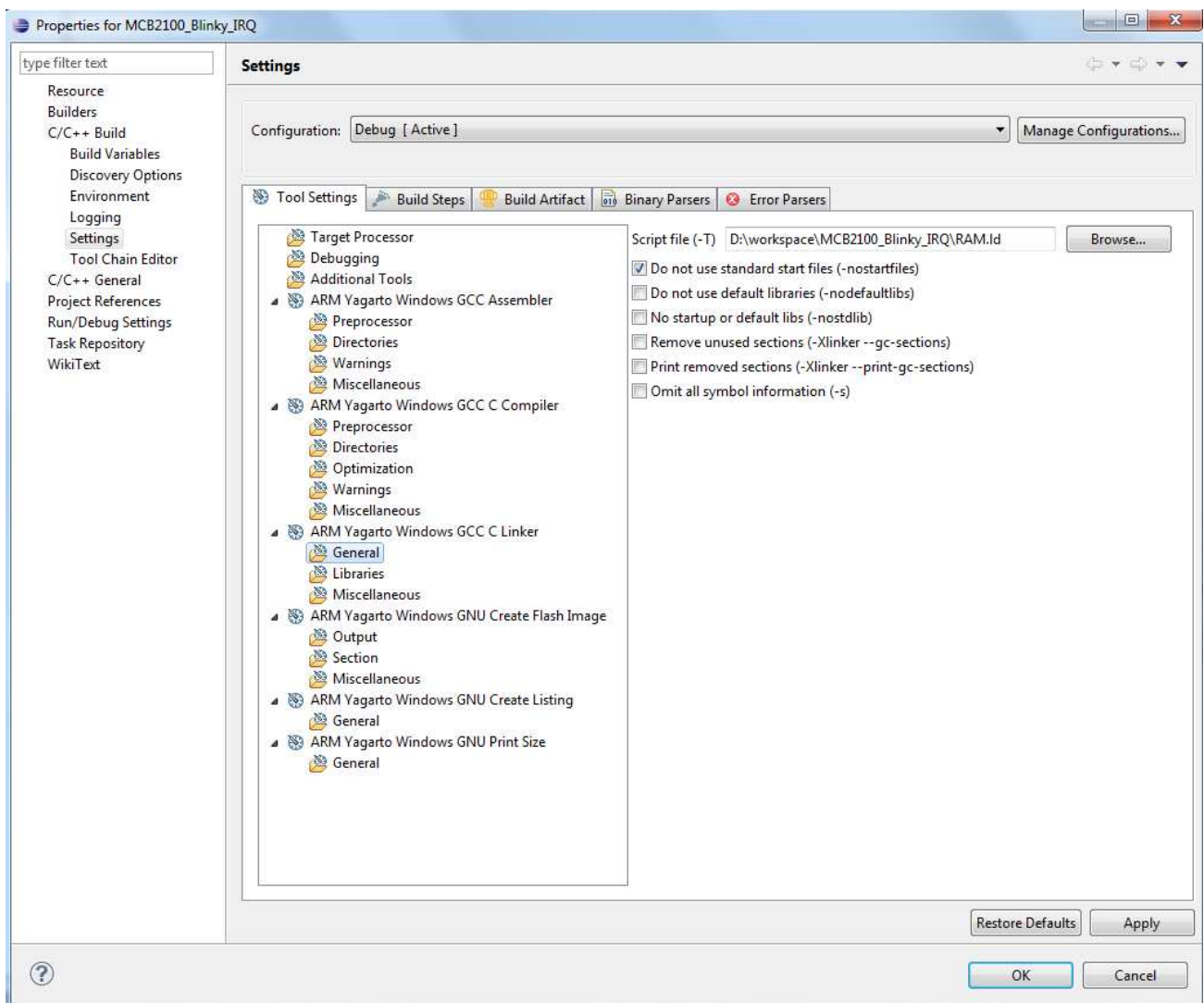


Fig. 3.7: Dialog to set the script file.

Now we can build our project. Ensure that the **MCB2100_Blinky_IRQ** folder is selected in **Project Explorer**, and from the *Project* menu, choose **Build Project**. When the build is finished, the output should be similar to the following in **Console**.

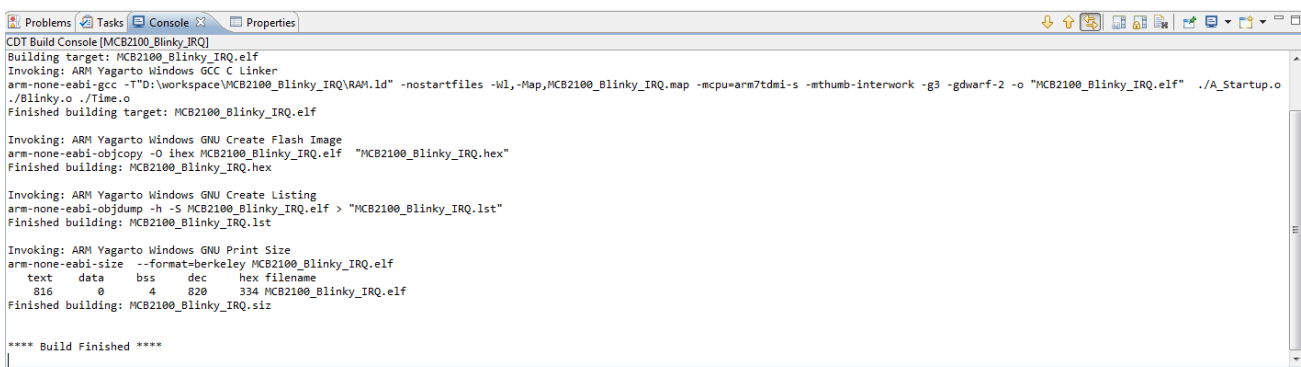


Fig. 3.8: Output of the build finished in console.

After the build has been completed successfully, you have a “MCB2100_Blinky_IRQ.elf” file in the

Debug folder of the project. We need to download this to the target and execute it. Here we need the Segger debugger.

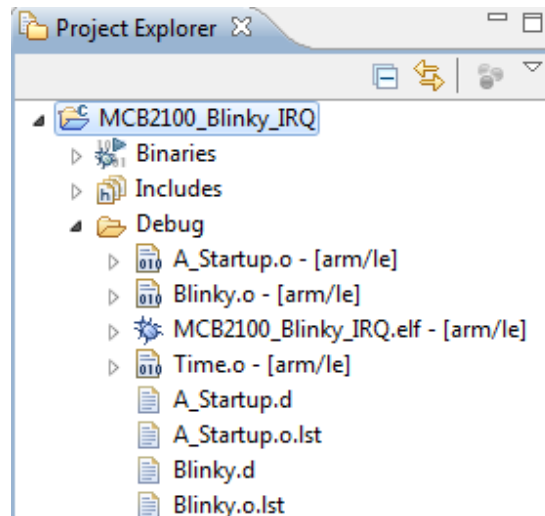


Fig. 3.9: The debug file is generated automatically after the build

3.4 Setup a GDB Debug Configuration for the J-Link GDB Server

As mentioned in Chapter 2.6, the GDB Server is started. The GDB configuration is a special Eclipse CDT configuration dedicated to debugging an application.

1. select *Eclipse* menu, **Run** → **Debug Configurations...**
2. select the **GDB Hardware Debugging**
3. click the **New** button or double click the **GDB Hardware Debugging** to create a new configuration
4. select the **Main** tab
5. check if the C/C++ Application was properly selected (it should look like Debug/MCB2100_Blinky_IRQ.elf)
6. check if the project was properly selected (it should look like MCB2100_Blinky_IRQ) as shown in the Fig. 3.10

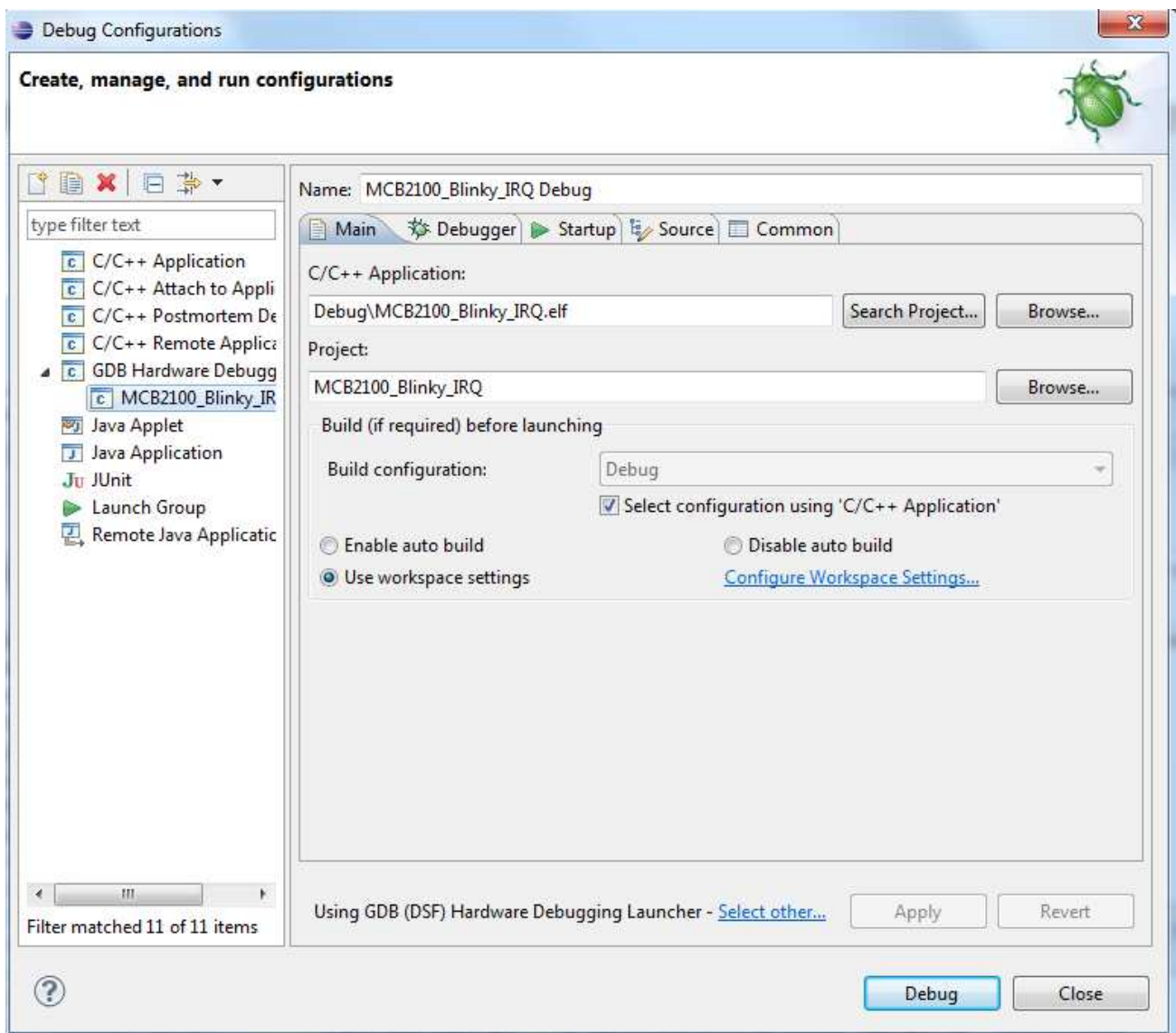


Fig. 3.10: Dialog of debug configuration

7. check the bottom line of the window if the **Standard GDB Hardware Debugging Launcher** is selected; if not, change the setting, eventually in the workspace scope

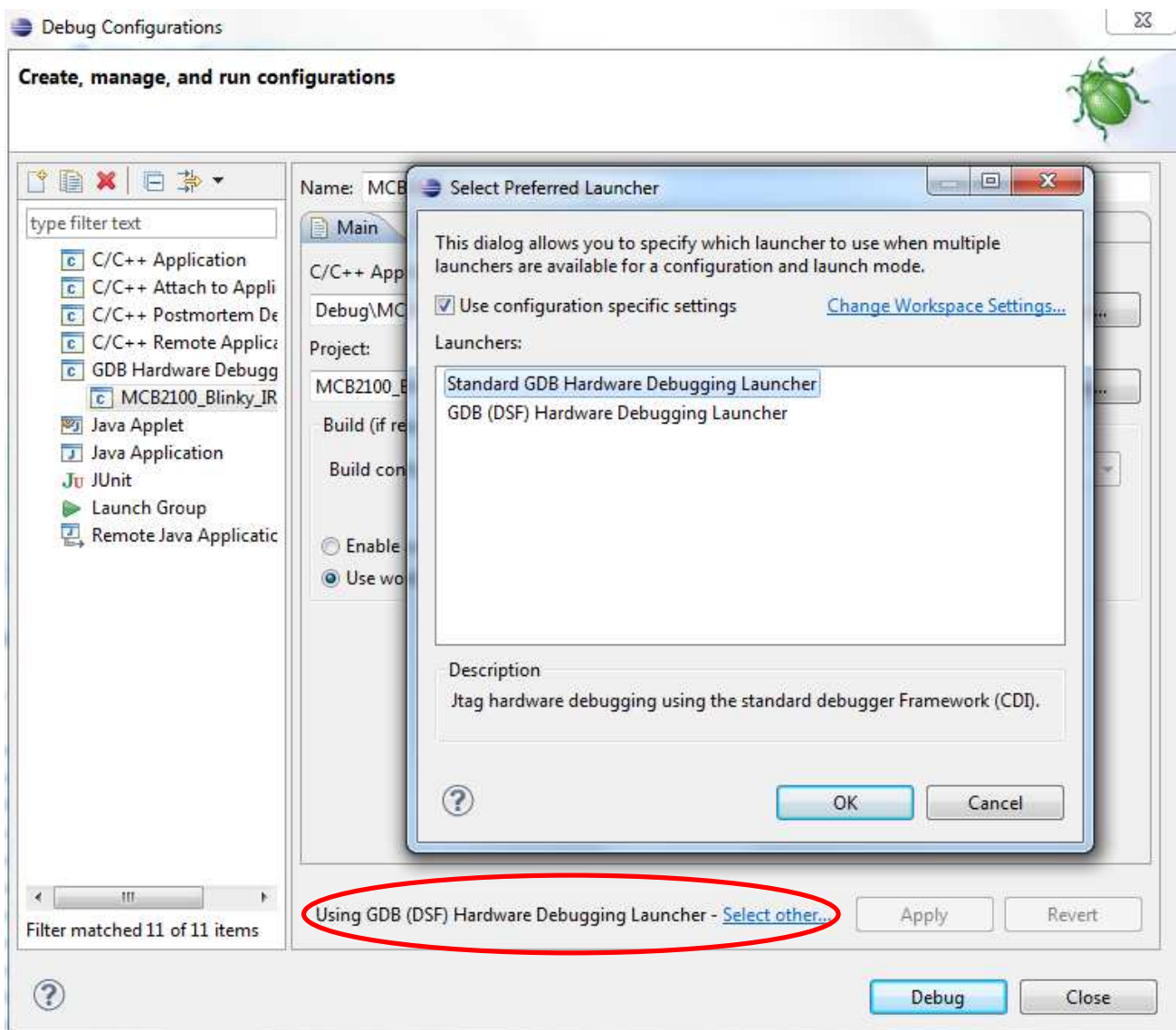


Fig. 3.11: Dialog of select the preferred hardware debugging launcher

8. select the **Debugger** tab
9. in the GDB Command, enter the full path of the gdb program (like C:\yagarto-20121222\bin\arm-none-eabi-gdb.exe)
10. select the **Standard** command set
11. select the **mi** protocol
12. enable **Use remote target**
13. select **Generic TCP/IP JTAG** device
14. select the host name or IP address where the GDB Server is running (127.0.0.1 or localhost when the server is running on the same machine)
15. select the port number (2331 as default for J-Link GDB Server)

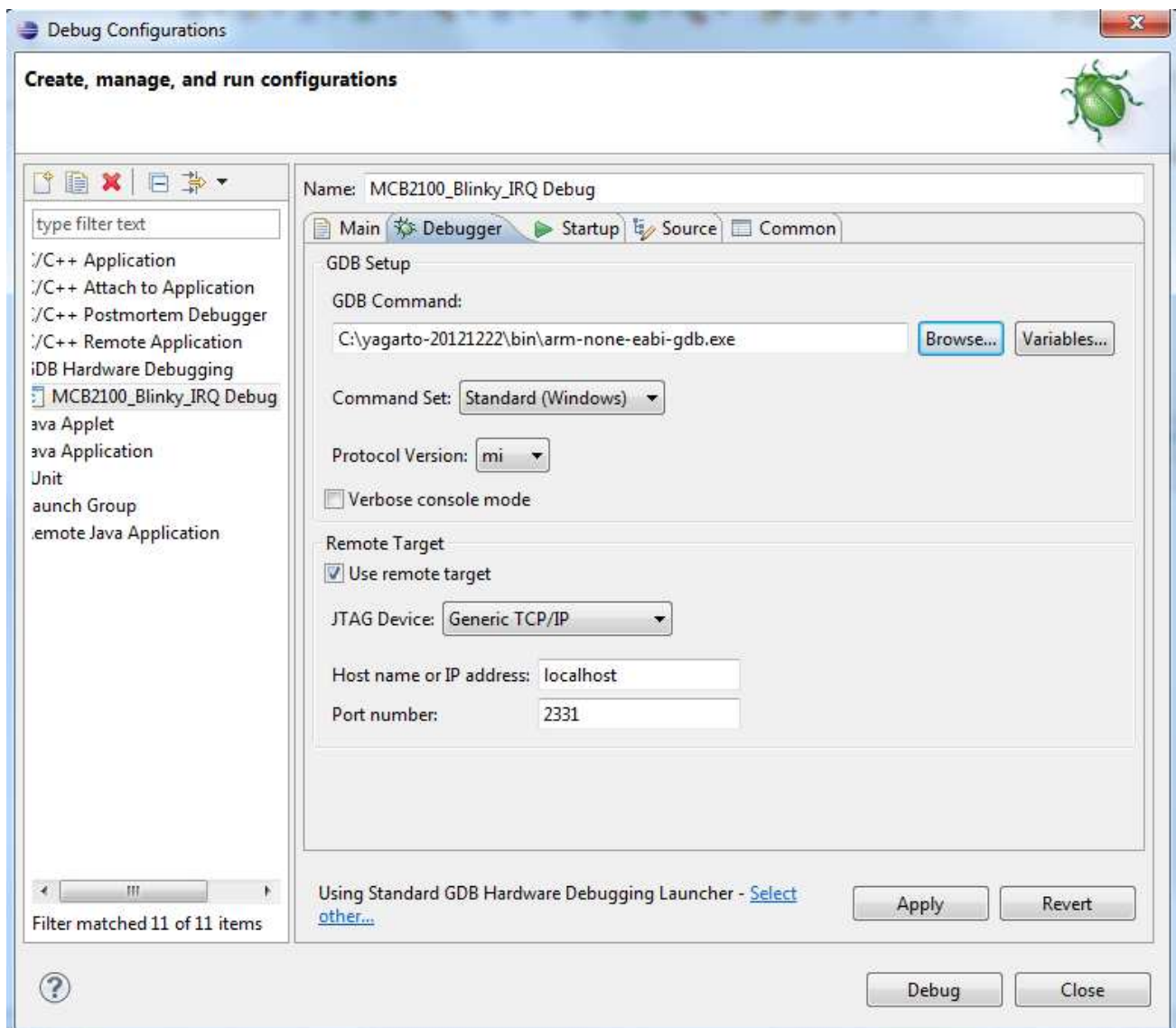


Fig. 3.12: Dialog of setting the GDB command and remote target.

16. click the **Apply** button to store the configuration
17. click the **Debug** button to actually start the debug session
18. The Eclipse should ask you:

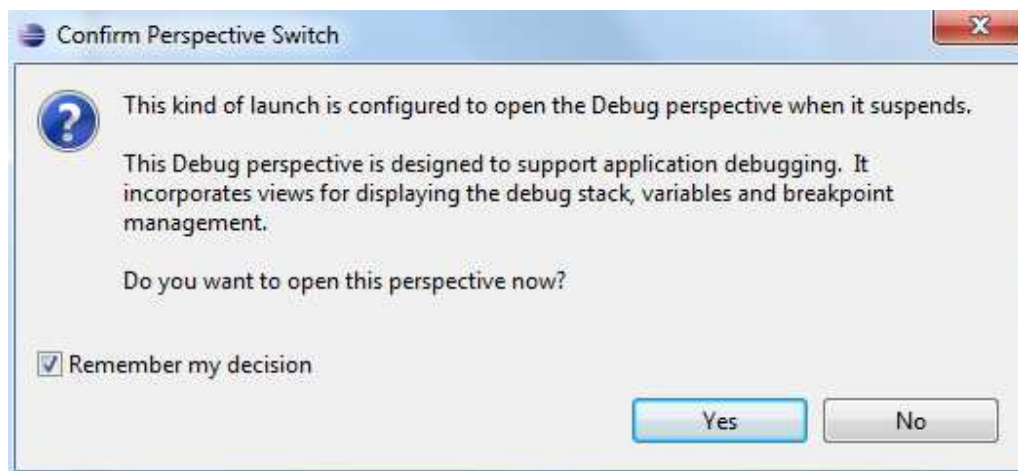


Fig. 3.13: Dialog of confirm perspective switch

Check remember my decision and click **Yes**. If Eclipse doesn't automatically switch to the Debug perspective, you can switch manually using menu *Windows* → **Open Perspective** → **Debug**.

19. Now the debugger is started:

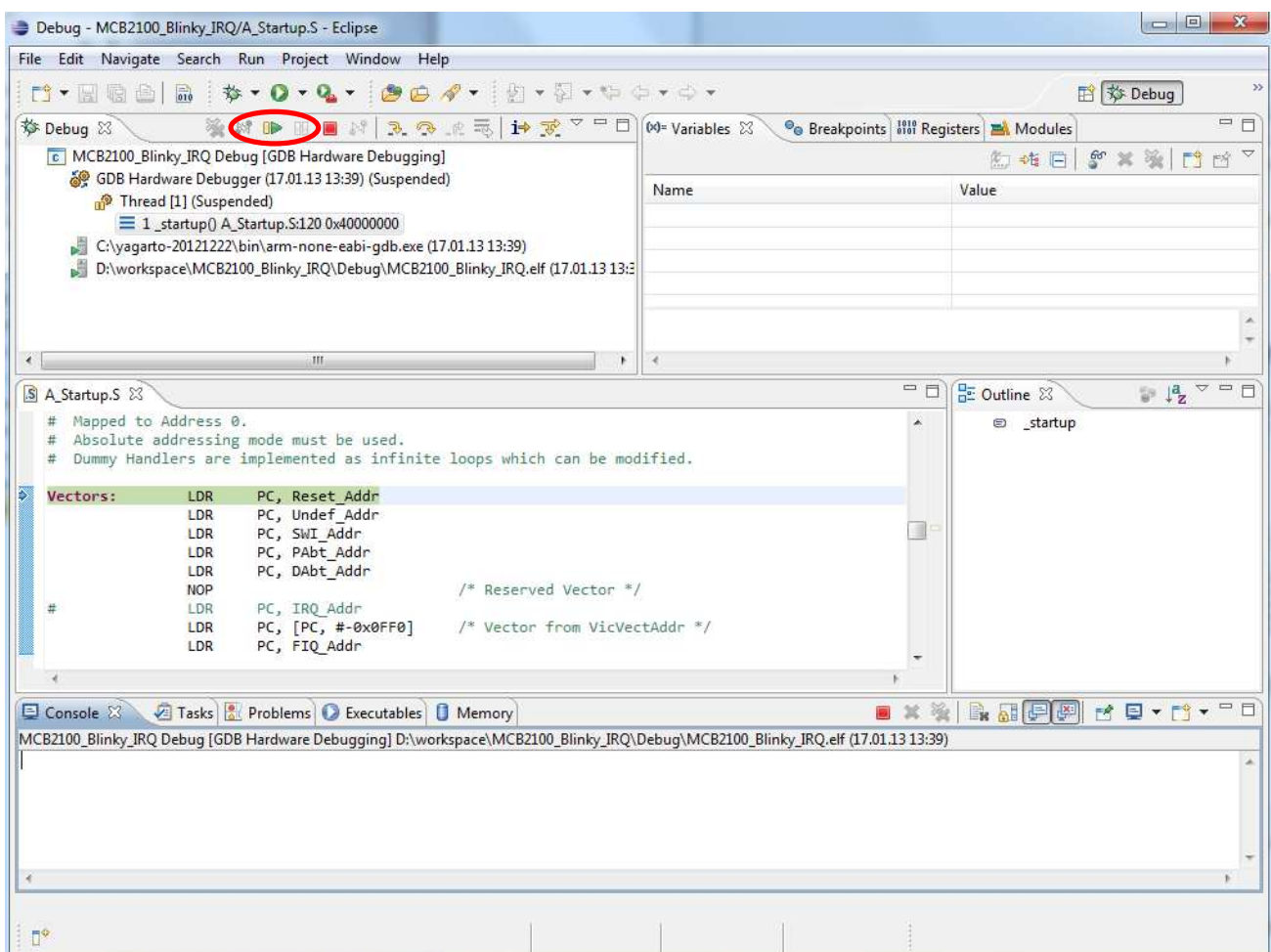


Fig. 3.14: Layout of the debug.

20. Click the **Resume** button, the project will run and you can see the blink of the LEDs.
21. If now you look the J-Link GDB, you will found that the GDB is connected to the local host as shown in the Fig. 3.15.

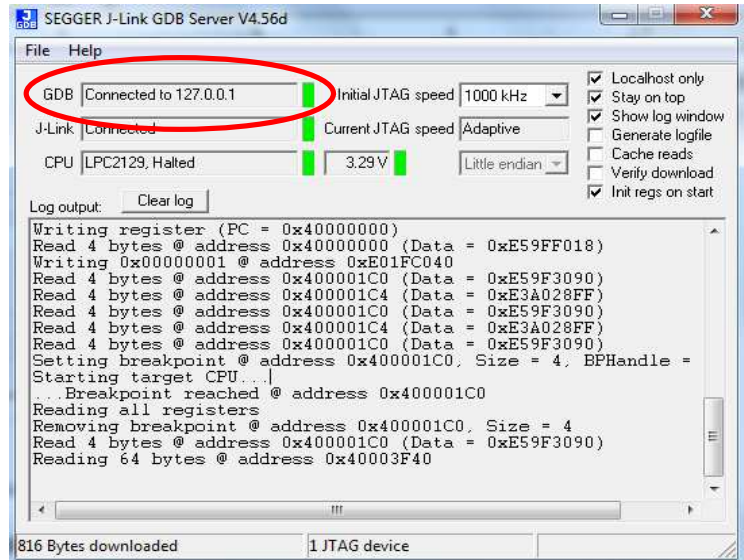


Fig. 3.15: Diagram of the segger J-Link GDB server during the running of the project.

4. Zylin Embedded CDT for Native Debug (for ASM Projects)

In addition, I will introduce the Zylin Embedded CDT to run an assembly language project. The main interested feature of the Zylin is that you can debug the project without hardware.

4.1 Install the Zylin Embedded CDT plug-in

Follow the same steps as described in Chapter 2.2.1

- Click the **Add** button on the top right of the **Install** dialog to open the **Add Repository** dialog.
- Type **Zylin Embedded CDT** in the **Name** field (this is only a name, so choose the one you prefer).
- Insert the following web address in the **Location** field: <http://opensource.zylin.com/zylincdt>
- Press the **OK** button to confirm your choice.
- After a while the **Install** dialog displays the list of the available Plug-ins.
- Select **CDT GNU Cross Development Tools** and press **Next**.

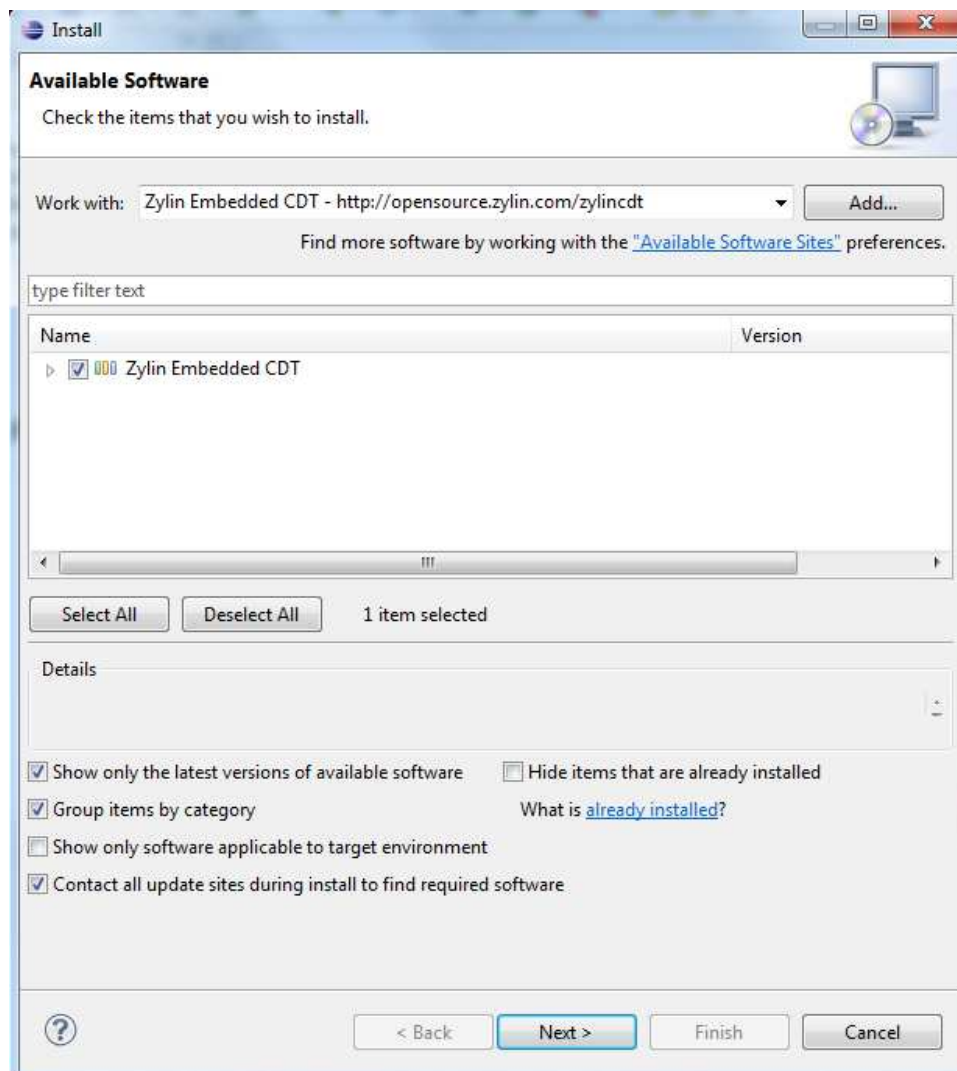


Fig. 4.1: Dialog of install the Zylin Embedded CDT Eclipse Plug-in

- Follow the onscreen instruction, don't worry about (but read ;-) the warning, and restart Eclipse when required to complete the installation.

4.2 Import an Existing Project

From the *File* menu, click **Import** and a dialog will be shown. In the treeview, expand **General** and highlight the **Existing Project into Workspace**.

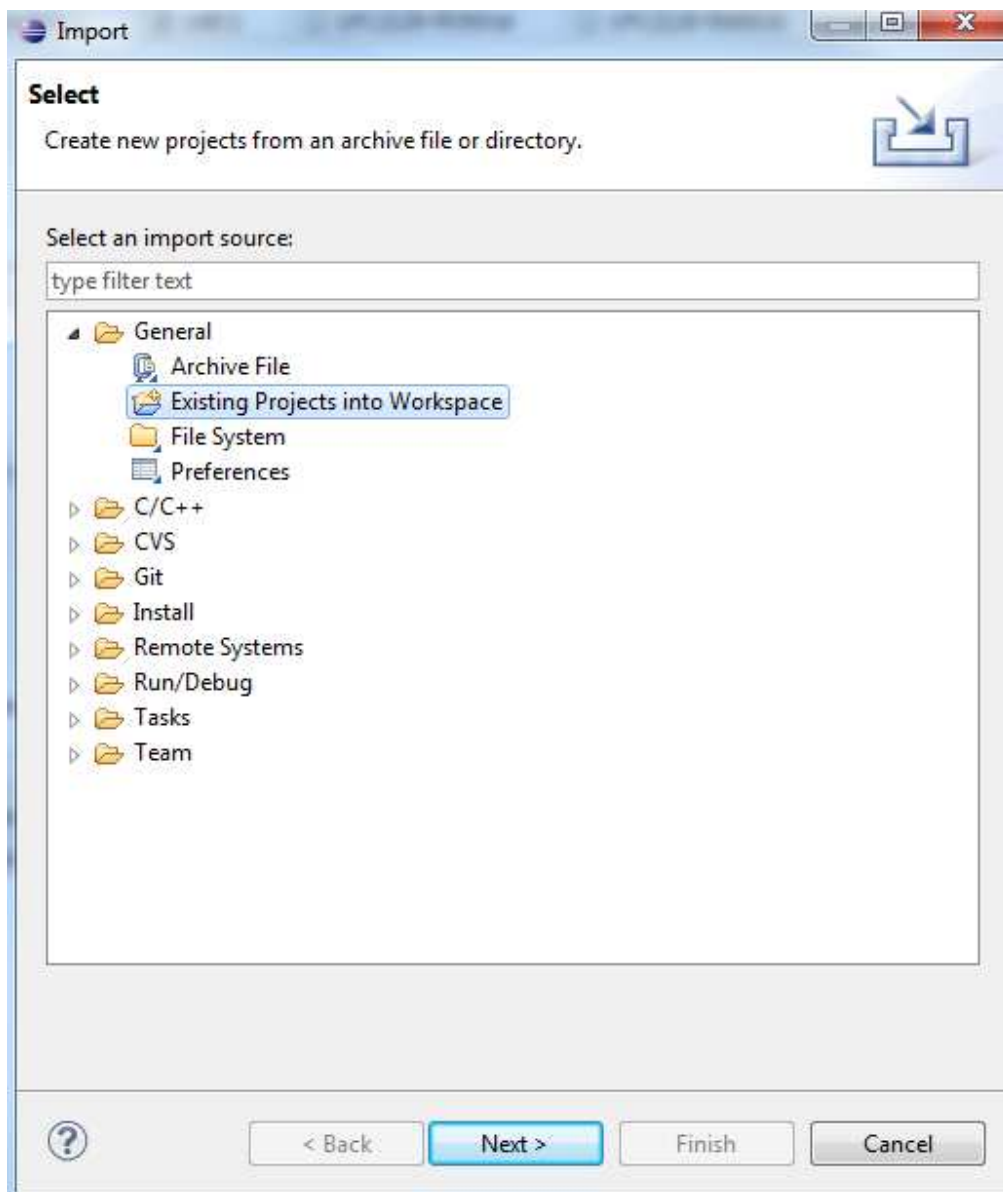


Fig. 4.2: Dialog of import action

Click **Next** and select the proper root directory of the project you want to imported. Here the “ASM_Gibson_43” project is selected as shown in the Fig. 4.3. Make sure that the “Copy projects into workspace” is selected.

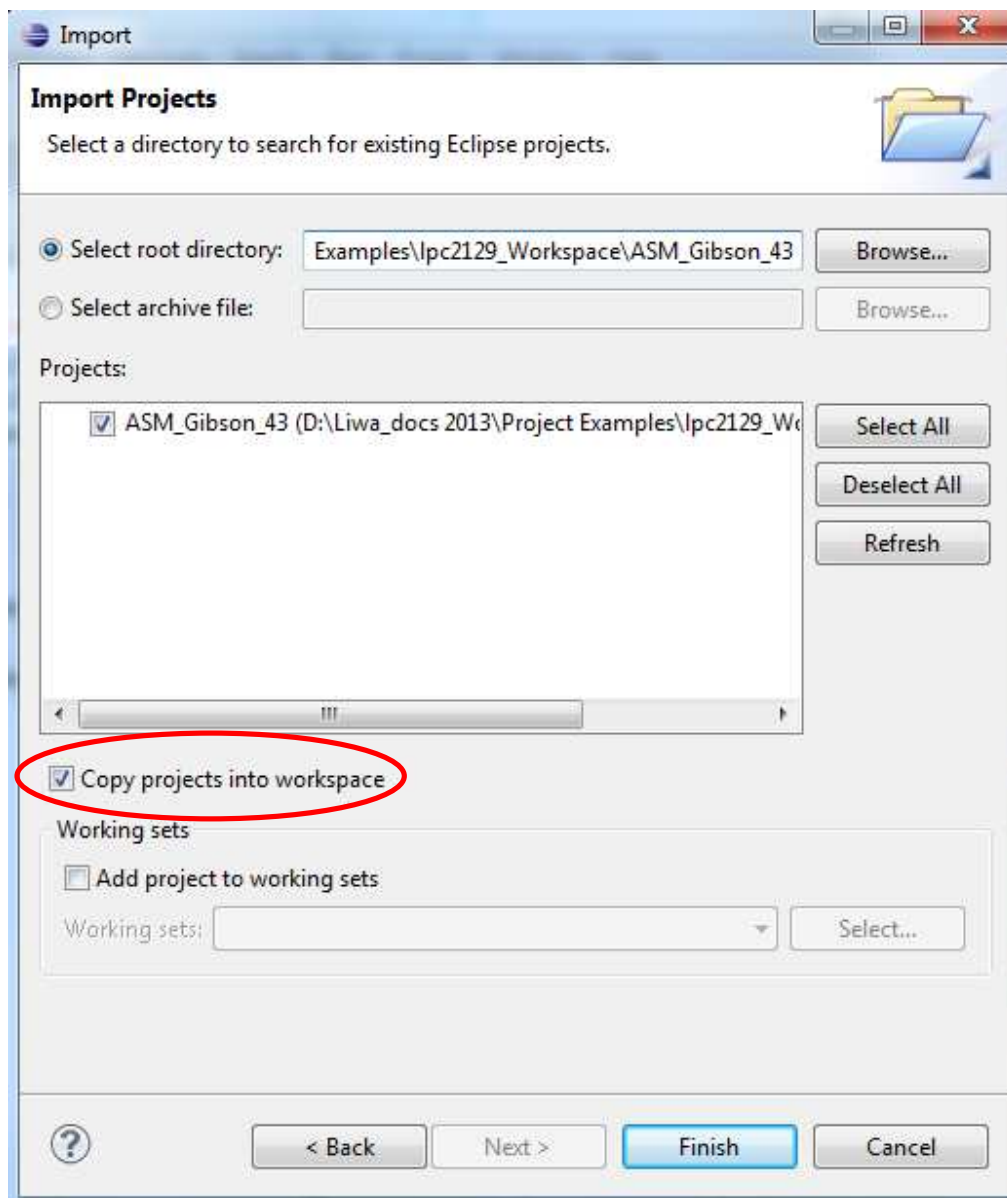


Fig. 4.3: Dialog of import projects

When you click **Finish**, the project will be imported and you can see it in the **Project Explorer** on the left side of your screen. Highlight **ASM_Gibson_43**, the whole project contents will be shown.

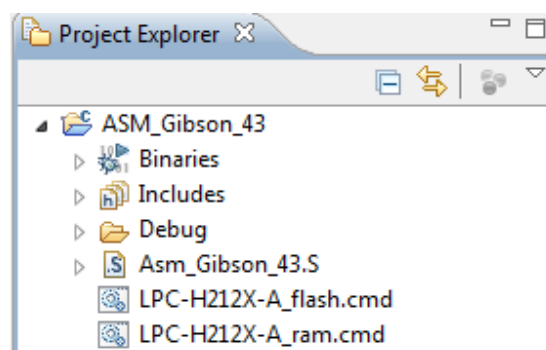
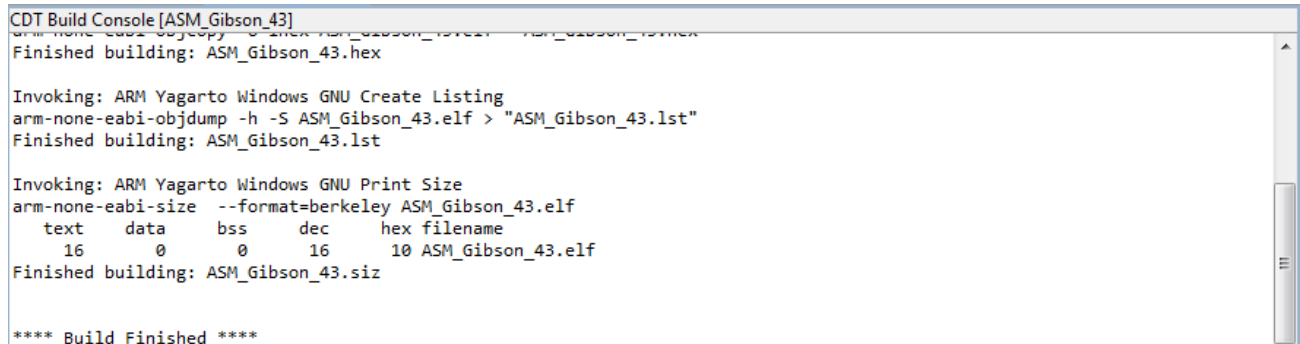


Fig. 4.4: Project structure in the project explorer

5.3 Build the Project

Follow the same steps described in Chapter 3.3 to build the project. When the build is finished, the output should be similar to the following in **Console**.



```
CDT Build Console [ASM_Gibson_43]
Finished building: ASM_Gibson_43.hex

Invoking: ARM Yagarto Windows GNU Create Listing
arm-none-eabi-objdump -h -S ASM_Gibson_43.elf > "ASM_Gibson_43.lst"
Finished building: ASM_Gibson_43.lst

Invoking: ARM Yagarto Windows GNU Print Size
arm-none-eabi-size --format=berkeley ASM_Gibson_43.elf
  text  data   bss   dec   hex filename
   16     0     0    16    10 ASM_Gibson_43.elf
Finished building: ASM_Gibson_43.siz

**** Build Finished ****
```

Fig. 4.5: Output of the build finished in console.

5.4 Setup Debug Configuration

1. Highlight the **ASM_Gibson_43** project, right click it, select **Debug As→ Debug Configurations...**

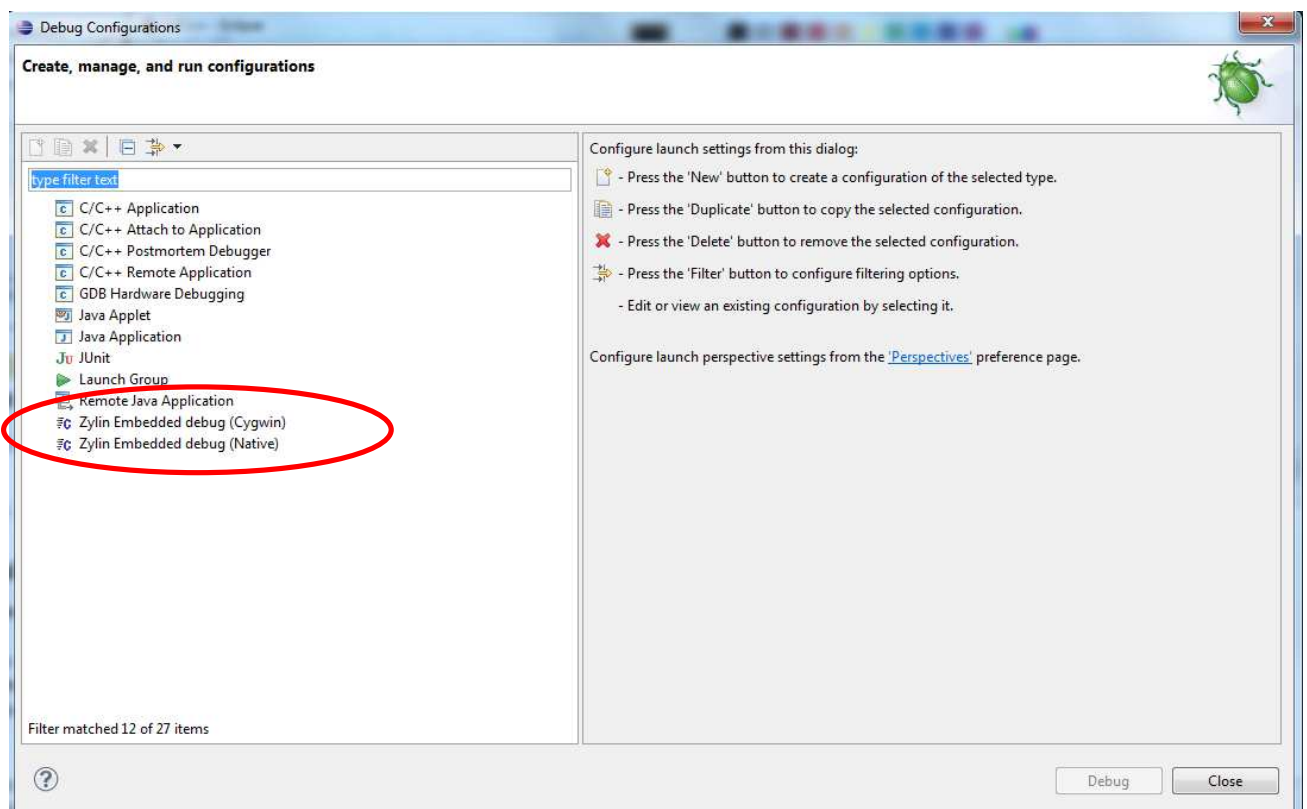


Fig. 4.6: Dialog of debug configurations

You can easily find that there are two additional selections compare to the Fig. 3.10.

2. Double click the **Zylin Embedded debug (Nativ)** to create a new configuration
3. select the **Main** tab

4. check if the C/C++ Application was properly selected (it should look like Debug\ASM_Gibson_43.elf)
5. check if the project was properly selected (it should look like ASM_Gibson_43) as shown in the Fig. 4.7

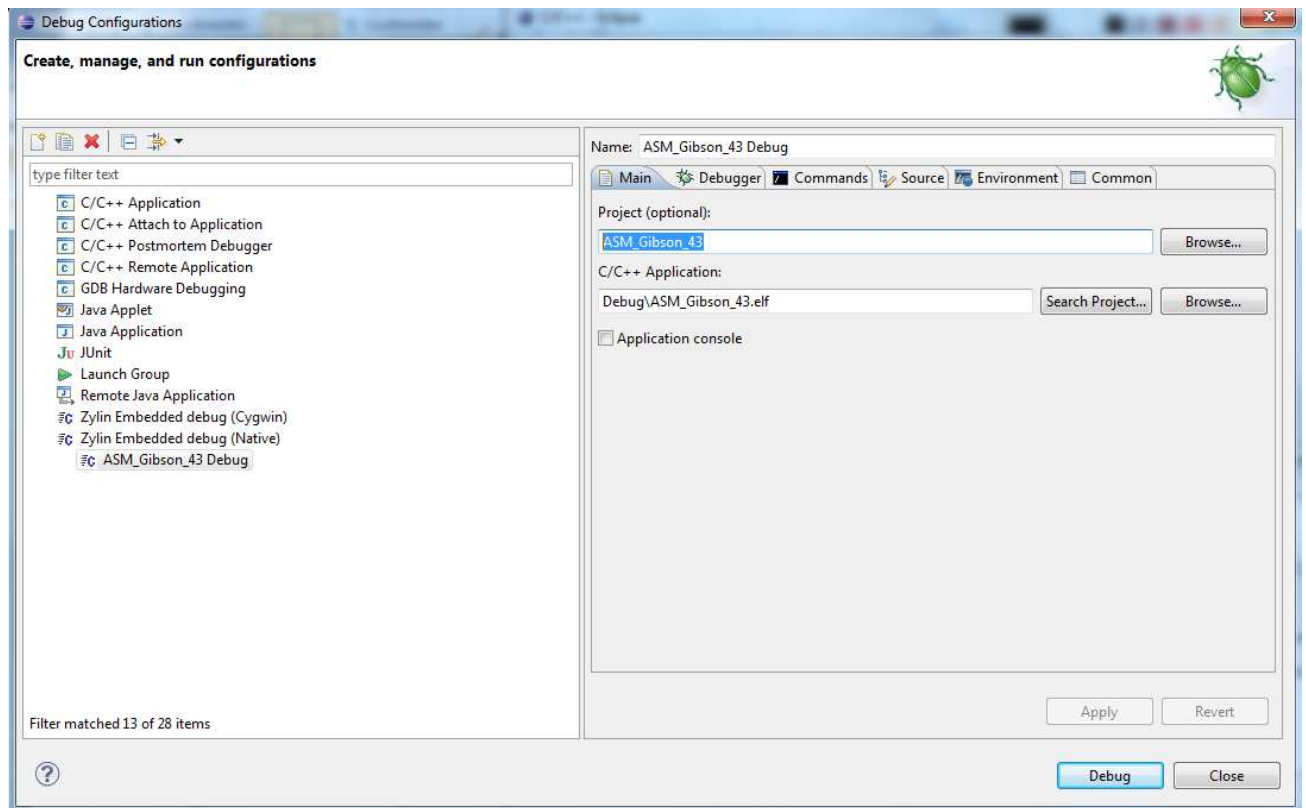


Fig. 4.7: Main setup of the debug configuration dialog

6. Select the **Debugger** tab, uncheck the “Stop on startup at main”, and in the GDB debugger text field, change the “arm-elf-gdb” to “arm-none-eabi-gdb” as shown in the Fig. 4.8.

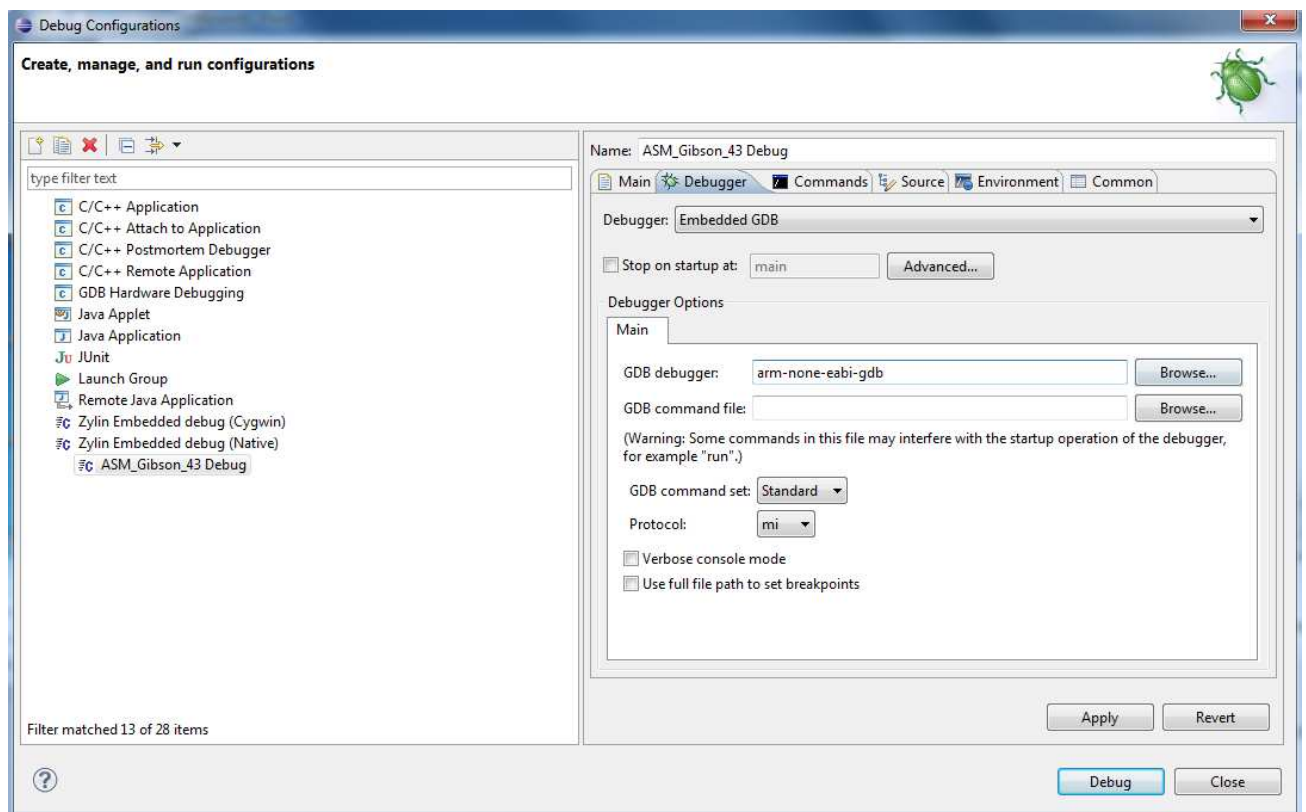


Fig. 4.8: Debugger setup of the debug configuration dialog

7. Select the **Commands** tab. In the 'Initialize' commands field write commands as shown in the Fig. 4.9.
8. click the **Apply** button to store the configuration
9. click the **Debug** button to actually start the debug session

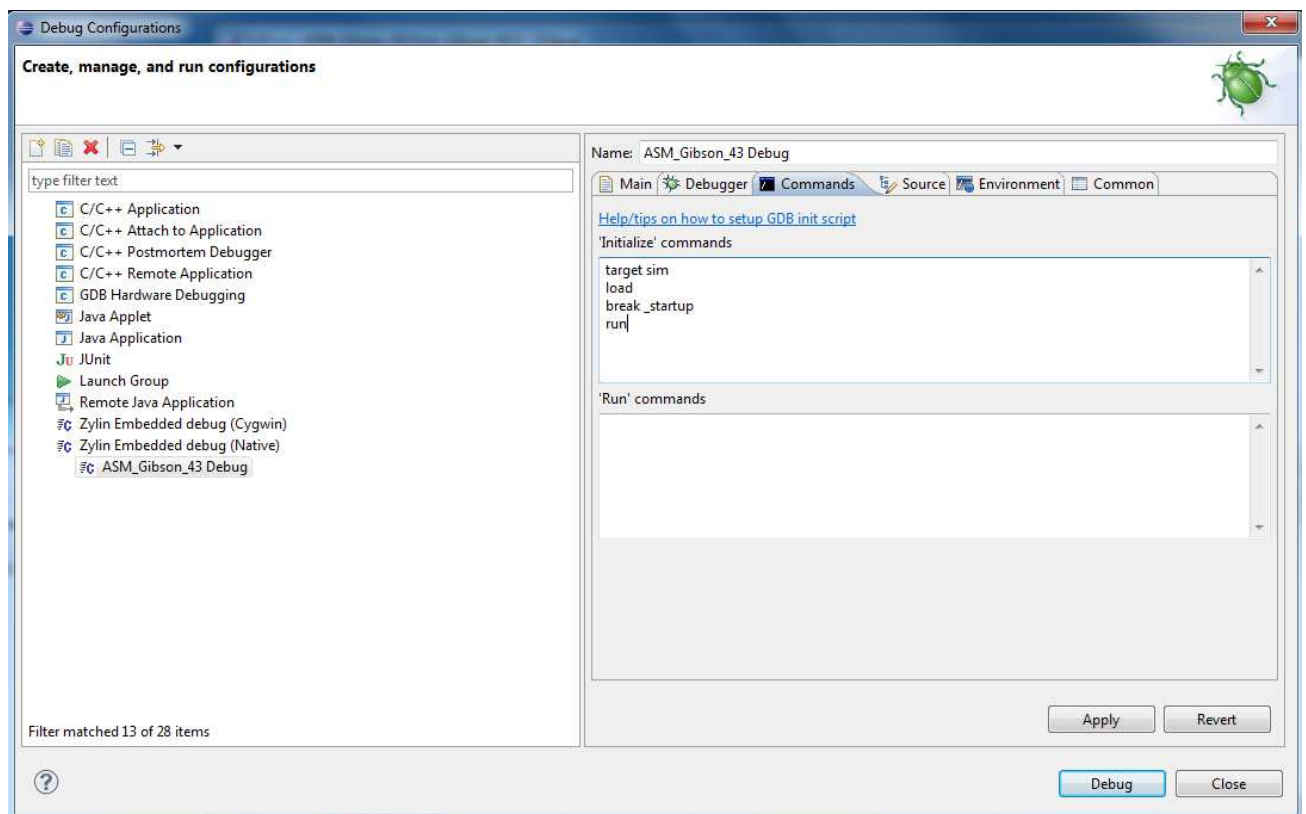


Fig. 4.9: Commands setup of the debug configuration dialog

10. Now the debug is started

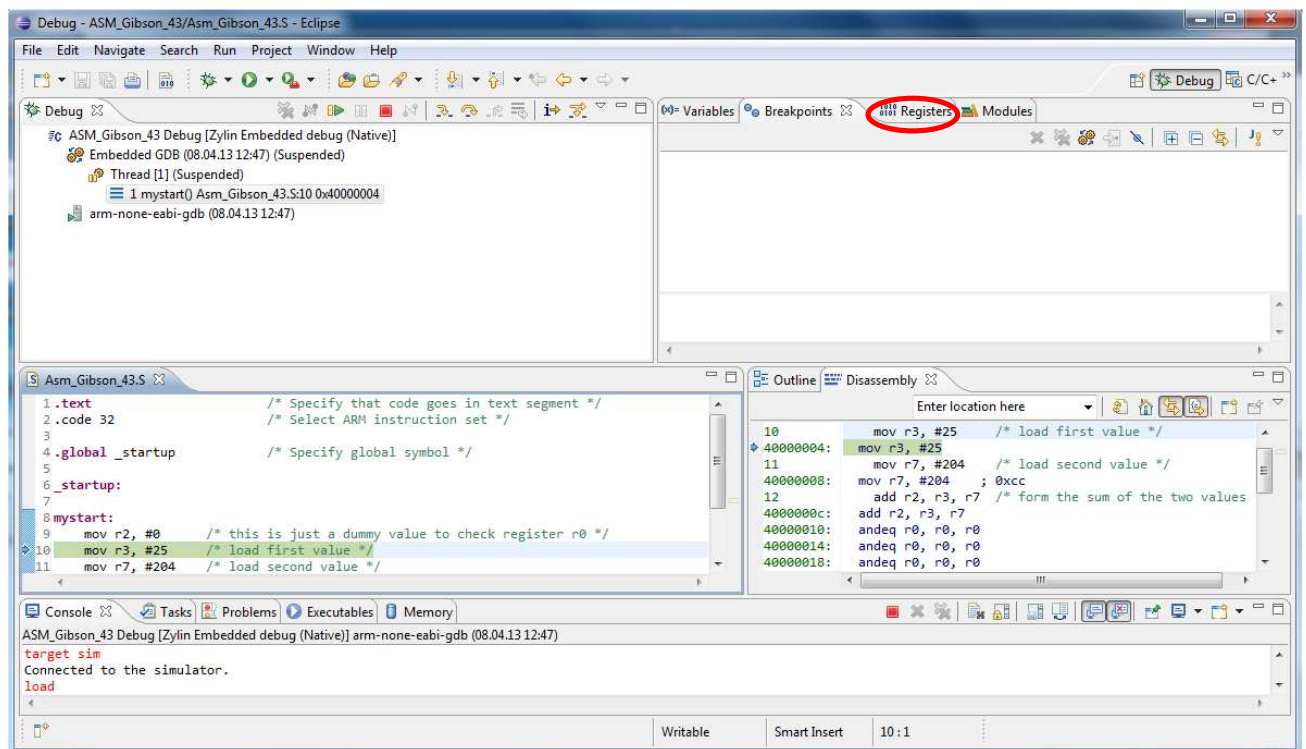


Fig. 4.10: Layout of the debug.

11. Click the **Registers** and then highlight the **Main**, you can monitor the value of all the registers as shown in the Fig. 4.11.

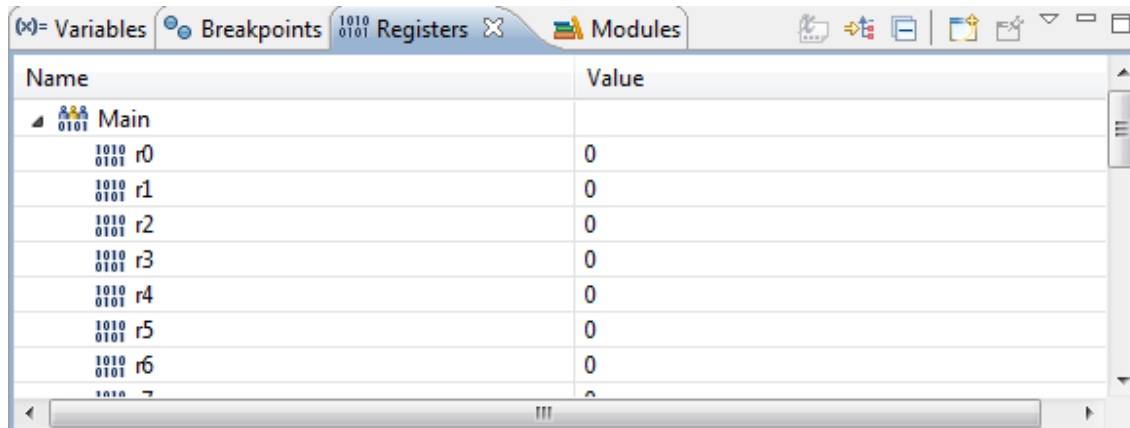


Fig. 4.11: Dialog of register monitor

12. Use the **Step Over** icon or **F6**, you can run the program step by step and monitor the variation of the value of the corresponding registers. Good Luck☺.