



... the world's most energy friendly microcontrollers

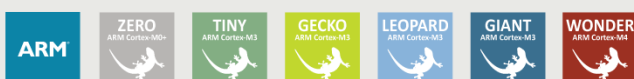
Configuring Eclipse for EFM32 Development

AN0023 - Application Note

This application note explains how to install Eclipse on Windows and set up a simple project. Compiling and debugging the code on an EFM32 microcontroller is demonstrated. All the software components used in this application note come free of charge.

This application note includes:

- This PDF document
- Eclipse support files (zip)
 - Eclipse Embedded Systems Register View plugin
 - Eclipse GDB Hardware Debugging plugin
 - Cortex-M CPU register view patch



1 Prerequisites

You will need an Energy Micro Starter or Development kit with an EFM32 Cortex-M3 microcontroller.

The zip-file `an0023_efm32_eclipse_toolchain.zip` that comes with this application note contains the required Eclipse plug-ins and an xml-file that patches older versions of J-Link GDB Server to show the correct CPU registers for Cortex-M3 targets. This zip-file can be zipped directly into the Eclipse install folder.

1.1 A note on versions

To succeed in making your setup work, it is important to notice that there are many different tools from different sources involved. The versions shown in Table 1.1 (p. 2) has been tested to work, and it is recommended to use these versions as a starting point. Once the setup is stable, feel free to experiment with newer tool versions.

Microsoft Windows 7 Professional was used to prepare this application note.

These software packages also exists for Linux, and although this application note is Windows specific, the setup procedure should be fairly similar.

Table 1.1. Tool version information

Tool name	Version
Eclipse IDE for C/C++ Developers	Version: 3.7.2 Build id: 20120216-1857 (Indigo)
C/C++ GDB Hardware Debugging plugin for Eclipse	Version: 7.0.0.201202111925
Embedded Systems Register View plugin for Eclipse	0.2.0
Sourcery CodeBench Lite for ARM/EABI	2012.03-56 gcc version 4.6.3 gdb version 7.2.50.20100908-cvs
SEGGER J-Link ARM	V4.36E
Simplicity Studio	n/a

J-Link GDB Server is part of SEGGER's J-Link Software and Documentation Pack that comes with Simplicity Studio. Newer versions can be downloaded from SEGGER. These newer versions don't need the Cortex-M register patch that is included with this application note.

1.2 Installing the tools

1.2.1 Simplicity Studio

Download Simplicity Studio from

<http://www.energymicro.com/simplicity>

and install it.

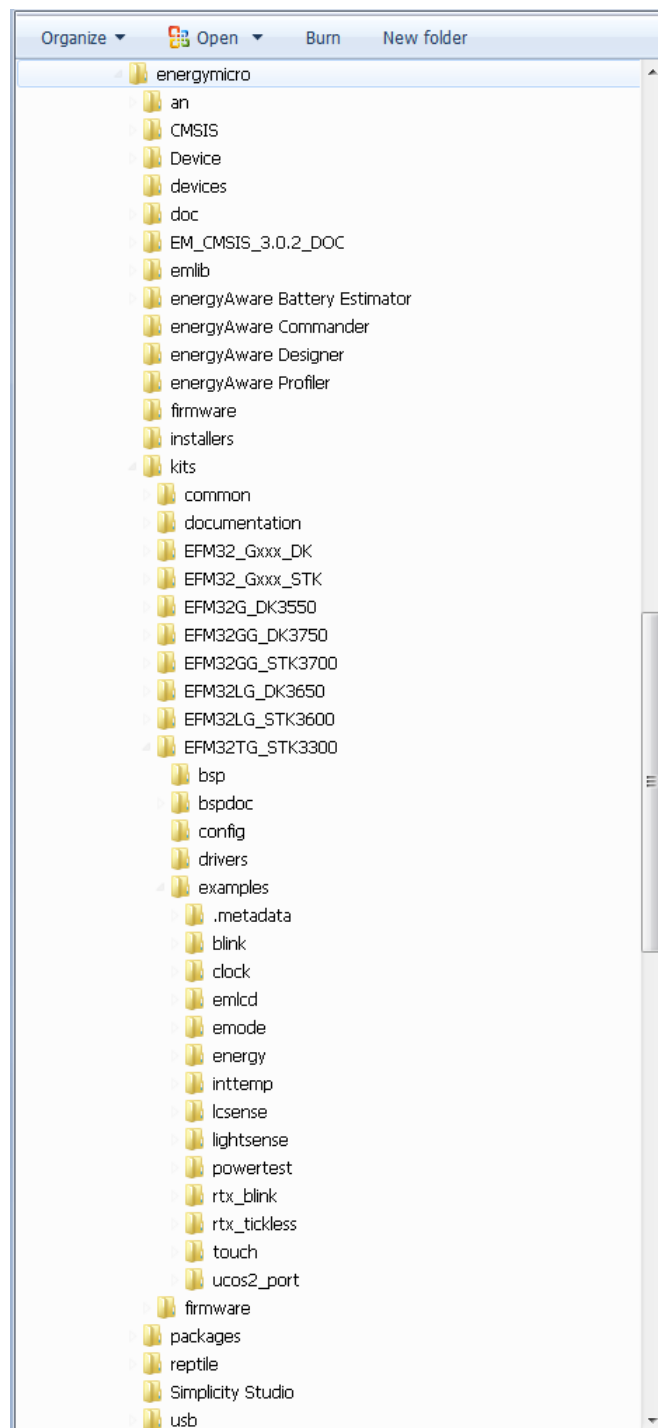
Launch Simplicity Studio and answer **Yes** to the question about installing the recommended packages. Answer **OK** to the question about installing the J-Link drivers.

You will get a directory structure as shown below within your Simplicity Studio work directory at

`C:\%APPDATA%\energymicro\`

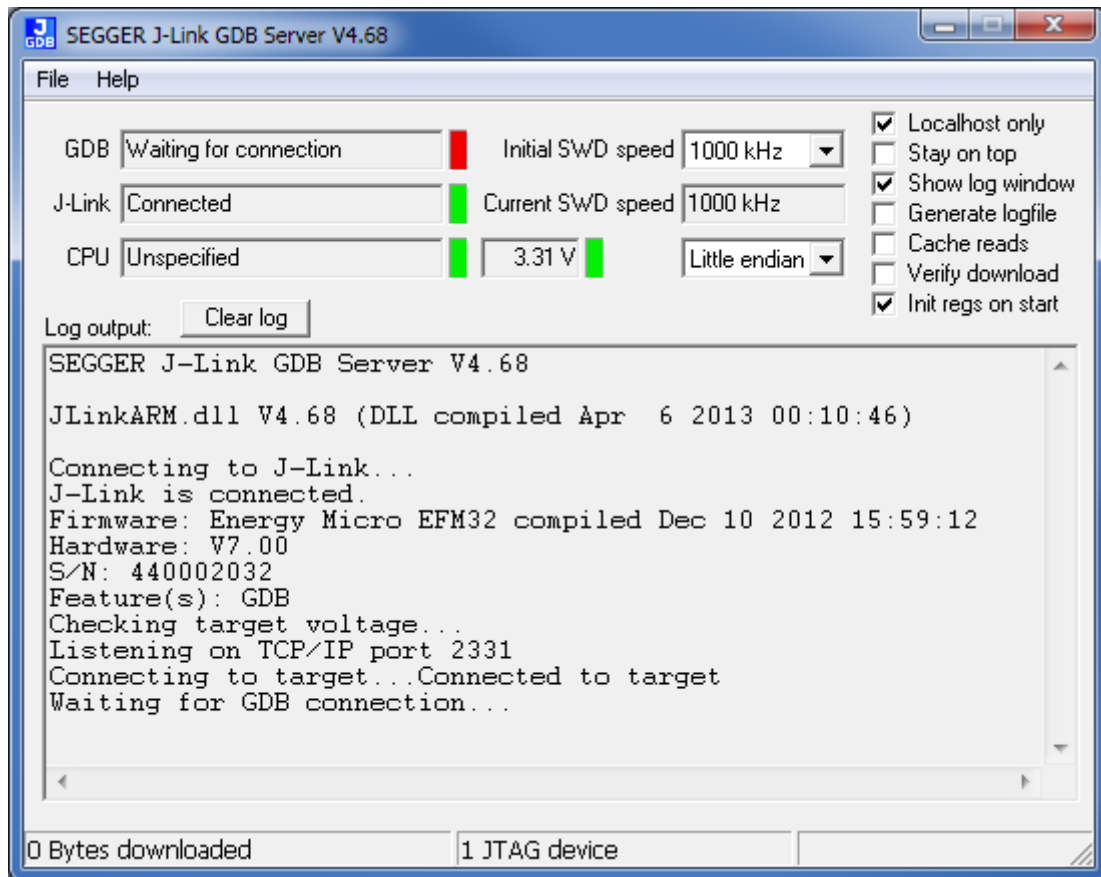
The location of %APPDATA% depends on which version of Windows you are, and the directory is hidden by default. Pay special attention to the `blink` example as it will be used later in this application note.

Figure 1.1. Energy Micro folder structure



It is now time to connect the kit to your machine via the supplied USB cable and check that the J-Link device driver was properly installed.

From the start menu, launch the program called *J-Link GDB Server via SWD* found in the SEGGER folder. If everything is OK, the server should be able to connect to the board and the GUI should look something like:

Figure 1.2. J-Link GDB Server GUI

Do not exit this program before continuing.

1.2.2 Sourcery CodeBench Lite

Install the *Sourcery CodeBench Lite*. It is available from:

<http://www.mentor.com/embedded-software/sourcery-tools/sourcery-codebench/editions/lite-edition>

Download the EABI Release for ARM processors.

When prompted with a question to add CodeBench to the PATH environment variable, you should accept.

1.2.3 Eclipse IDE for C/C++ Developers

As Eclipse is a Java application you must have Java Runtime Environment (JRE) installed on your computer before installing Eclipse. Most computers already have a Java installation. You can check if Java is installed by visiting:

<http://java.com>

and click on the link named *Do I have Java?*, or by issuing the command:

```
java -version
```

in a command prompt.

Eclipse is available from:

[http://www.eclipse.org/downloads/packages/eclipse-ide-cc-developers-
includes-incubating-components/indigosr2](http://www.eclipse.org/downloads/packages/eclipse-ide-cc-developers-includes-incubating-components/indigosr2)

When downloading Eclipse make sure you select the Windows version. The Indigo (SR2) edition was used in this application note.

Simply unpack the zip file at C:\ and you have a full Eclipse installation at C:\eclipse.

The two plugins needed can be installed from online repositories as described in the following sections, or by extracting `an0023_efm32_eclipse_toolchain.zip` into your Eclipse install folder. If you elect to use the plugins in the zip file, copy the relevant files into the `features` and `plugins` subdirectories under C:\eclipse. This directory structure is replicated inside the zip file.

1.2.3.1 C/C++ GDB Hardware Debugging plugin

Start Eclipse by executing C:\eclipse\eclipse.exe. When prompted for workspace navigate to the `examples` directory for your kit in Simplicity Studio's working directory. In our case

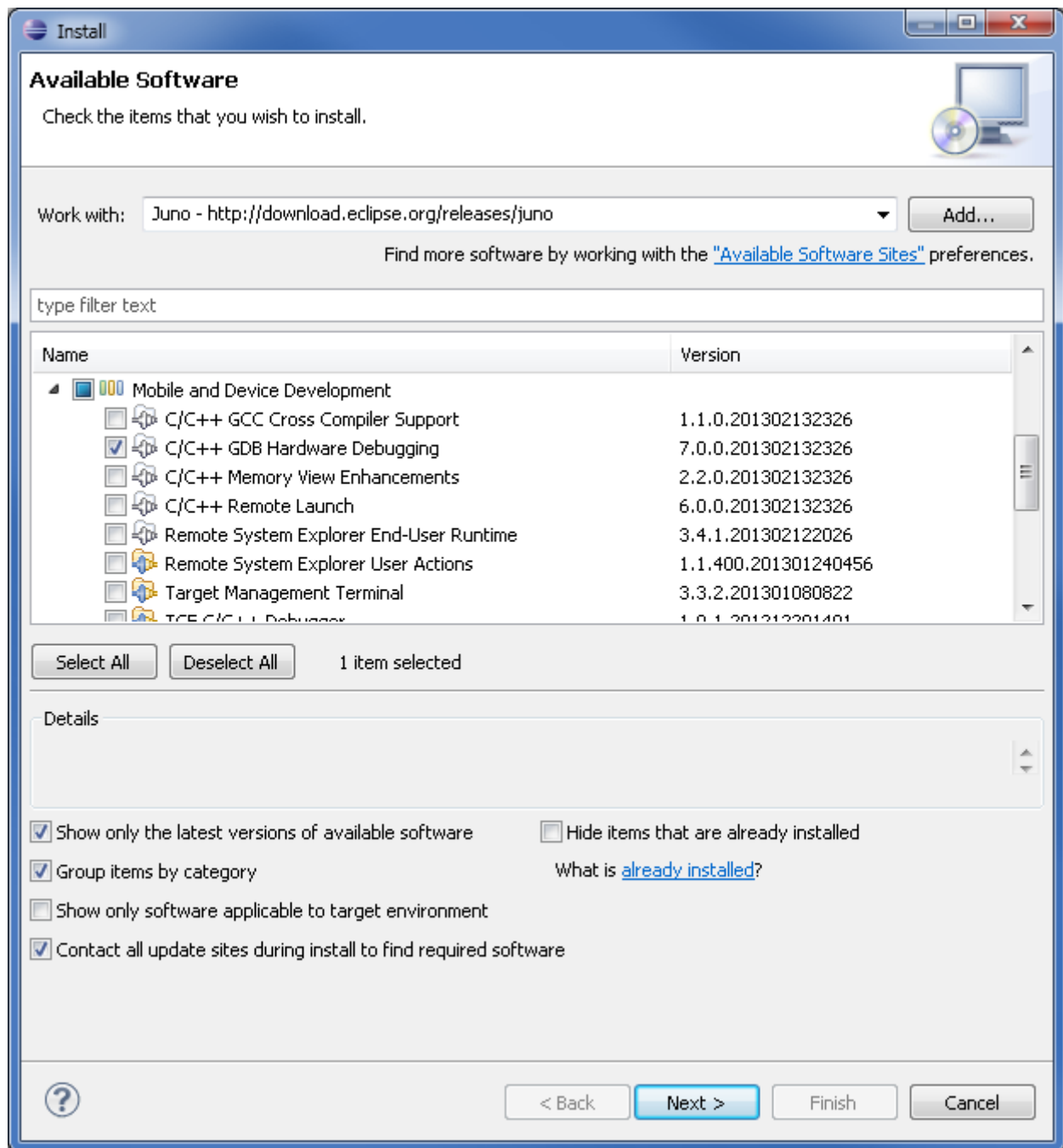
C:\%APPDATA%\energymicro\kits\EFM32TG_STK3300\examples

You will now be greeted by Eclipse's welcome screen.

Select *Install New Software...* from the *Help* pulldown menu. In the *Work with* field, enter:

<http://download.eclipse.org/releases/indigo>

You might need to wait a while for the plugins to show up. Browse to *Mobile and Device Development* and select *C/C++ GDB Hardware Debugging*. Press **Next >**, and follow the instructions.

Figure 1.3. GDB Hardware Debugging plugin installation

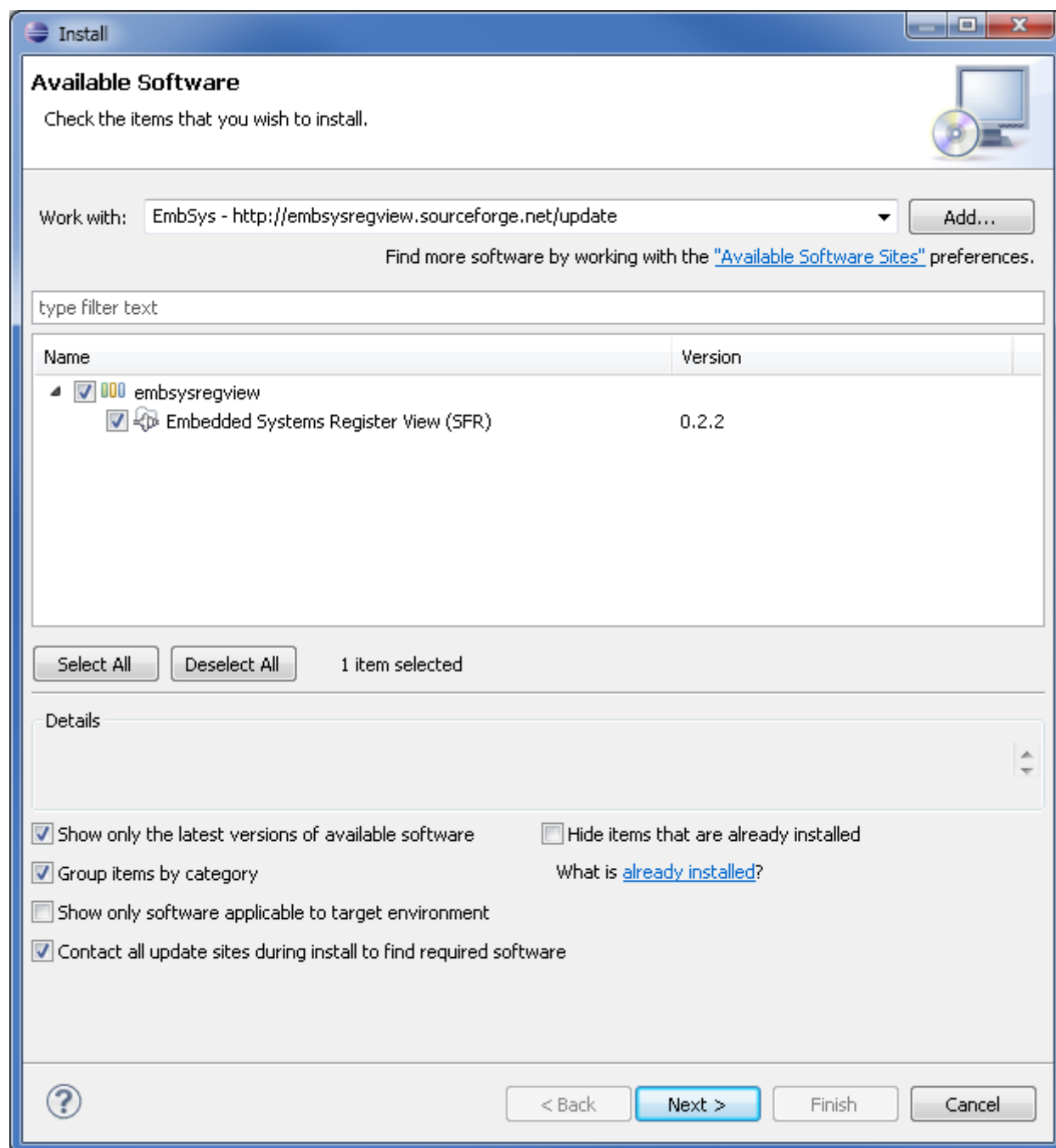
The installation process may take a while. When asked to restart Eclipse, do so.

1.2.3.2 Eclipse Embedded Systems Register View plugin

Proceed as described in the previous section, but in the Work with field, enter:

`http://embsysregview.sourceforge.net/update`

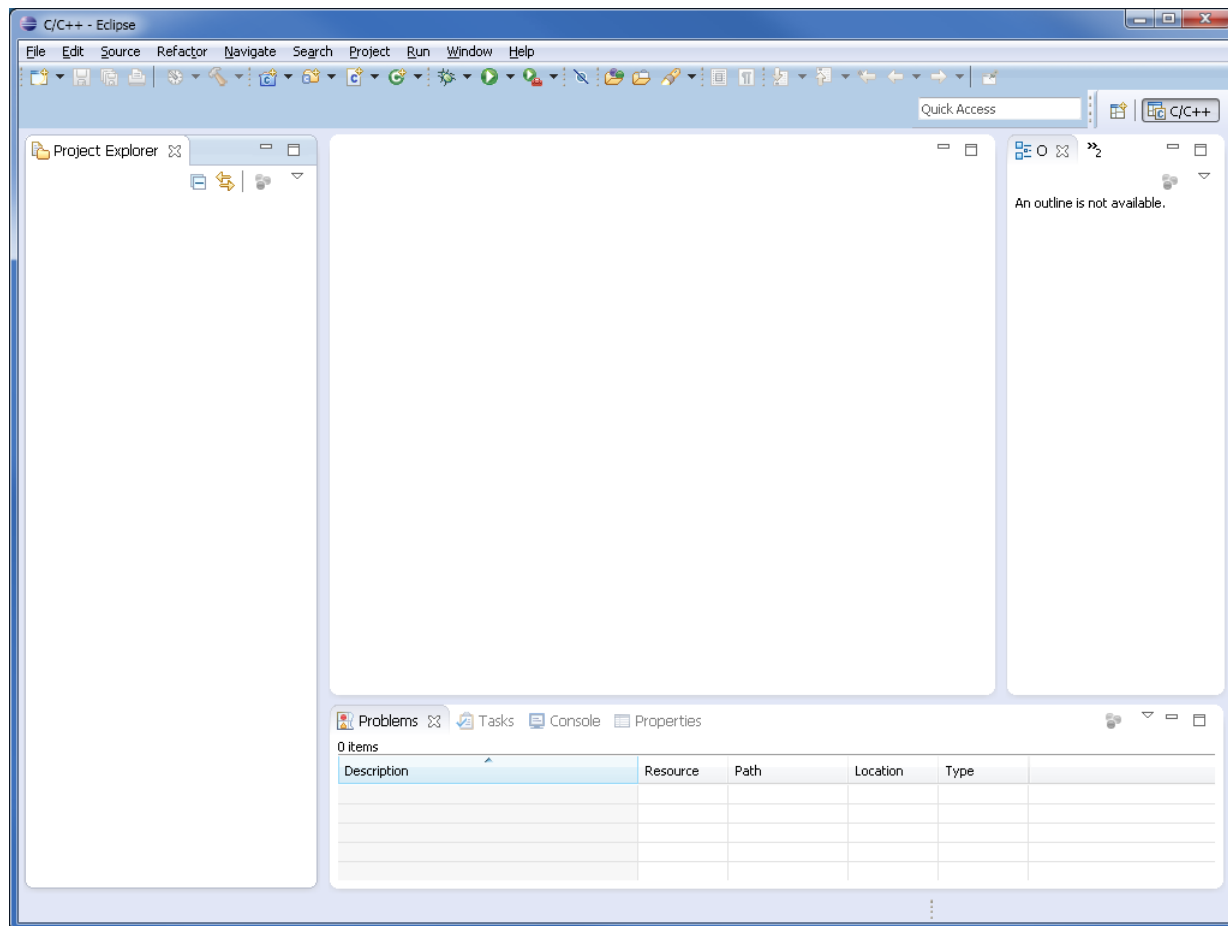
Select *Embedded Systems Register View*, click **Next >** and restart Eclipse when asked to do so.

Figure 1.4. Embedded Systems Register View plugin installation

2 Working with Eclipse

Now it is time to continue with Eclipse. Close Eclipse's welcome screen tab and Eclipse's Workbench view appears:

Figure 2.1. The Eclipse Workbench View

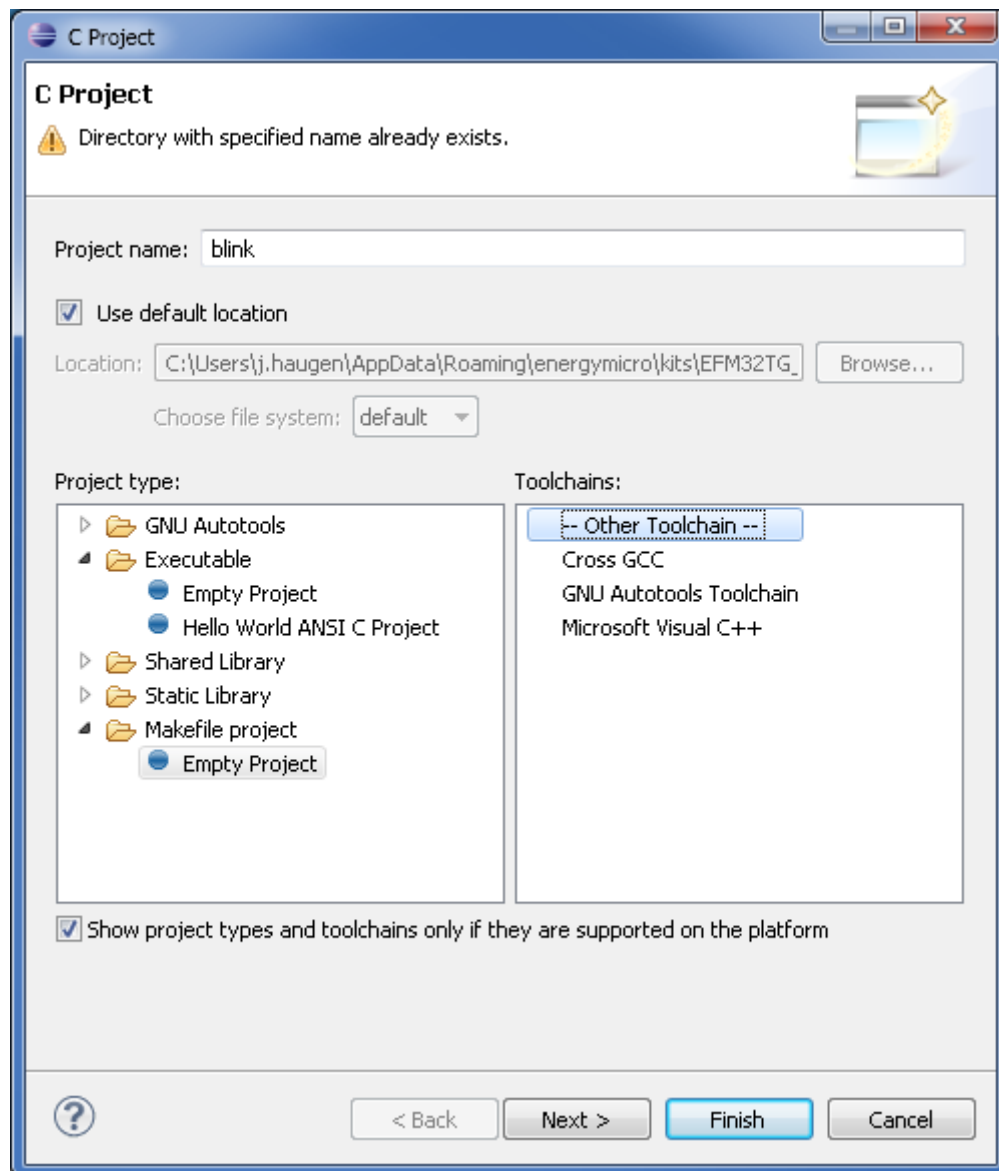


2.1 Create a project

The new project will be based on the 'blink' example project for the EFM32TG-STK3300. It is quite simple to follow this procedure to setup a project for another EFM32 device.

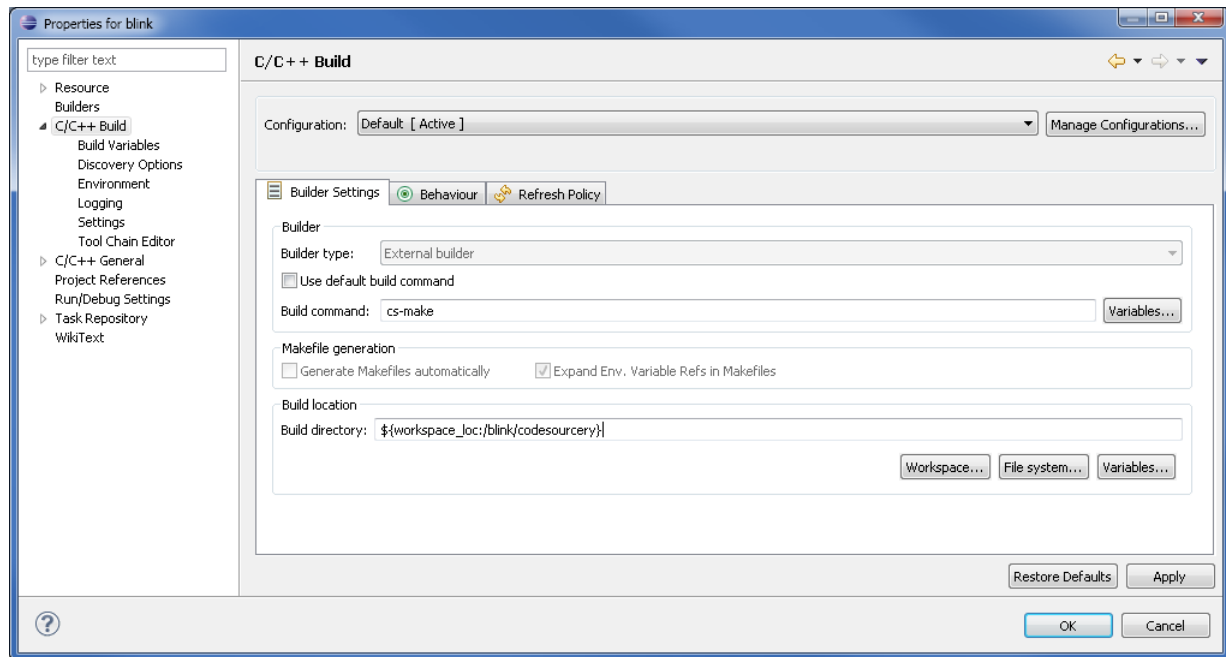
Create the project:

1. Select **File -> New -> C Project**
2. For Project name use **blink** (same name as the project directory)
3. For Project type select **Makefile project**
4. For Toolchain select **-- Other Toolchain --**
5. Click the **Finish** button to complete the project definition.

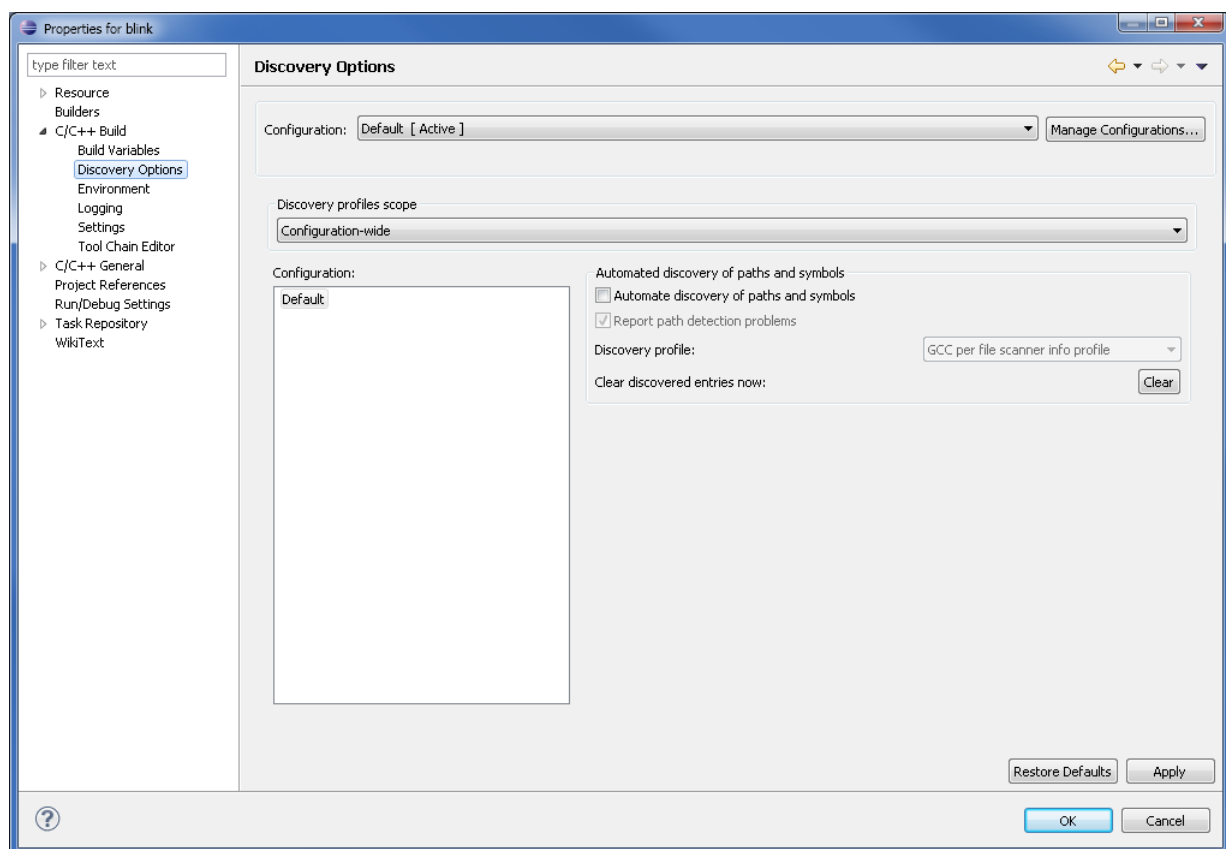
Figure 2.2. Create a new project in Eclipse

Add project properties:

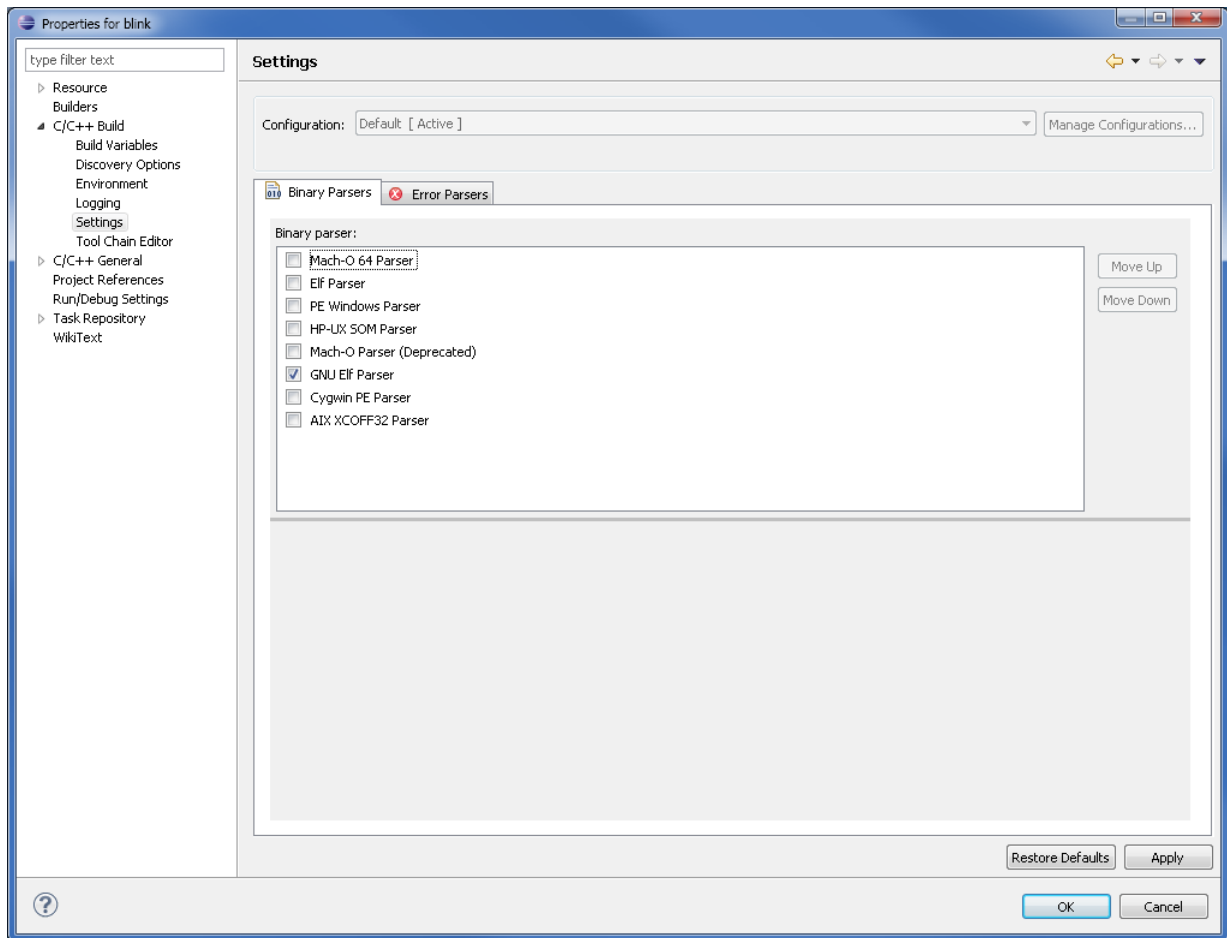
1. Select **Project** -> **Properties**
2. Expand **C/C++ Build**
3. **Uncheck** "Use default build command"
4. For "Build command" type in **cs-make**
5. Modify Build directory to **\${workspace_loc:/blink/codesourcery}**

Figure 2.3. Eclipse Project Properties - Build

6. Navigate to **C/C++ Build -> Discovery Options**
7. **Uncheck** "Automate discovery of paths and symbols"

Figure 2.4. Eclipse Project Properties - Discovery Options

8. Navigate to **C/C++ Build -> Settings**
9. **Check** GNU Elf Parser

Figure 2.5. Eclipse Project Properties - Build Settings

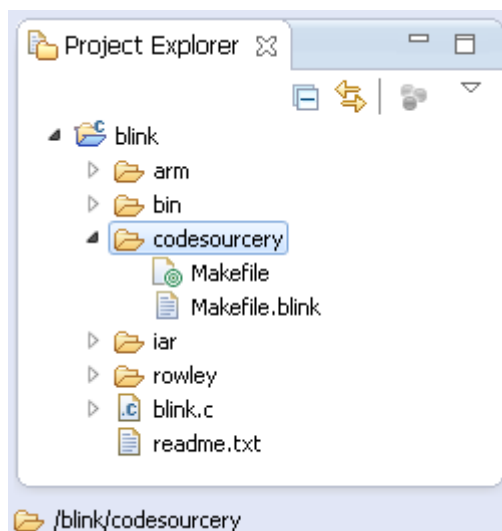
Click the **OK** button to save project properties.

2.1.1 Makefile

Some changes also needs to be made to the Makefile. We will want to inspect the Makefile and understand how it is setup and ensure it is setup correctly for our project.

1. Find the `Makefile.blink` in the `blink/codesourcery` directory
2. Save it with filename `Makefile` without a file type suffix

These operations are easily performed by right-clicking on the files in the Project Explorer pane and selecting Copy and Paste. In the Eclipse Project Explorer you should now see a green bullseye next to the Makefile.

Figure 2.6. Makefile location in Eclipse Project Explorer

Open the Makefile inside Eclipse by double-clicking on it in the Project Explorer. Do the following changes:

1. Change the WINDOWSCS variable. It should point to your CodeBench Lite installation relative to the Program Files folder

```
WINDOWSCS = CodeSourcery/Sourcery_CodeBench_Lite_for_ARM_EABI
```

2. Check that the CFLAGS macro contains option -O0. The -O<n> option select code optimization level. Using -O0 makes it easier to use the debugger.

```
debug: CFLAGS += -DDEBUG -O0 -g3
```

3. Set the PROJECTNAME macro to the same name as the project root directory

```
PROJECTNAME = blink
```

4. Set the DEVICE macro to reflects the EFM32 you use

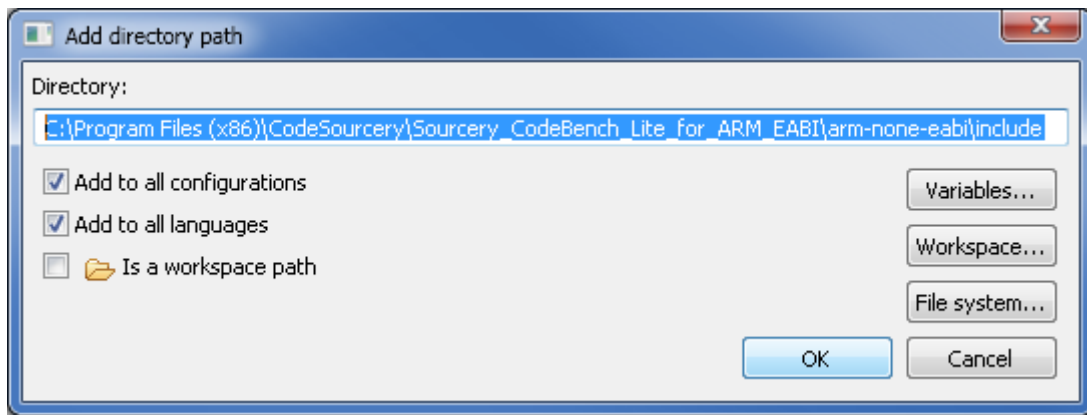
```
DEVICE = EFM32TG840F32
```

In the Makefile you will find the Include paths for header files as well as C source files that are being compiled in the project. When you want to add more functionality from `emlib` to your project, you have to add the proper path and C source file location to the makefile.

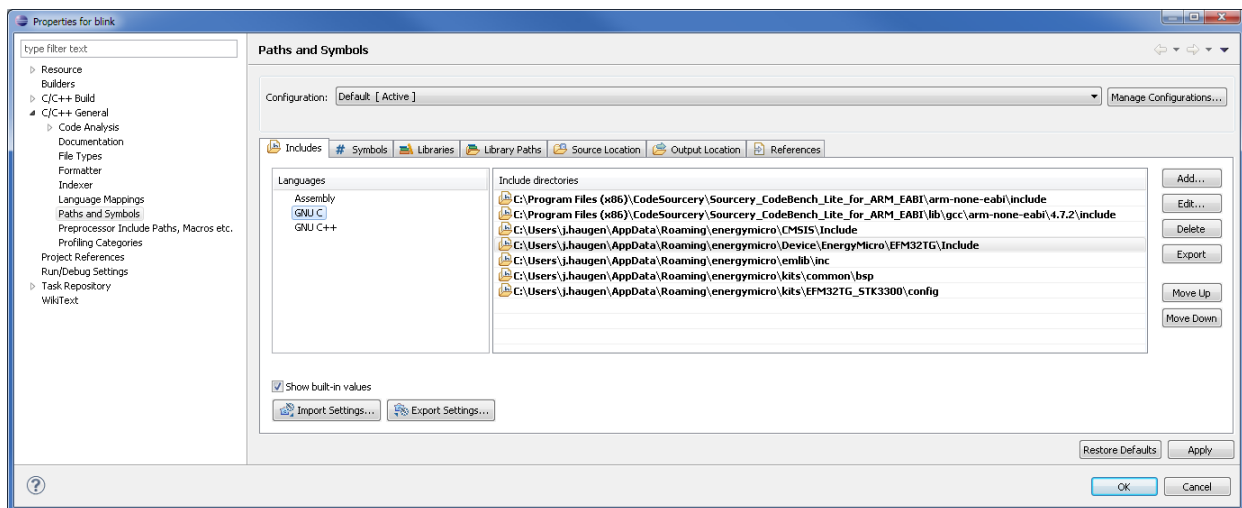
2.1.2 Add Eclipse Paths and Symbols

Even though the project will now compile, Eclipse will mark errors. This is because Eclipse is rather separated from the CodeBench GCC toolchain, and Eclipse do not know where the various header files are located. GCC knows this because of the Makefile but these details are not inherited by Eclipse. So we need to tell Eclipse where these files are.

1. From main menu, open **Project -> Properties**
2. Navigate to **C/C++ General -> Paths and Symbols**
3. Select **Add...**
4. Select **Add to all configurations** and **Add to all languages**
5. Select **File system...** to add folders to path

Figure 2.7. Add directory paths

6. Add all the folders from INCLUDEPATH in the Makefile
7. Add the include folders for CodeBench GCC as shown in Figure 2.8 (p. 13)

Figure 2.8. Eclipse Project Properties - Paths and Symbols

2.2 Build the code

We are now ready to compile the project. It can be useful to make sure the Console Tab is showing in the bottom of Eclipse so we can see the CodeBench GCC compiler output.

There are multiple ways to build the project. Some are listed below:

- From the main menu, select **Project -> Build Project**
- From the Project Explorer pane, right click on the top level 'blink' project and select **Build Project**
- Use **Ctrl+B** from the keyboard

When starting a project build, the Progress Window will appear, and information will scroll by in the Console Window.

2.3 Download and debug application code

2.3.1 Create a debug launch configuration

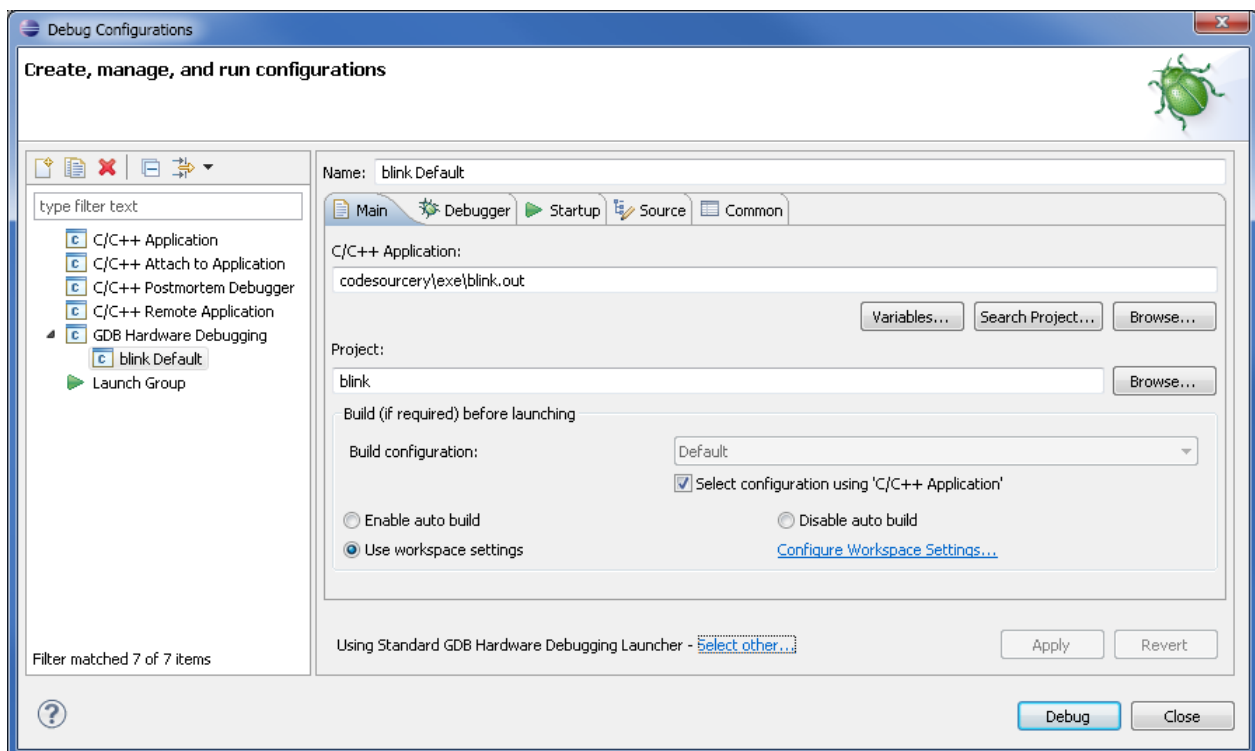
Eclipse do not come with a pre-defined debug configuration. To be able to download and debug an application, a Debug Configuration must be created.

1. From the main menu, select **Run -> Debug Configurations...**
2. **Right-click** GDB Hardware Debugging
3. Click **New**

Do the following changes in the **Main** tab:

4. Click the **Select other...** link on the bottom where it says Using GDB (DSF) Hardware Debugging Launcher
5. Select **Standard GDB Hardware Debugging Launcher**
6. Apply changes by clicking the **OK** button.

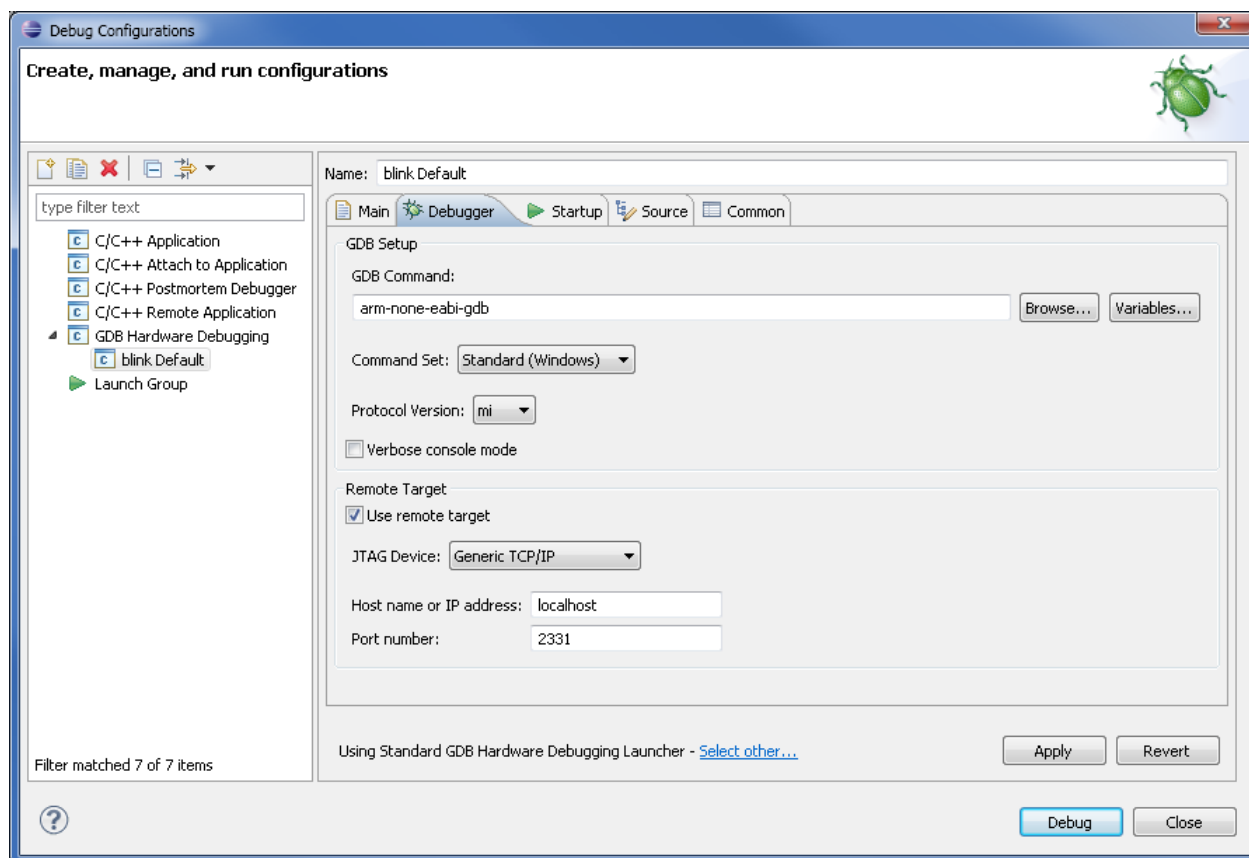
Figure 2.9. Debug configuration - Main tab



Now proceed to the **Debugger** tab.

7. Change GDB Command to **arm-none-eabi-gdb**
8. Check **Use remote target**
9. Set JTAG Device to **Generic TCP/IP**
10. Set Port number to **2331**

Figure 2.10. Debug configuration - Debugger tab



Proceed to the **Startup** tab.

11. Uncheck **Reset and Delay**

12. Uncheck **Halt**

13. In the textbox **Initialization Commands** enter:

```
set tdesc filename target-m3.xml
mon speed 4000
mon endian little
mon flash download = 1
mon flash device = EFM32TG840F32
mon reset 0
```

Ensure that the device matches your target device.

14. If you are using a newer version of the J-Link GDB Server, omit the first line above

15. Check the **Set breakpoint at** box

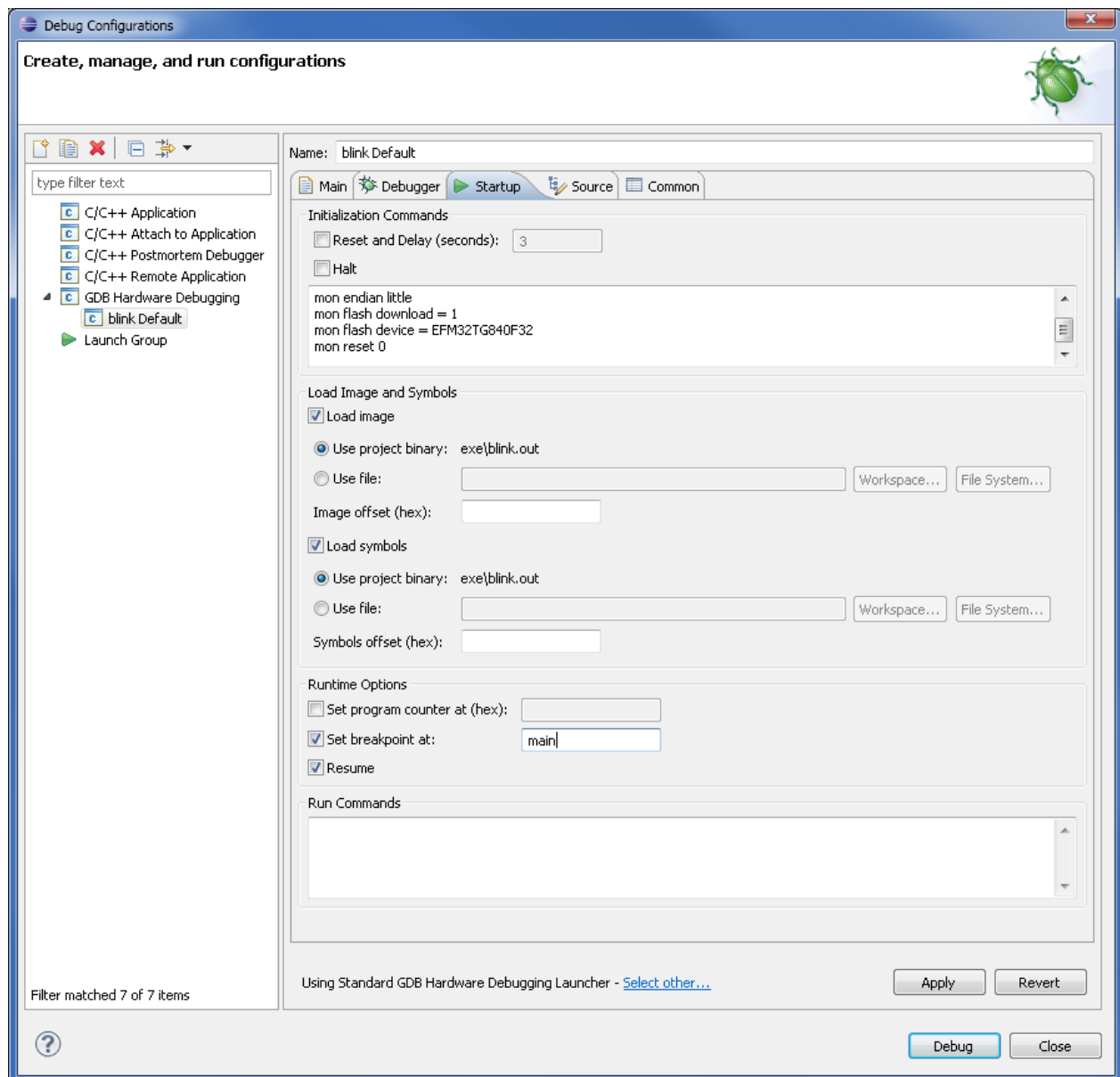
16. Enter **main** in the corresponding textbox

17. Check the **Resume** box

18. The `target-m3.xml` file is included in the zip-file that comes with this application note. It should be extracted to the root Eclipse folder.

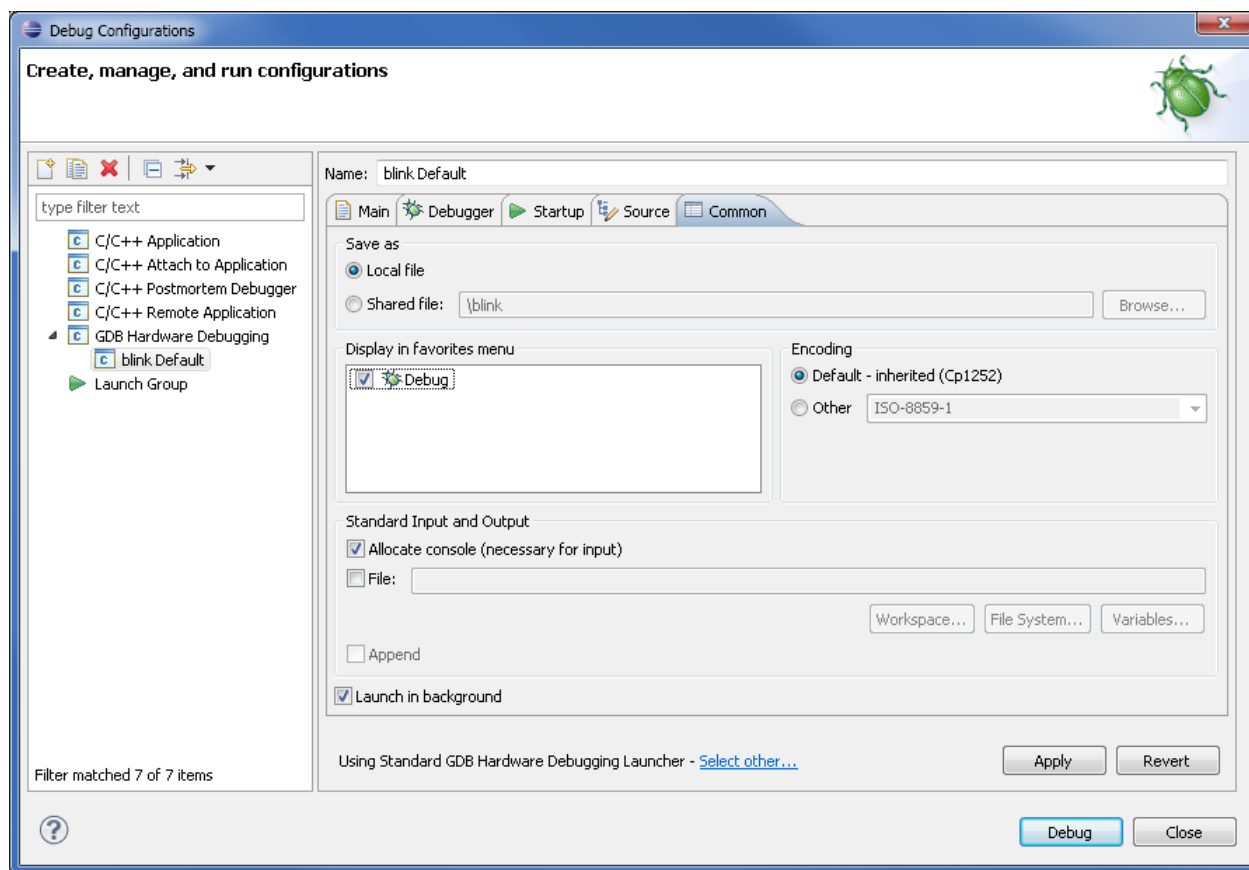
The "set desc" command applies a patch that corrects the CPU register view. This has been fixed in newer version of J-Link GDB Server. The "mon" command passes commands on to the hardware debugger itself. Please refer to SEGGER's UM08001 J-Link/J-Trace User Guide for an explanation of the initialization commands.

Figure 2.11. Debug configuration - Startup tab



Next, go to the **Common** tab

19. Check the **Debug** checkbox in the "Display in favorites menu" field.
20. Save your changes by clicking **Apply** and **Close**

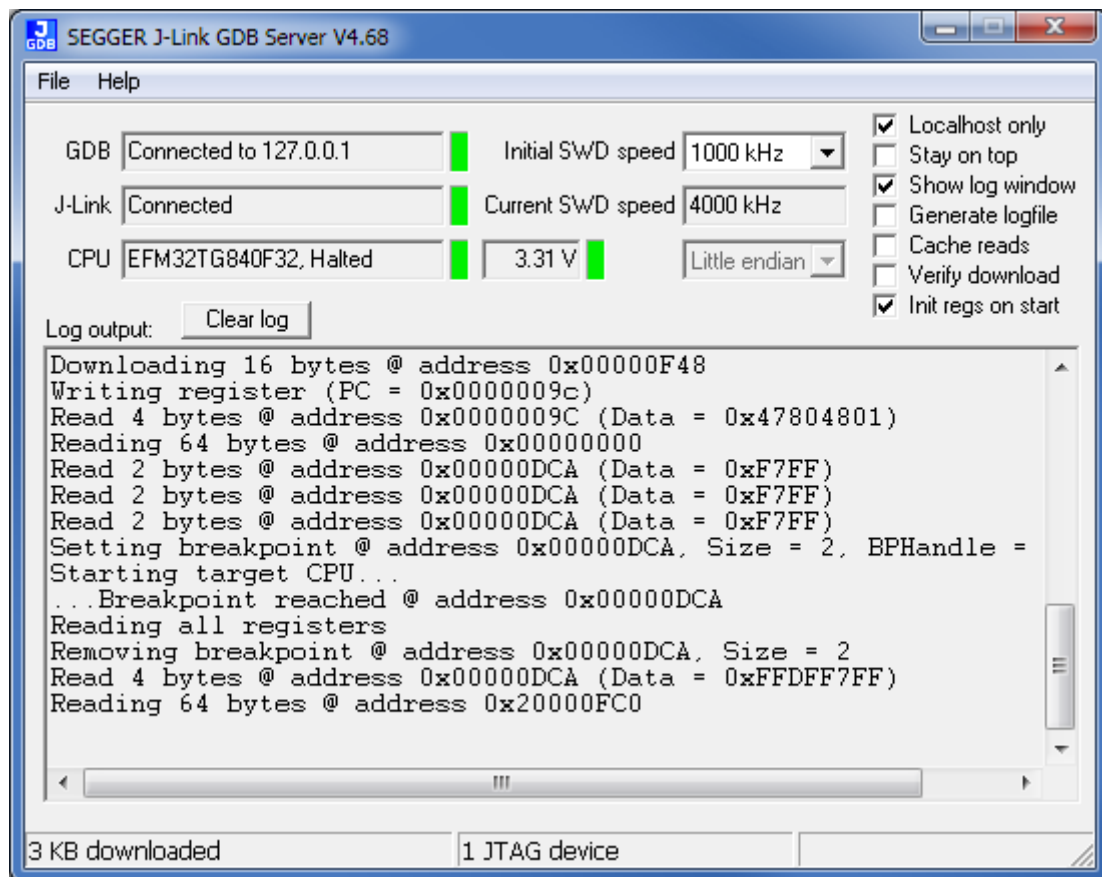
Figure 2.12. Debug configuration - Common tab

Before starting a debug session, the J-Link GDB Server must be running in the background. On the Windows Start Menu, navigate to the SEGGER J-Link ARM folder and start **J-Link GDB Server via SWD** and connect to your target.

Now you can flash your microcontroller and start debugging your code directly by clicking the **Debug** icon on the main toolbar and select **blink** in the dropdown menu.

After debugging has started, verify that the connection between Eclipse through the J-Link GDB Server to your target hardware is OK. Switch focus to the running J-Link GDB Server

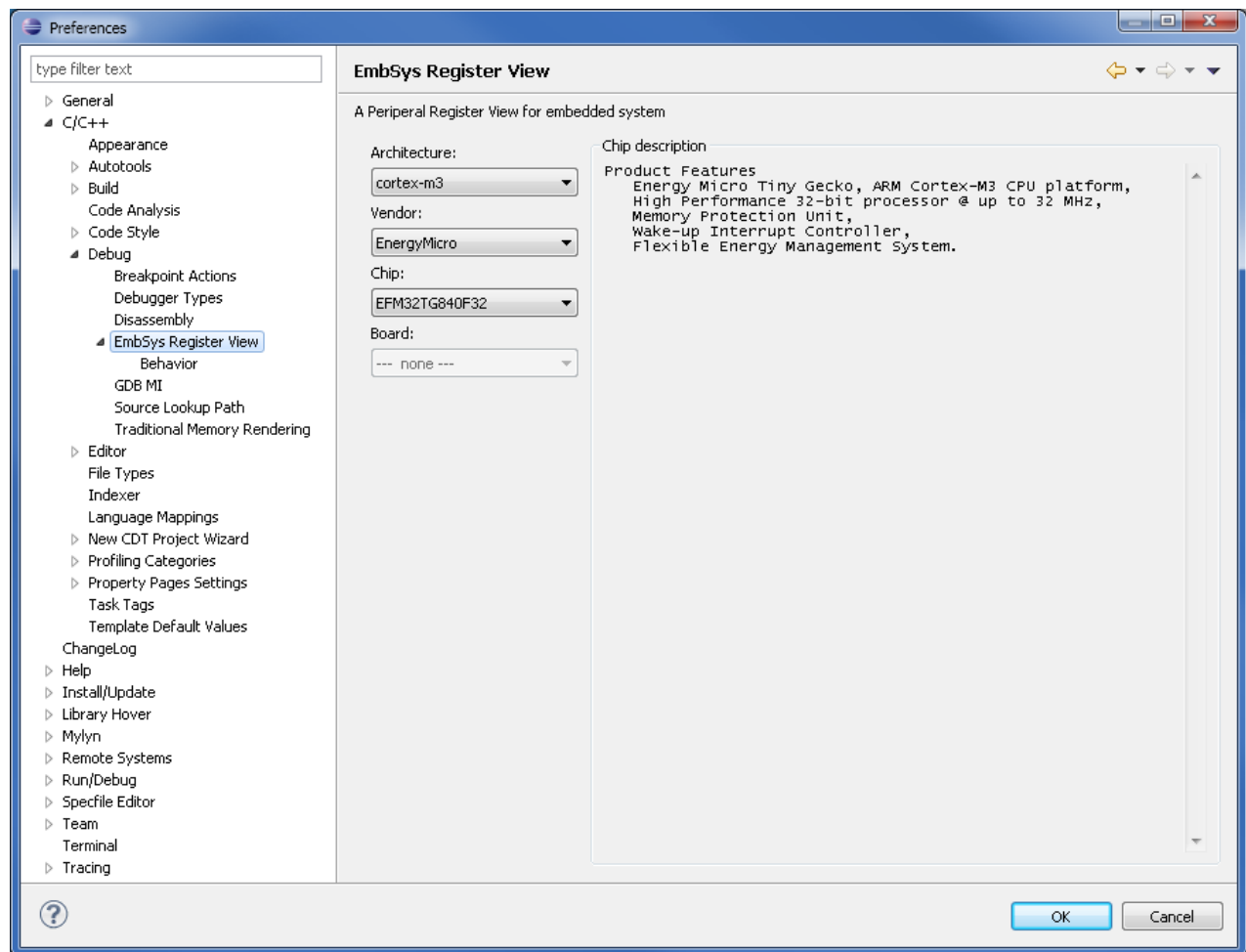
Figure 2.13. J-Link GDB Server in a debugging session



2.3.2 Embedded Systems Register View

The register viewer is a practical tool when you are debugging code for the peripherals of the microcontroller. Embedded Systems Register View (EmbSys) also contains documentation on the peripheral registers and their bitfields (as tooltips). To show the proper peripheral registers, EmbSys must be configured for the target device. To do this, follow these steps:

1. From main menu, select **Window -> Preferences**
2. In the Preferences menu, expand **C/C++ -> Debug -> EmbSys Register View**
3. Select Architecture: **cortex-m3**
4. Select Vendor: **EnergyMicro**
5. Select Device: **EFM32TG840F32**
6. Select **OK** to save your changes and return

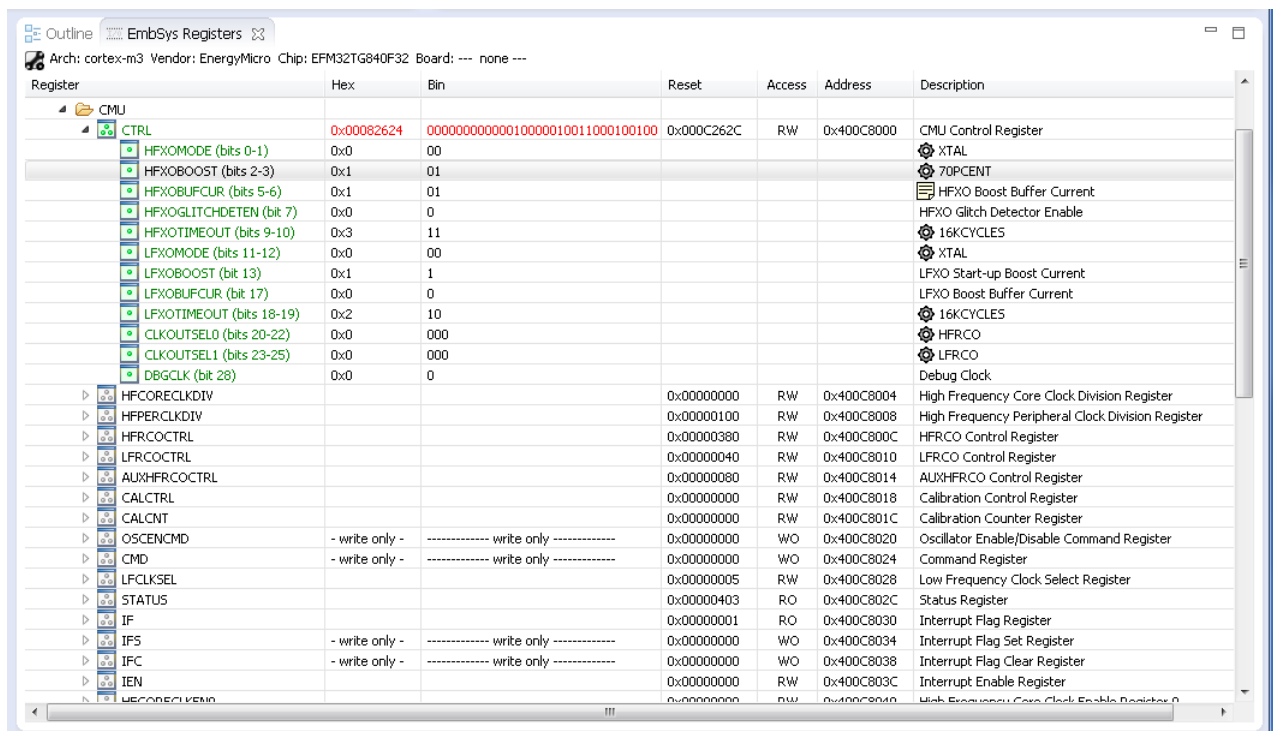
Figure 2.14. EmbSys register viewer device selection

7. Navigate to **C/C++ -> Debug -> EmbSys Register View -> Behavior**
8. Set "Number of elements shown in the dropdown List" to the maximal amount.
9. Save the changes by clicking **OK**

To show EmbSys, change to debug perspective. From main menu select

10. **Window -> Show view... -> Other...**
11. Navigate to **Debug**
12. Select **EmbSys Registers**
13. Confirm with **OK**

Figure 2.15. EmbSys register viewer in action



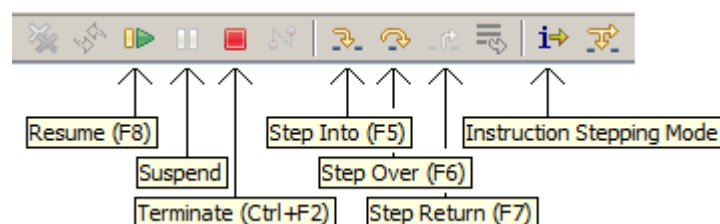
Register	Hex	Bin	Reset	Access	Address	Description
CMU						
CTRL	0x00082624	00000000000010000010011000100100	0x000C262C	RW	0x400C8000	CMU Control Register
HFXTMODE (bits 0-1)	0x0	00				XTAL
HFXTBOOST (bits 2-3)	0x1	01				70PCENT
HFXTBUFCLUR (bits 5-6)	0x1	01				HFXT Boost Buffer Current
HFXTGLITCHDETEN (bit 7)	0x0	0				HFXT Glitch Detector Enable
HFXTTIMEOUT (bits 9-10)	0x3	11				16KCYCLES
LFXTMODE (bits 11-12)	0x0	00				XTAL
LFXTBOOST (bit 13)	0x1	1				LFXT Start-up Boost Current
LFXTBUFCLUR (bit 17)	0x0	0				LFXT Boost Buffer Current
LFXTTIMEOUT (bits 18-19)	0x2	10				16KCYCLES
CLKOUTSEL0 (bits 20-22)	0x0	000				HFRCO
CLKOUTSEL1 (bits 23-25)	0x0	000				LFRCO
DBGCLK (bit 28)	0x0	0				Debug Clock
HFCORECLKDIV			0x00000000	RW	0x400C8004	High Frequency Core Clock Division Register
HFPERCLKDIV			0x00000100	RW	0x400C8008	High Frequency Peripheral Clock Division Register
HFRCCOCTRL			0x00000380	RW	0x400C800C	HFRCO Control Register
LFRCCOCTRL			0x00000040	RW	0x400C8010	LFRCO Control Register
AUXHFRCCOCTRL			0x00000080	RW	0x400C8014	AUXHFRCO Control Register
CALCTRL			0x00000000	RW	0x400C8018	Calibration Control Register
CALCNT			0x00000000	RW	0x400C801C	Calibration Counter Register
OSCENCMD	- write only -	----- write only -----	0x00000000	WO	0x400C8020	Oscillator Enable/Disable Command Register
CMD	- write only -	----- write only -----	0x00000000	WO	0x400C8024	Command Register
LFCLKSEL			0x00000005	RW	0x400C8028	Low Frequency Clock Select Register
STATUS			0x00000403	RO	0x400C802C	Status Register
IF			0x00000001	RO	0x400C8030	Interrupt Flag Register
IFS	- write only -	----- write only -----	0x00000000	WO	0x400C8034	Interrupt Flag Set Register
IFC	- write only -	----- write only -----	0x00000000	WO	0x400C8038	Interrupt Flag Clear Register
IEN			0x00000000	RW	0x400C803C	Interrupt Enable Register
HFCCOCLKEN0			0x00000000	RW	0x400C8040	High Frequency Core Clock Enable Register 0

Double click on a register to start viewing its content. Registers which you have selected get a green font. Changes in register contents are shown with red values. When hovering over a register's description column you see documentation for that register. To change a register, click on a cell in the Hex column, or the Bin column.

2.3.3 Run, Stop, Single-Step, Breakpoints

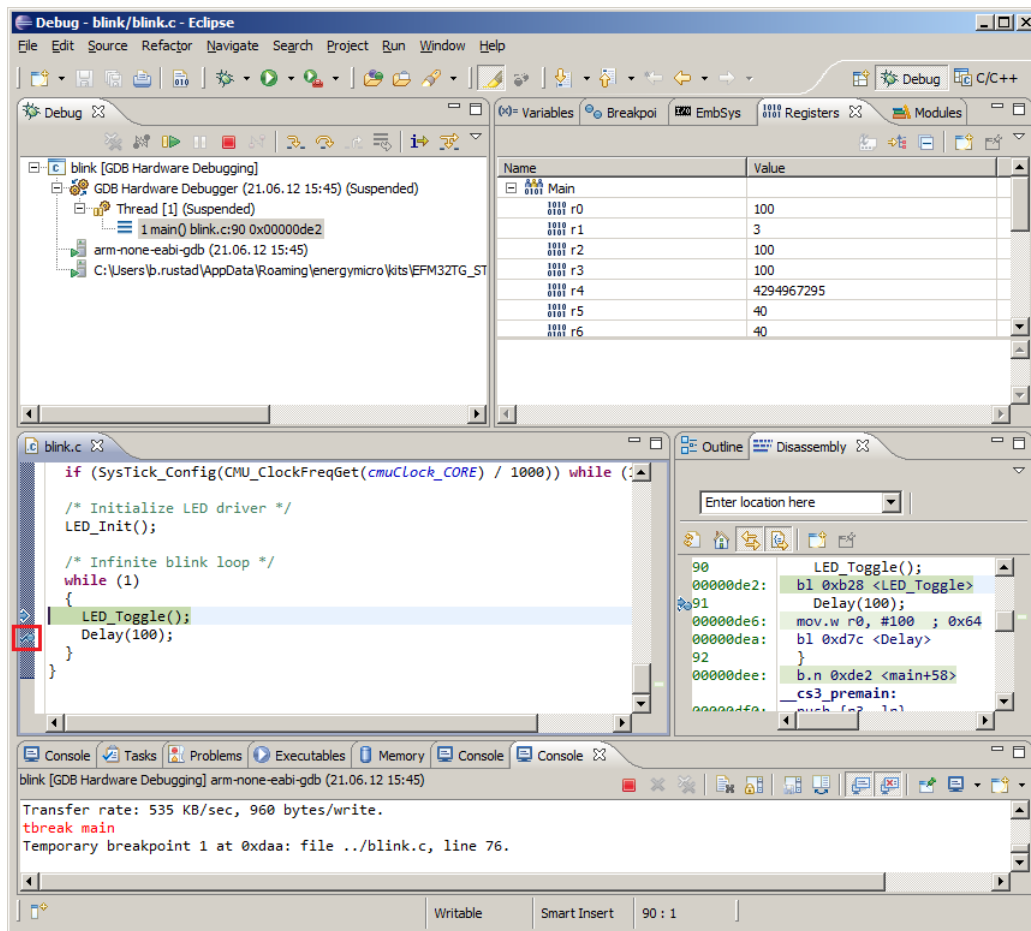
Look for the buttons shown in the figure below in the debug tab.

Figure 2.16. Debug button



These are all you need to do simple debugging. Breakpoints are set by doubleclicking in the left gutter in the source code tab. Set a breakpoint on the *Delay(100)* function call in the end of *main()* in *blink.c*, hit F8 several times and observe how the LED's on the STK/DK blink. (Hint: You must *Terminate* before reflashing and starting a new debug session. Exit and restart the GDB server if you get stuck).

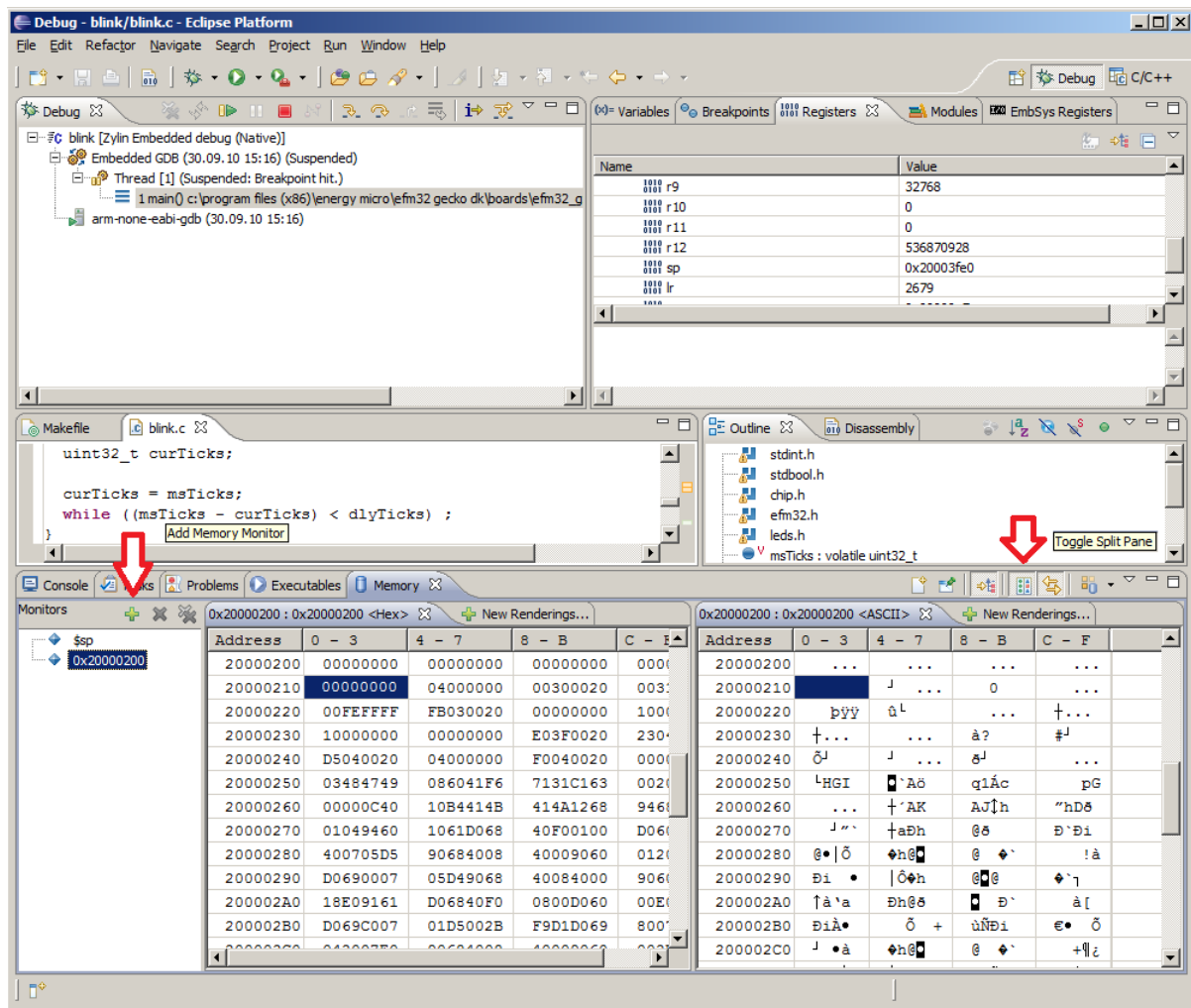
Figure 2.17. Debug with breakpoint



2.3.4 Using the Memory viewer

Eclipse's memory monitor view is a default part of the debug view. The figure below shows two active memory monitors, one at address 0x20000200 and one at current stackpointer address. The view uses the *Split Pane* functionality, which in this case give you one hexadecimal rendering, and one ASCII rendering. New monitors are added by clicking the green plus sign as indicated.

Figure 2.18. Eclipse Memory Monitor



2.4 Tweaking Eclipse

2.4.1 Disabling parts of the Code Analysis feature

Code Analysis is a feature of Eclipse that tries to spot errors in your code while you are writing it, but it sometimes gets in your way, and is not always so good at finding definitions hidden in other files.

You can disable some or all of the *Code Analysis* warnings in *Window -> Preferences*, under *C/C++ -> Code Analysis*.

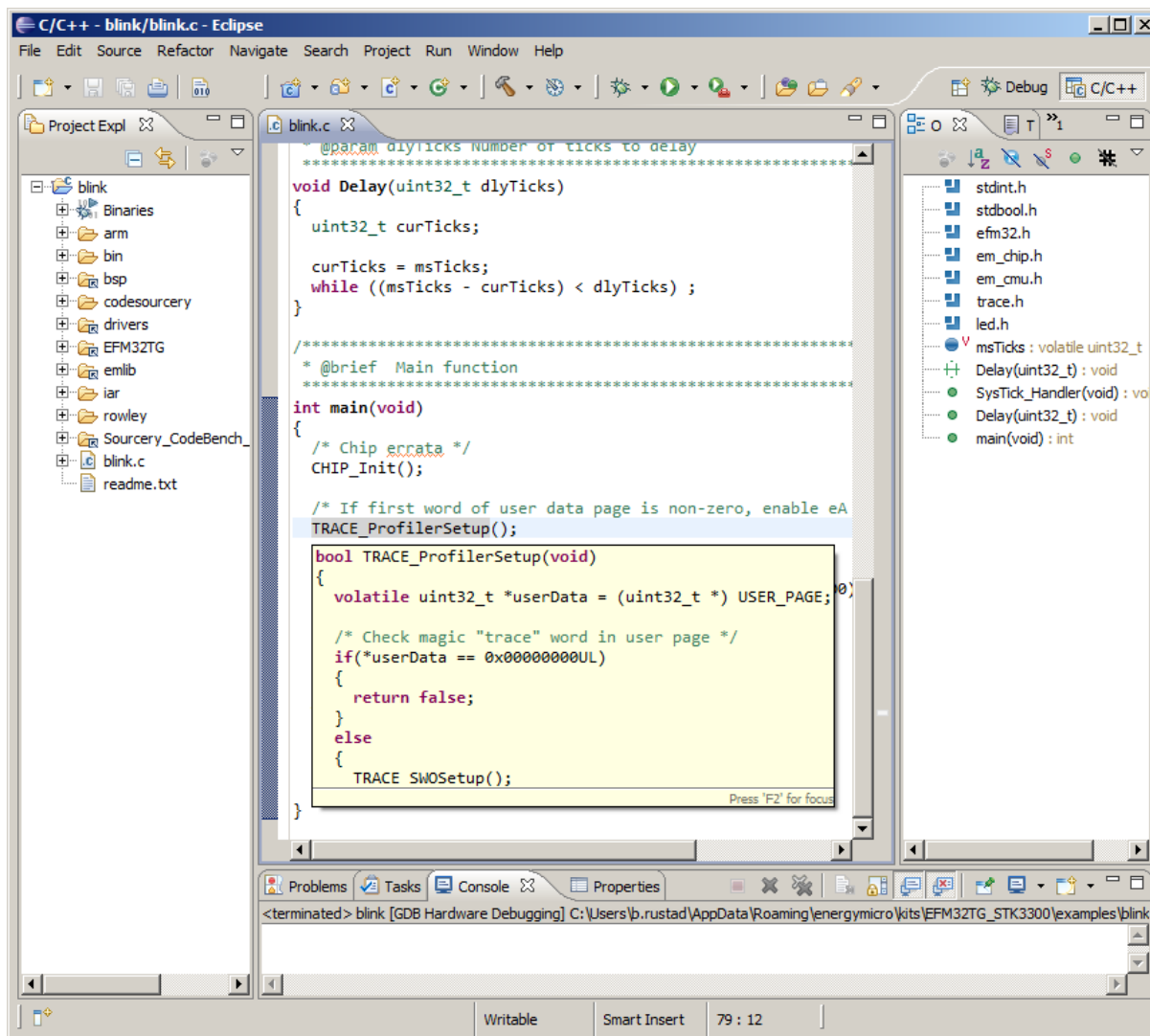
2.4.2 Telling Eclipse about the external libraries

To take advantage of more of Eclipse's advanced IDE-features you can add the external libraries as linked folders in your project. Do this by right-clicking the project name, and selecting *New -> Folder*. Under *Advanced >>*, select *Link to alternate location* and browse to the library you want to link to.

Add *bsp* and *drivers* from the directory corresponding to your kit. Also add the *emlib* folder and the folder corresponding to your device under *Device/EnergyMicro*. Lastly, add a link to the install location of Sourcery CodeBench, and Eclipse will now be able help you look up function definitions etc.

When you have done this, Eclipse will show you function definitions when hovering over function calls, and by right-clicking you can jump to the file where the function is defined.

Figure 2.19. Hovering a function call gives you the beginning of its definition



3 Eclipse summary

This has just been an introduction to the basic steps of installing, configuring and using Eclipse as a code development environment for writing programs for EFM32 microcontrollers. Keep in mind that Eclipse is an advanced tool with hundreds of configuration options. The reader is encouraged to experiment further to finetune Eclipse. Use a search engine on the internet and you will find lots of articles, tutorials and books on using Eclipse.

4 Revision History

4.1 Revision 1.13

2013-09-03

New cover layout

4.2 Revision 1.12

2013-05-14

Replaced the included zip-file with the correct version

4.3 Revision 1.11

2013-05-08

Added info on how to resolve symbols

Simplified the included zip-file

Debug Configurations: Removed Reset and Delay from Startup

Debug Configurations: target-m3.xml not needed for newer versions of J-Link GDB Server

Debug Configurations: Set device reset type to 0 which resets both CPU and peripheral registers

Debug Configurations: Added Resume

4.4 Revision 1.10

2012-06-25

New versions of Eclipse, Sourcery CodeBench and EmbSysRegView.

Recommend installing Simplicity Studio.

Use C/C++ GDB Hardware Debugging plugin instead of Zylín.

Use GDB for programming directly instead of eA Commander.

New target-m3.xml.

4.5 Revision 1.04

2011-08-30

Fixed Zylín update link.

4.6 Revision 1.03

2011-06-28

Fixed typo regarding EMEclipseSupport.zip to an0023_efm32_eclipse_toolchain.zip renaming.

4.7 Revision 1.02

2011-03-24

Changed the name of the zip file EMEclipseSupport.zip to an0023_efm32_eclipse_toolchain.zip on section 1.2.4

4.8 Revision 1.01

2010-11-22

Added comment about the necessity of having Java installed before installing Eclipse.

4.9 Revision 1.00

2010-11-10

Initial revision.

A Disclaimer and Trademarks

A.1 Disclaimer

Silicon Laboratories intends to provide customers with the latest, accurate, and in-depth documentation of all peripherals and modules available for system and software implementers using or intending to use the Silicon Laboratories products. Characterization data, available modules and peripherals, memory sizes and memory addresses refer to each specific device, and "Typical" parameters provided can and do vary in different applications. Application examples described herein are for illustrative purposes only. Silicon Laboratories reserves the right to make changes without further notice and limitation to product information, specifications, and descriptions herein, and does not give warranties as to the accuracy or completeness of the included information. Silicon Laboratories shall have no liability for the consequences of use of the information supplied herein. This document does not imply or express copyright licenses granted hereunder to design or fabricate any integrated circuits. The products must not be used within any Life Support System without the specific written consent of Silicon Laboratories. A "Life Support System" is any product or system intended to support or sustain life and/or health, which, if it fails, can be reasonably expected to result in significant personal injury or death. Silicon Laboratories products are generally not intended for military applications. Silicon Laboratories products shall under no circumstances be used in weapons of mass destruction including (but not limited to) nuclear, biological or chemical weapons, or missiles capable of delivering such weapons.

A.2 Trademark Information

Silicon Laboratories Inc., Silicon Laboratories, the Silicon Labs logo, Energy Micro, EFM, EFM32, EFR, logo and combinations thereof, and others are the registered trademarks or trademarks of Silicon Laboratories Inc. ARM, CORTEX, Cortex-M3 and THUMB are trademarks or registered trademarks of ARM Holdings. Keil is a registered trademark of ARM Limited. All other products or brand names mentioned herein are trademarks of their respective holders.

B Contact Information

Silicon Laboratories Inc.

400 West Cesar Chavez

Austin, TX 78701

Please visit the Silicon Labs Technical Support web page:

<http://www.silabs.com/support/pages/contacttechnicalsupport.aspx>

and register to submit a technical support request.

Table of Contents

1. Prerequisites	2
1.1. A note on versions	2
1.2. Installing the tools	2
2. Working with Eclipse	8
2.1. Create a project	8
2.2. Build the code	13
2.3. Download and debug application code	13
2.4. Tweaking Eclipse	22
3. Eclipse summary	24
4. Revision History	25
4.1. Revision 1.13	25
4.2. Revision 1.12	25
4.3. Revision 1.11	25
4.4. Revision 1.10	25
4.5. Revision 1.04	25
4.6. Revision 1.03	25
4.7. Revision 1.02	25
4.8. Revision 1.01	26
4.9. Revision 1.00	26
A. Disclaimer and Trademarks	27
A.1. Disclaimer	27
A.2. Trademark Information	27
B. Contact Information	28
B.1.	28

List of Figures

1.1. Energy Micro folder structure	3
1.2. J-Link GDB Server GUI	4
1.3. GDB Hardware Debugging plugin installation	6
1.4. Embedded Systems Register View plugin installation	7
2.1. The Eclipse Workbench View	8
2.2. Create a new project in Eclipse	9
2.3. Eclipse Project Properties - Build	10
2.4. Eclipse Project Properties - Discovery Options	10
2.5. Eclipse Project Properties - Build Settings	11
2.6. Makefile location in Eclipse Project Explorer	12
2.7. Add directory paths	13
2.8. Eclipse Project Properties - Paths and Symbols	13
2.9. Debug configuration - Main tab	14
2.10. Debug configuration - Debugger tab	15
2.11. Debug configuration - Startup tab	16
2.12. Debug configuration - Common tab	17
2.13. J-Link GDB Server in a debugging session	18
2.14. EmbSys register viewer device selection	19
2.15. EmbSys register viewer in action	20
2.16. Debug button	20
2.17. Debug with breakpoint	21
2.18. Eclipse Memory Monitor	22
2.19. Hovering a function call gives you the beginning of its definition	23

List of Tables

1.1. Tool version information 2

silabs.com

