Energy Micro University

UM003 - Setting Up Development Environments

In this lesson the workflow for developing for Energy Micro's Energy Friendly Microcontrollers (EFMs) will be presented. First, a general overview of parts of the toolchain will be presented, before a closer look at a Linux based toolchain. Lastly, some of the Integrated Development Environments (IDEs) available on Windows will be listed.

You will get an introduction to:

- **Compilers**
- **Linkers**
- **Flashing the MCU**
- **Running Programs**
- **Debugging**
- **Linux Toolchains**
- **Windows Toolchains**

This lesson includes:

- **This PDF document**

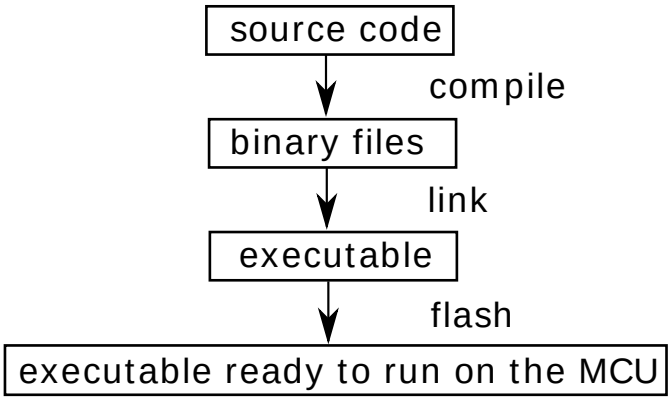
1 Introduction

A number of components make up the toolchain required to develop for MCUs. These are often hidden in a single Integrated Development Environment (IDE), where the user can write and debug software from within a single application. Several steps are often combined into a push of a button. This makes them easy to use, but it is also helpful to know a little about what is going on behind the scenes, which will be presented in the next chapter. Afterwards development on Linux will be presented before the various Windows IDEs. Setting up things on Linux is a bit more elaborate, but the payoff is a better understanding of the different components - and the software is free. The Windows solutions range from those available on Linux to robust commercial packages, that offers ease of use as well as high quality compilers, such as IAR and Keil.

2 Toolchain

Regardless if your toolchain consist of one large application or many small, the main steps performed are the same. A short introduction to the components and the terminology follows.

Figure 2.1. An overview of some of the development steps.



The compilation process is fairly straight-forward. First you write your c-code in a texteditor (like Vim or WordPad) and save it with the ".c" extension. Then the compiler converts your code file into an object (".o"). The linker then combines several objects into an executable (".out" or ".axf") This executable can then be converted into pure binary and flashed onto the microcontroller.

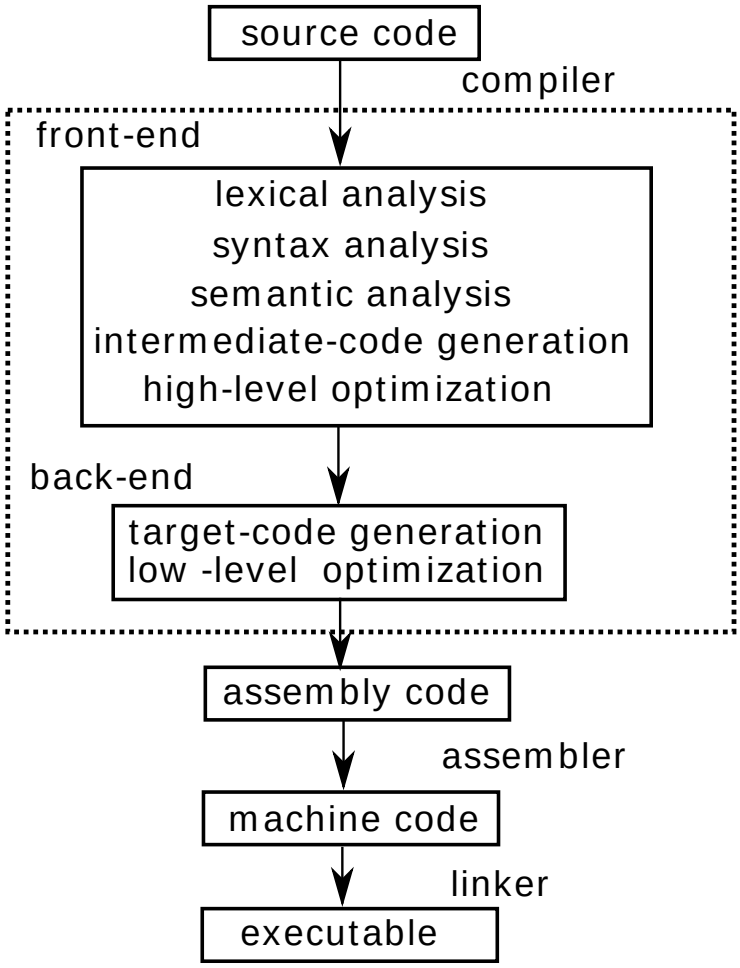
2.1 Text Editors

Text editors are what the source code is written in. Text editors range from rudimentary ones, like Notepad, to advanced editors like Vim and Emacs. Some are standalone while others are embedded in IDEs.

2.2 Compilers

Compilers often consist of many steps, including optimization at several levels. Some of the most common parts of a compiler will be introduced here. Compilers are often divided into two parts: the front-end and the back-end. This is convenient in order to reuse parts of the compiler for different programming languages and different target platforms. A front-end for the C language can for instance be used for both PCs and different MCUs, but the back-end has to be changed to support the target hardware.

Figure 2.2. Compilation steps.



2.2.1 Lexical, syntactic and semantic analysis

The lexical analyser give meaning to the input text by recognizing things like variables, numbers and keywords (for, if, while, etc.) The syntax analysis then checks if the input conforms to the grammar of the language, i.e. if the placement of the different words is correct. The semantic analysis checks if the statements are meaningful, e.g. that the types of an assignment are compatible.

2.2.2 Code generation and optimizations

By producing intermediate code, different front-ends and back-ends can be combined. High-level optimizations are performed on this intermediate representation - it is independent from the target architecture. The intermediate representation is then translated into the target assembly language and target specific optimizations are performed - here the compiler can tailor the program to best fit the hardware it will run on.

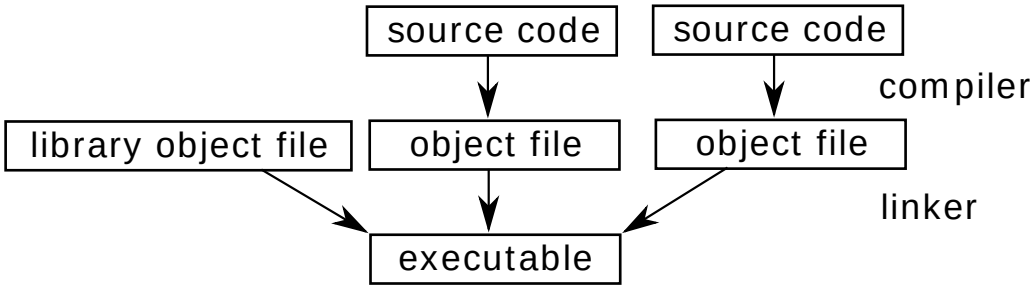
2.2.3 Assembler

The assembler takes the assembly code and translate it into machine code, i.e. the binary instructions the target machine can read. This translation is 1-1, i.e. the two representations are equivalent. Therefore the inverse process, called disassembly, can be performed when debugging.

2.3 Linker

Each source file is usually compiled into its own file called an object file. The linker then collects these into a single binary executable, which is the finished file the target machine can run. When static linking is used, external library routines are included in the executable by the linker, the result is a standalone binary that can be run on the target machine.

Figure 2.3. Each source code file produce an object code file which are combined into one executable by the linker.



2.4 Flash

The step of downloading the executable to the MCU is called flashing the MCU. This is because the program is stored in the Flash memory of the MCU. Energy Micro's energyAware Commander is one solution to accomplish this step.

2.5 Release/Debug

When compiling code, you have the options of a release or debug target. The debug version allows easier debugging of the program at runtime by including debug symbols as well as performing less optimization. The debug symbols make it possible to view the original source code for the instructions that are running, while the reduced optimization makes it easier to understand the disassembly, since the most code obfuscating optimizations have not been performed. The release version on the other hand aims to produce the best performing code possible.

After the executable is downloaded to the MCU, it is ready to run. Program execution can be triggered by resetting the device. Debugging is more elaborate: it requires a debug connection to a PC and appropriate software. The payoff is the ability to see the state of the device as it is running the code.

A debug session might starts with pausing the execution right before the execution of `main()`. This is done by setting a breakpoint at the line of `main()`. Breakpoints can be set on multiple lines in the code. There are usually several ways to resume execution. The first is to continue execution until a new breakpoint is reached or the execution is paused by the user. Instructions can also be executed in groups corresponding to a single line in the source code. This is called stepping and has various forms when executing a function call. You can usually choose between executing the function as one step or to step into the function. Once there, it is possible to step line by line or to step out of the function, i.e. execute the remaining instructions in the function. When the execution is paused, you can check the state of the program; what line of the source code is the next to be run, previous and upcoming instructions that the CPU will execute as well as the contents of registers, RAM and Flash. Bugs can usually be identified by setting breakpoints at appropriate places and to assert that the contents of registers and RAM/Flash are what they should be at those points in the program.

2.6 Energy Profiling

If application requires low power consumption it is useful to use a energy profiler. An energy profiler displays not only the time used in each function as a regular profiler would, it also shows the power consumption. This is useful for the programmer to identify the power critical parts of the code to energy optimize. Energy Micro provides the energyAware Profiler for this purpose.

3 Working with EFMs

The steps introduced so far are largely independent of MCU vendor and even computer platform, some more details on working with Energy Micro's Energy Friendly Microcontrollers are in order.

3.1 CMSIS

The Cortex Microcontroller Software Interface Standard (CMSIS) was developed by ARM to offer a vendor independent software interface to the MCUs using the Cortex architecture. The standard includes how to use #defines and structs to simplify register manipulation.

3.2 emlib

The emlib is written to ease the use of features specific to Energy Micro's EFM32 MCUs. This includes configuring peripherals and initiating the different sleep modes.

3.3 J-Link

SEGGER's J-Link interface is used to communicate between the kits and PCs.

3.4 SWD

Serial Wire Debug (SWD) is used to debug the MCUs. Among other things it offers access to reading and writing register values at runtime.

3.5 EABI

The Embedded-Application Binary Interface (EABI) is a standard specifying the layout of object files and how to use runtime resources like registers and the stack. This is for instance useful because it allows code compiled by different compilers to be linked together as long as they both comply to the same EABI. ARM specifies the standard used by the Cortex architecture and hence the EFM32 MCUs.

4 Linux Toolchain

In this chapter you will learn how to set up a free toolchain on Linux. The Operating System (OS) of choice when writing this guide was Ubuntu 11.04, but the tools should be distribution independent. The Tiny Gecko (TG) Starter Kit (STK) with the TG840F32 MCU was used as the target device, but the process is similar for other EFM32s.

Table 4.1. Software Versions (Linux)

Software	Version
Ubuntu	11.04
SEGGER J-Link	422a
Energy Micro energyAware Commander	2.20
Sourcery CodeBench Lite Edition	2011.03
Eclipse	3.5.2
Zylin Embedded CDT	4.16.1
Eclipse Embedded Systems Register View	0.1.9

4.1 J-Link

First we will install the J-Link software, which establish the connection between the PC and the STK. Get the beta software version for Linux from **http://www.segger.com/cms/jlink-software.html**

Extract it into e.g `/opt/jlink/JLink_Linux_V422a` and follow the instructions provided in the README file. (The syntax on Ubuntu is slightly different: `$ sudo apt-get install libusb-1.0-0` was used for this tutorial.) To see if the installation was successful connect the STK and run the GDB server (version 422a was used for this tutorial).

```
opt/jlink/JLink_Linux_V422a/JLinkGDBServer -if SWD
```

The `-if SWD` option sets the server to use Single Wire Debug (SWD) rather than the default (JTAG). Instead of writing the full path the server every time, the JLink directory can be added to the `PATH` by adding

```
export PATH=$PATH:/opt/jlink/JLink_Linux_V422a
```

to your `~/.bashrc` file and running

```
$ source ~/.bashrc
```

in order to update the settings for the current terminal, the server can now be run with

```
$ JLinkGDBServer -if SWD
```

If you want to reduce the required typing further, you can add an alias to your `~/.bashrc` file

```
jlgdbs='JLinkGDBServer -if SWD'
```

after updating using `source` again, you can run the server with

```
$ jlgdbs
```

The output should be something like the following

Example 4.1. JLinkGDBServer output when connected to a Tiny Gecko STK

```
SEGGER J-Link GDB Server V4.22

JLinkARM.dll V4.22 (DLL compiled Apr  5 2011 13:54:52)

Listening on TCP/IP port 2331

J-Link connected
Firmware: Energy Micro EFM32 compiled May 12 2011 10:25:44
Hardware: V7.00
S/N: 440000008
Feature(s): GDB
```

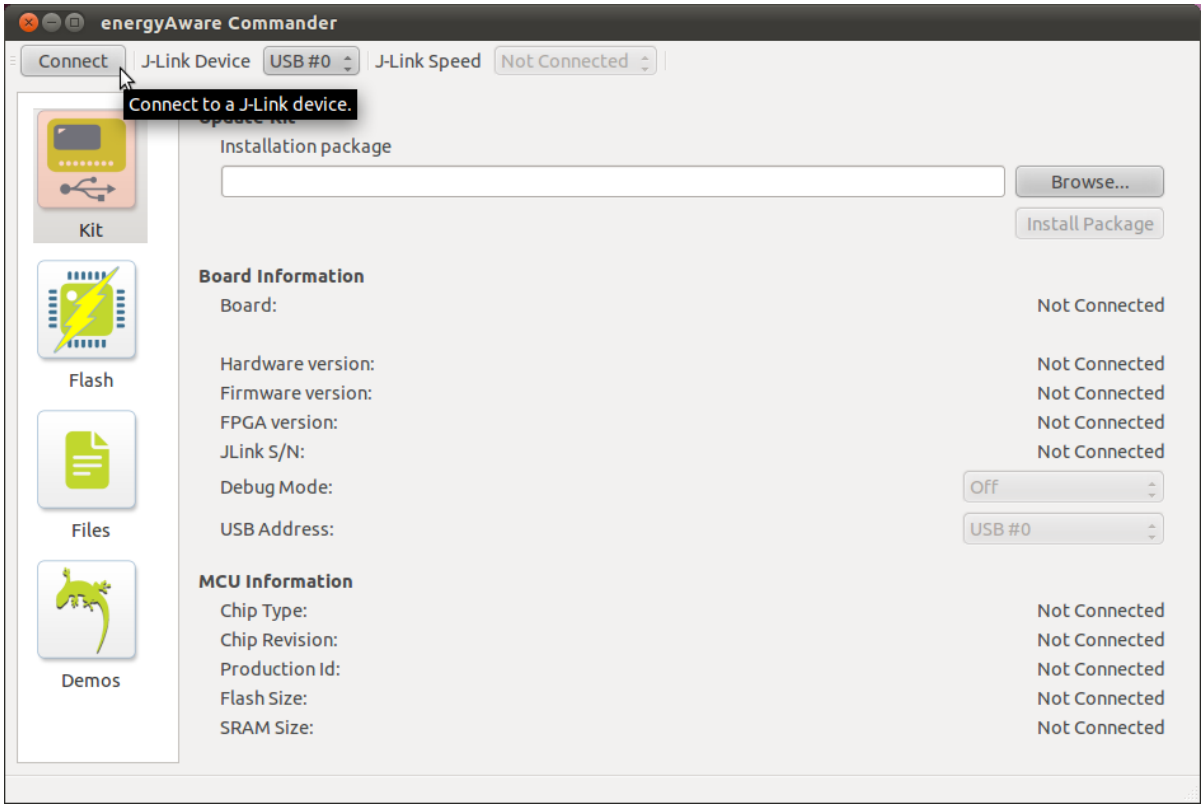
4.2 energyAware Commander

Now that we have established a connection with the kit, it is time to get the energyAware Commander from **www.energymicro.com/downloads/software**. The energyAware Commander will be used to flash the STK with the binaries we compile. The STK can only communicate with one program at the time, so the GDB server has to be shut down before running the eACommander with

```
$ ./eACommander
```

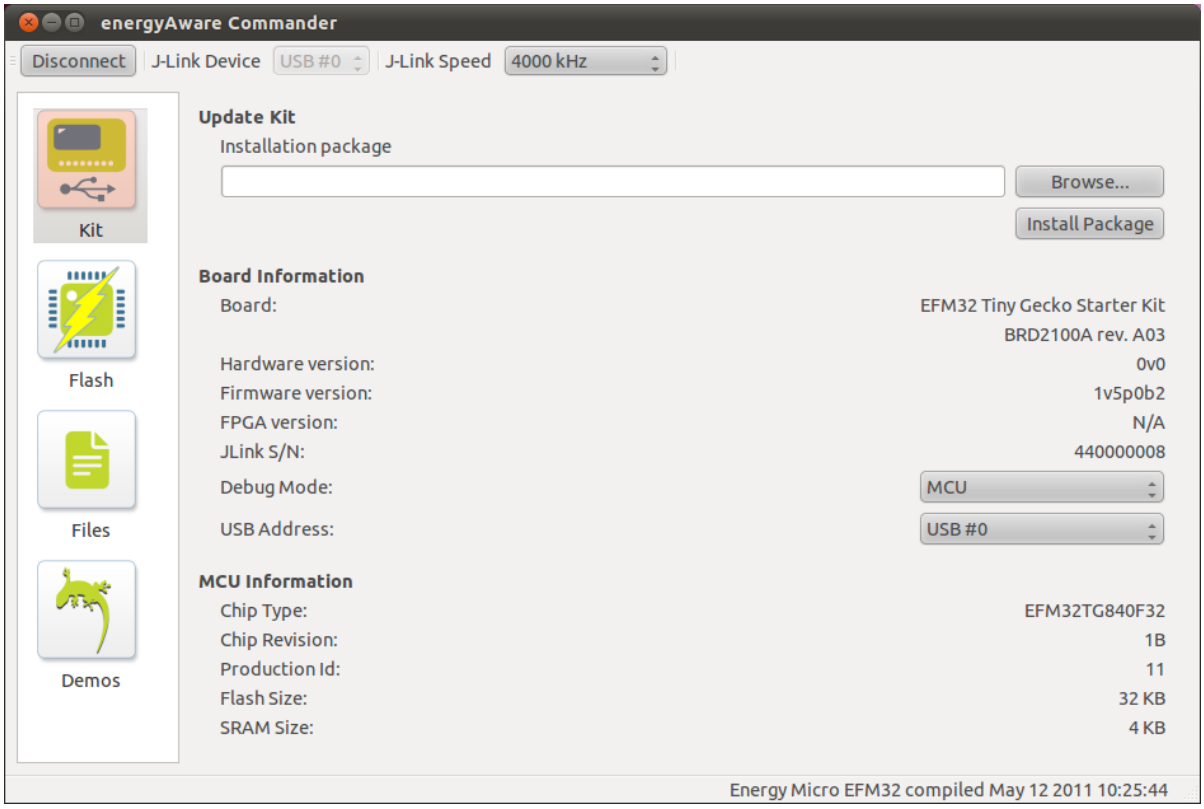
Connect to the STK by pressing the connect button in the upper left corner.

Figure 4.1. Connecting to the kit.



When connected, information about STK will be displayed.

Figure 4.2. Information about the kit.



Navigate to Demos (menu on the left) and run the Blink demo. The LED marked USER LED should be blinking. Binaries can be downloaded to the EFM32 through the Flash interface in the menu on the left.

4.3 Codesourcery WorkBench Lite Edition

Now that we can download binaries to the STK, it is time to try and compile our own. For this we will use the Codesourcery WorkBench Lite Edition, which is a command line interface based on the GNU tool chain. In particular, it has the GCC compiler and the GDB debugger. Get the software from

http://www.codesourcery.com/sgpp/lite/arm/portal/subscription3053

version 2011.03 was used for this tutorial. Get the TAR archive – extract it to e.g. /opt/codesourcery/ and create the following alias in your ~/.bashrc file.

```
export PATH=$PATH:/opt/codesourcery/arm-2011.03/bin'
```

To check if we are able to compile, download the energyAware Starter Kit (STK) Board Support Library and Example Code as well as EFM32 CMSIS from **www.energymicro.com/downloads/software**. Extract the files to ~/efm/. You should now have three folders called boards, CMSIS and emlib in your ~/efm folder. Move to the Codesourcery version of the blink example

```
$ cd ~/efm/boards/EFM32GG_STK3300/examples/blink/codesourcery
```

In the Makefile, change the LINUXCS to point to where you placed Codesourcery

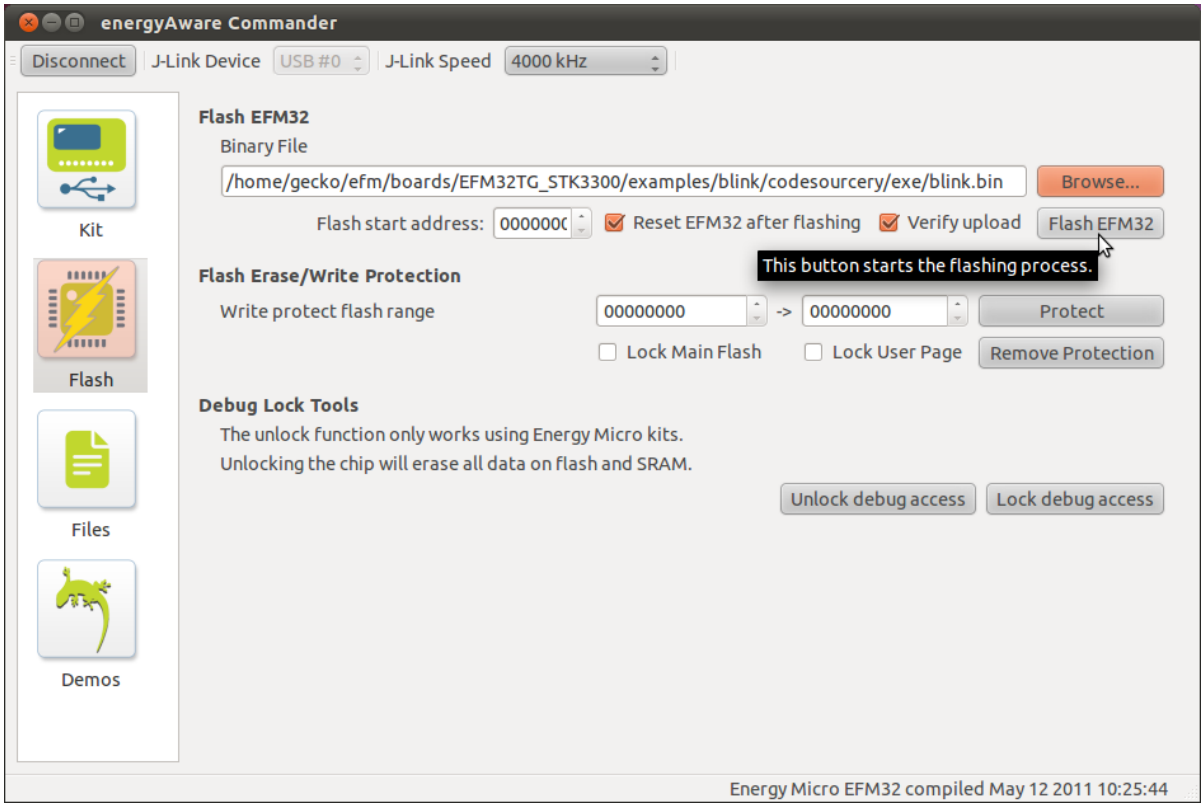
```
LINUXCS    = /opt/codesourcery/arm-2011.03/
```

Compile using the linux program make, which executes the tasks specified in the Makefile

```
$ make -f Makefile.blink
```

Upload the resulting binary ~/efm/boards/EFM32GG_STK3300/examples/blink/codesourcery/exe/blink.bin with the energyAware Commander.

Figure 4.3. Flashing the MCU.



4.4 Debugging using GDB

Close eACommander if it is running and start the GDB server

```
$ JLinkGDBServer
```

In order to start GDB correctly create the file ~/efm/gdbinit which should contain

```
target remote :2331
set tdesc filename ../CortexCpuRegs/target.xml
set mem inaccessible-by-default off
set remote memory-read-packet-size 1200
set remote memory-read-packet-size fixed
mon speed 4000
mon endian little
mon reset 1
tbreak main
cont
```

Edit `~/.bashrc` to include the alias

```
alias agdb='arm-none-eabi-gdb -x ~/efm/gdbinit'
```

`arm-none-eabi-gdb` is in the same directory as arm-none-eabi-gcc and was therefore made available by extending the PATH above. The `-x ~/efm/gdbinit` is to initialize the debugger correctly. Open a new terminal and move to the blink exe folder

```
$ cd ~/efm/boards/EFM32GG_STK3300/examples/blink/codesourcery/exe
```

Run GDB

```
$ agdb blink.out
```

Figure 4.4. A GDB session with GDB on the right and the server on the left.

Now you can use GDB as usual, for instance, try to continue

```
(gdb) c
```

To break press Ctrl+c. To single step

```
(gdb) s
```

Execute next statement

```
(gdb) n
```

To see the content of the CPU registers

```
(gdb) info registers
```

To see the source code for upcoming instructions

```
(gdb) list
```

To see the disassembly for recent and upcoming instructions

```
(gdb) disassemble
```

To see the stack

```
(gdb) info stack
```

Get help

```
(gdb) help
```

4.5 Eclipse (Optional)

If an IDE is preferred over a text editor, makefiles and GDB, then Eclipse is a good alternative. Application Note 23 from **www.energymicro.com/downloads/application-notes** explains how to set up Eclipse on Windows. The process is similar on Linux, but some tips will be outlined here. Get Eclipse IDE for C/C++ Linux Developers from eclips.org or, in Ubuntu, type

```
$ sudo apt-get install eclipse-cdt
```

Start Eclipse (version 3.5.2 used here)

```
$ eclipse
```

Set ~/efm/boards/EFM32GG_STK3300/examples/ as your workspace when prompted. To find out where to copy the contents of EmbSys and Zylin (from the zip file of an0023) type

```
$ whereis eclipse
```

Ubuntu 11.04 gave the following result

```
eclipse: /usr/bin/eclipse /etc/eclipse.ini /usr/lib/eclipse
/usr/share/eclipse /usr/share/man/man1/eclipse.1.gz
```

/usr/share/eclipse is the folder of interest. Follow an0023 to complete the setup of Eclipse. It is not necessary to change make into cs-make, the usual GNU Make will suffice. Since we have already used the Makefile to compile a project, no further changes has to be made to it. If Eclipse cannot find programs that should be available in the PATH or by aliases, try to use full paths instead like

```
/opt/codesourcery/arm-2011.03/bin/arm-none-eabi-gdb
```

and add the initialize command as explained in an0023. Versions does not seem to be a problem on Linux. Both the versions used in an0023 and the easiest to get hold of on Ubuntu 11.04 at the time of writing (listed in Table 4.1 (p. 7)) worked fine.

5 Windows Toolchain

In this chapter you will learn how to set up a free toolchain on Windows. The Operating System (OS) of choice when writing this guide was Windows 7 Ultimate 32-bit. The Giant Gecko (GG) Starter Kit (STK) with the GG990F1024 MCU was used as target device, but the process is similar for other EFM32s.

5.1 Simplicity Studio

First we will download Simplicity Studio which is the innovative solution which helps you to always have the latest documentation, examples, firmware and software. Go to the Energy Micro homepage (**http://www.energymicro.com**), click on Downloads, and download Simplicity Studio.

Figure 5.1. Download Simplicity Studio.

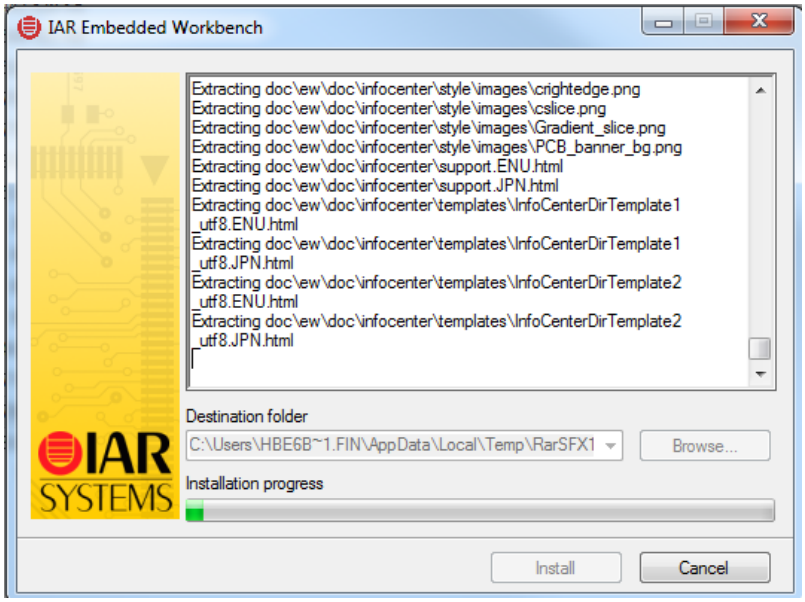
When the download is complete, click run, and follow the intructions.

Lauch Simplicity Studio, click on Add/Remove and install everything. For information about how to set up the energyAware Commander, see the instructions given in chapter 4.2.

5.2 IAR Embedded Workbench

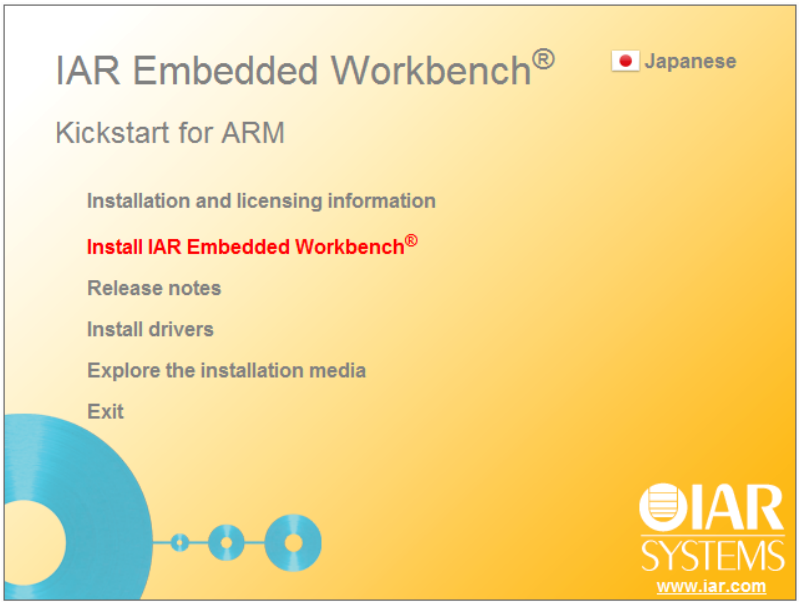
IAR Embedded Workbench is an commercial IDE. The KickStart, which has a 32kb code size limit, is shipped with the Giant Gecko STK. To install IAR, use the CD included in the Giant Gecko Starter Kit.

Figure 5.2. Install IAR.



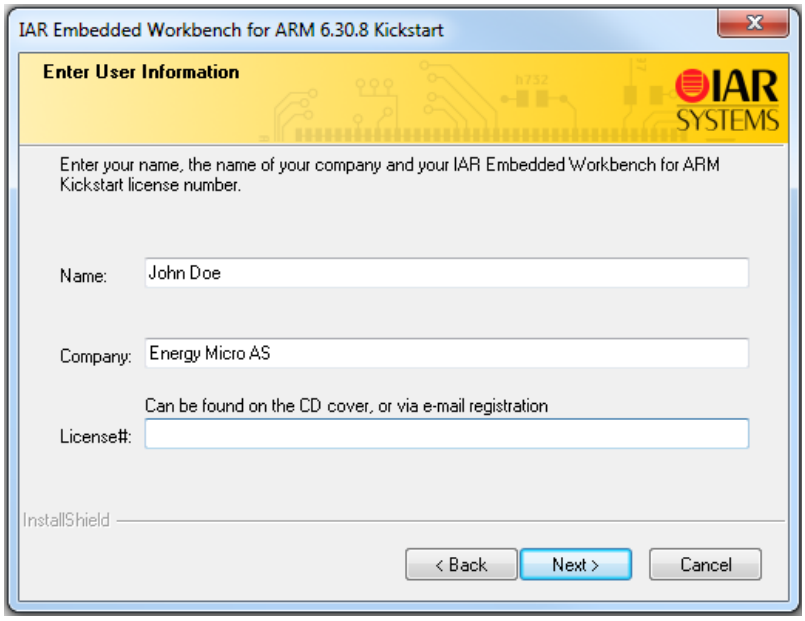
When the installation is complete, click on **Install IAR Embedded Workbench**.

Figure 5.3. Click on Install IAR Embedded Workbench.



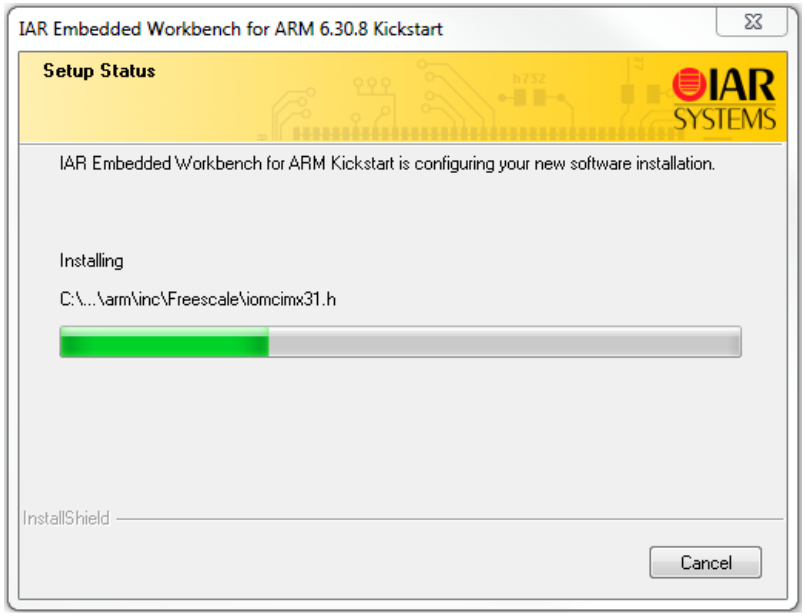
Click next until you are beeing asked to fill in your name, company and license number. To get your license number, go to **hhtp://supp.iar.com/Download/sw/?item=EWARM-KS32**, and fill in the form. When you are done, you will reveice an e-mail with your license number and license key.

Figure 5.4. Type in your name, company and license number.



Click next and fill in your license key. When this is done click next and install the IAR Embedded Workbench. This may take a few minutes.

Figure 5.5. Install the IAR Embedded Workbench.



5.3 Keil µVision

Keil µVision is an commercial IDE developed by ARM. It is also available in a version with 32kb code size limit

5.4 Rowley Crossworks

Rowley Crossworks is a commercial IDE.

5.5 Codesourcery

Codesourcery offers a commercial alternative, to the command line tools presented in the previous chapter called, Sourcery CodeBench.

6 Summary

In this lesson the general steps of a toolchain has been presented. Then followed a guide on how to set up a Linux toolchain, before listing the available Windows development environment.

7 Revision History

7.1 Revision 1.00

2011-06-22

Initial revision.

7.2 Revision 1.10

2012-07-27

Updated for Giant Gecko STK.

A Disclaimer and Trademarks

A.1 Disclaimer

Energy Micro AS 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 Energy Micro 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. Energy Micro 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. Energy Micro 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 Energy Micro. 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. Energy Micro products are generally not intended for military applications. Energy Micro 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

Energy Micro, EFM32, EFR, logo and combinations thereof, and others are the registered trademarks or trademarks of Energy Micro AS. ARM, CORTEX, THUMB are the registered trademarks of ARM Limited. Other terms and product names may be trademarks of others.

B Contact Information

B.1 Energy Micro Corporate Headquarters

Postal Address	Visitor Address	Technical Support
Energy Micro AS P.O. Box 4633 Nydalen N-0405 Oslo NORWAY	Energy Micro AS Sandakerveien 118 N-0484 Oslo NORWAY	support.energymicro.com Phone: +47 40 10 03 01

www.energymicro.com
Phone: +47 23 00 98 00
Fax: + 47 23 00 98 01

B.2 Global Contacts

Visit **www.energymicro.com** for information on global distributors and representatives or contact **sales@energymicro.com** for additional information.

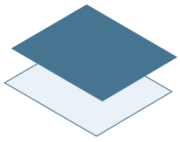
Americas	Europe, Middle East and Africa	Asia and Pacific
www.energymicro.com/americas	www.energymicro.com/emea	www.energymicro.com/asia

Table of Contents

List of Figures

List of Tables

List of Examples



ENERGY®
micro

Energy Micro AS
Sandakerveien 118
P.O. Box 4633 Nydalen
N-0405 Oslo
Norway

www.energymicro.com