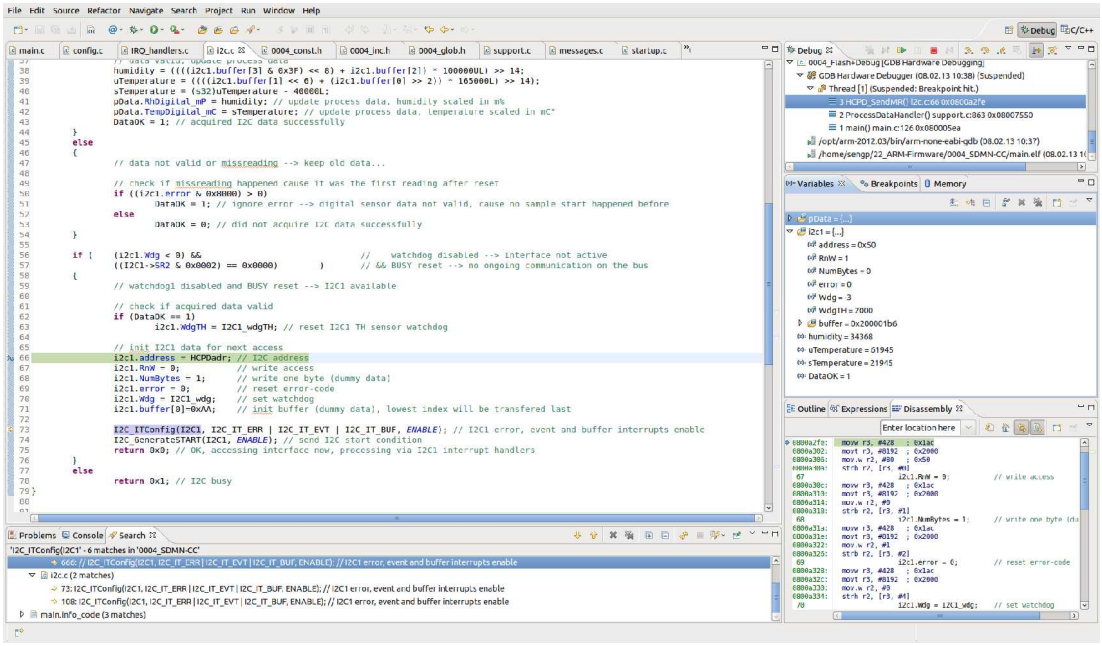
How-to manual

Installing a toolchain for Cortex-M3/STM32 on Ubuntu

Version 0.8.2, 2013-01-18



Disclaimer of Warranty

THERE IS NO WARRANTY FOR THE CONTENT, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE CONTENT “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE CONTENT IS WITH YOU. SHOULD THE CONTENT PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

Limitation of Liability

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE CONTENT AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE CONTENT (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAM), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Contents

1 About

this manual describes how to install a toolchain for Cortex-M3 on Ubuntu GNU/Linux. All packages used, except the GCC toolchain, are open source. For this part a free, unlimited and up to date version of “Sourcery CodeBench”, based on the GNU tools, is used in order to ease the install and build procedure.
Most content of this manual is based on the knowledge and the excellent how-to pages of Johan Simonsson at http://fun-tech.se/stm32/ (1) and Geoffrey McRae at http://stm32.spacevs.com/ (2). Consider this manual as a summary and extension of these guides. If any questions arise, please first have a look at these pages where much more aspects are touched and explained.

For better reading of this document command inputs and outputs via a terminal window are formated like this. The content of source files is enclosed in frames.
Hint: PDF documents do not contain tab formatting marks and empty lines. So it is not possible to copy source code out of a PDF document by copy and paste without loss of this information.

The content of this manual may not be up to date. So before downloading and installing any package, please check if the mentioned packages are still up to date. If newer packages exist and it is sensible to use them please adapt the instructions to these conditions.

Much thanks and lot's of greetings to all those people developing and improving these artful tools running on GNU/Linux.

After nearly one year of coding, using the toolset for hours most days, it has proven to be reliable, comfortable and very satisfying. Any improvements necessary will be documented in future versions of this manual.
Any comments welcome, please mail to: info@seng.de

2 Hardware

Hardware used:

- Olimex “ARM-USB-OCD-H”. USB ARM JTAG device with one additional RS-232 port. The device is based on the FTDI “FT2232H” chip.
- Olimex “STM32-H103”. Header board for “STM32F103RBT6”. The microcontroller integrates 128KB Flash, 20KB RAM, 3xUART, ...

3 Software

The toolchain consists of following packages:

- OpenOCD
- stm32flash by Geoffrey McRae (2)
- “Sourcery CodeBench Lite for ARM EABI”
- STM32F10x standard peripheral library
- Project template and makefile by Geoffrey McRae (2)
- Doxygen
- Git
- Eclipse IDE

4 Basic tools

This chapter is about installing the basic toolchain.

4.1 OpenOCD

Open On-Chip Debugger is the part of software that is needed to enable the JTAG-hardware ("ARM-USB-OCD-H") to flash and debug the microcontroller, it is the software interface to GDB.
OpenOCD downloads and documentation can be found at:
http://openocd.sourceforge.net/about/

4.1.1 Download, build and install

Create a temporary directory:
```
mkdir ~/temp/stm32/
```

Download the documentation:
```
cd ~/temp/stm32/
wget http://openocd.sourceforge.net/doc/pdf/openocd.pdf
```

Think about a **structure to store the documentation** of this toolchain in and store it there.

Install some packages that are needed to build the program:
```
sudo apt-get install libftdi-dev libftdi1 libtool git-core asciidoc \
   build-essential flex bison \
   libgmp3-dev libmpfr-dev autoconf \
   texinfo libncurses5-dev libexpat1 libexpat1-dev \
   tk tk8.4 tk8.4-dev
```

Get and compile the program (these instructions will install version 0.5.0 of the package):
```
mkdir -p ~/temp/stm32/stm32-tools
cd ~/temp/stm32/stm32-tools
git clone git://openocd.git.sourceforge.net/gitroot/openocd/openocd OpenOCD
cd OpenOCD
#git tag
git reset --hard v0.5.0
./bootstrap
./configure --enable-maintainer-mode \
          --enable-ft2232_libftdi
make $PARALLEL
sudo make install
```

Check where the program was installed and which version:
```
which openocd
openocd -v
```
Default install directory for OpenOCD when compiled by yourself is "/usr/local/" so you should see something like this:
`/usr/local/bin/openocd` and some more version info.

4.1.2 Install JTAG device

Connect the “ARM-USB-OCD-H” JTAG device to your computer and check if it is recognized:
```
lsusb
```

You should see something like this:
```
...
Bus 001 Device 010: ID 15ba:002b Olimex Ltd.
...
```
The Olimex device is based on the FT2232H USB-chip from FTDI. This is a Hi-Speed Dual USB UART/FIFO IC, that implements 2 serial/parallel ports in one USB-device. One port is used to implement a JTAG port, the other port is used to implement a RS232 serial port. The chip normally is automatically recognized by the operating system as an FTDI device upon connection with the PC. Olimex re-programs the FTDI USB id's to Olimex values (idVendor=15ba, idProduct=002b) during manufacturing. So the device can not be identified automatically by the operating system any more. To make the JTAG and RS232 serial port of the device usable, a rules file has to be added to the GNU/Linux system. Content of file and futher comments see down.
Name and location of this file should be: /etc/udev/rules.d/OLIMEX_ARM-USB-OCD-H.rules

```
#Scope: making JTAG port and RS232 serial port of Olimex ARM-USB-OCD-H device work on a linux system.
#-----------------------------------------------------------------------------------------------------------
#Name and location of this file should be: /etc/udev/rules.d/99-OLIMEX_ARM-USB-OCD-H.rules
#These commands are a sum of investigation on the web and seem to work properly, but are not well understood by the author.
#Peter Seng, 2011-12-22, 2013-01-18
#For further info please see: http://rowley.zendesk.com/entries/45561-how-to-set-up-linux-for-usb-jtag-adapters
#-----------------------------------------------------------------------------------------------------------
#Olimex ARM-USB-OCD-H device is based on FT2232H. This is a Hi-Speed Dual USB UART/FIFO IC, that implements 2 serial/parallel ports on one USB-device.
#One port is used to implement a JTAG port, the other port is used to implement a RS232 serial port.
#Olimex replaced FTDI id's with it's own Olimex id's (idVendor=15ba, idProduct=002b).
#-----------------------------------------------------------------------------------------------------------
#Hint: if used with Olimex ARM-USB-OCD (without -H) or similar devices adjust Product ID's to appropriate values. Device id's can be found by use of command "lsusb" in a terminal window.
#-----------------------------------------------------------------------------------------------------------
#Following section is valid for Ubuntu 10.04
#Statement SYSFS{idProduct}=="002b", SYSFS{idVendor}=="15ba", MODE="664", GROUP="plugdev" is necessary to notify changed id's, so that JTAG port can work.
#Statement ""MODE="664"" may also be ""MODE="666"" which is used by other implementations found during the investigation.
#Lines "BUS!="usb", ACTION!="add", SUBSYSTEM!=="usb_device", GOTO="kcontrol_rules_end"" and "LABEL="kcontrol_rules_end"" are necessary to notify that the second port should be used as serial device.
#Statement ", RUN+="/sbin/modprobe -q ftdi_sio product=0x002b vendor=0x15ba"" is necessary to notify that the Olimex id's should also used be used by the FTDI driver, so that the serial port can work.
#Uncomment following 3 lines:
#BUS!="usb", ACTION!="add", SUBSYSTEM!=="usb_device", GOTO="kcontrol_rules_end"
#SYSFS{idProduct}=="002b", SYSFS{idVendor}=="15ba", MODE="664", GROUP="plugdev", RUN+="/sbin/modprobe -q ftdi_sio product=0x002b vendor=0x15ba"
#LABEL="kcontrol_rules_end"
#-----------------------------------------------------------------------------------------------------------
#Following section is valid for Ubuntu 12.04 and newer systems
#Uncomment following line:
SUBSYSTEMS=="usb", ATTRS{idVendor}=="15ba", ATTRS{idProduct}=="002b", MODE="0666"
#-----------------------------------------------------------------------------------------------------------
```
OLIMEX_ARM-USB-OCD-H.rules

check with:
```
dmesg | grep usb
```
that it is identified in a correct way, and you don't get any strange errors.

Now you should see something like this:
(Ubuntu 10.04 will not show second line)
```
[17611.036358] usb 1-5.1.3: new high-speed USB device number 44 using ehci_hcd
[17611.137606] usb 1-5.1.3: Ignoring serial port reserved for JTAG
[17611.140887] usb 1-5.1.3: Detected FT2232H
[17611.140889] usb 1-5.1.3: Number of endpoints 2
[17611.140891] usb 1-5.1.3: Endpoint 1 MaxPacketSize 512
[17611.140893] usb 1-5.1.3: Endpoint 2 MaxPacketSize 512
[17611.140895] usb 1-5.1.3: Setting MaxPacketSize 512
[17611.141229] usb 1-5.1.3: FTDI USB Serial Device converter now attached to ttyUSB0
```

List serial ports by use of:
dmesg | grep tty
Something like this should be displayed:
(Ubuntu 10.04 will show 2 additional FTDI devices)
[17611.141229] usb 1-5.1.3: FTDI USB Serial Device converter now attached to ttyUSB0

4.1.3 Configure

OpenOCD uses a configuration file called “openocd.cfg” on startup.
It contains information about:

1) the deamon (ports) configuration
2) the interface (JTAG) configuration
3) the board (microcontroller) configuration
4) the target (microcontroller) configuration

Cause this information may vary between projects, **the configuration file should be present in the directory of every project**.

The daemon section of “openocd.cfg” contains following text:

```
#daemon configuration
telnet_port 4444
gdb_port 3333
```

Interface, board and target sections for the most common devices are part of the OpenOCD package. These are present underneath the following directory:
“/usr/local/share/openocd/scripts/....”

The interface section for ARM-USB-OCD-H is a copy from:
“/usr/local/share/openocd/scripts/interface/olimex-arm-usb-ocd-h.cfg”

The board section for Olimex STM32-H103 is a copy from:
“/usr/local/share/openocd/scripts/board/olimex_stm32_h103.cfg”
Following supplement is necessary to define in which way OpenOCD will access the reset pin of the MCU via the JTAG device:

```
# reset_config parameter (see OpenOCD manual):
#    none              --> srst and trst of MCU not connected to JTAG device
#    srst_only         --> only srst of MCU connected to JTAG device
#    trst_only         --> only trst of MCU connected to JTAG device
#    srst_and_trst     --> srst and trst of MCU connected to JTAG device
# default setting: "reset_config none" will produce a single reset via SYSRESETREQ (JTAG commands) at reset pin of MCU (OpenOCD 0.5.0)
# setting: "reset_config srst_only separate srst_nogate srst_open_drain" will produce a double reset at reset pin of CPU (OpenOCD 0.5.0),
#         first (erroneous) reset via SYSRESETREQ, second via reset output of JTAG device.
reset_config none
```

Hardware access to trst pin (JTAG reset) of the MCU is not enabled, it can be accessed via JTAG commands. If the reset signal (srst) of the MCU is not available at the JTAG connector and/or reset is done via “SYSRESETREQ” parameter must be set to “none”. This setting has vital influence on the debugger configuration.

The target section for STM32F103RBT6 is a copy from:
“/usr/local/share/openocd/scripts/target/stm32f1x.cfg”

The complete content of “openocd.cfg” is build from these 4 sections. This file must be created by use of a text editor. **Store a copy of this file in directory ~/temp** for later use.
Beware to add space characters at end of lines – this will cause OpenOCD to produce strange results.

The merged and completed file should look like this:

```
#daemon configuration###############################################################
telnet_port 4444
gdb_port 3333

#interface configuration############################################################
# Olimex ARM-USB-OCD-H
interface ft2232
ft2232_device_desc "Olimex OpenOCD JTAG ARM-USB-OCD-H"
ft2232_layout olimex-jtag
ft2232_vid_pid 0x15ba 0x002b

#board configuration################################################################
# Work-area size (RAM size) = 20KB for STM32F103RB device
set WORKAREASIZE 0x5000
# reset_config parameter (see OpenOCD manual):
#     none                --> srst and trst of MCU not connected to JTAG device
#     srst_only           --> only srst of MCU connected to JTAG device
#     trst_only           --> only trst of MCU connected to JTAG device
#     srst_and_trst       --> srst and trst of MCU connected to JTAG device
# default setting: "reset_config none" will produce a single reset via SYSRESETREQ (JTAG commands) at reset pin of MCU (OpenOCD 0.5.0)
# setting: "reset_config srst_only separate srst_nogate srst_open_drain" will produce a double reset at reset pin of CPU (OpenOCD 0.5.0),
#          first (erroneous) reset via SYSRESETREQ, second via reset output of JTAG device.
reset_config none

#target configuration###############################################################
# script for stm32

if { [info exists CHIPNAME] } {
   set  _CHIPNAME $CHIPNAME
} else {
   set  _CHIPNAME stm32
}

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

# Work-area is a space in RAM used for flash programming
# By default use 16kB
if { [info exists WORKAREASIZE] } {
   set  _WORKAREASIZE $WORKAREASIZE
} else {
   set  _WORKAREASIZE 0x4000
}

# JTAG speed should be <= F_CPU/6. F_CPU after reset is 8MHz, so use F_JTAG = 1MHz
adapter_khz 1000

adapter_nsrst_delay 100
jtag_ntrst_delay 100

#jtag scan chain
if { [info exists CPUTAPID ] } {
   set _CPUTAPID $CPUTAPID
} else {
  # See STM Document RM0008
  # Section 26.6.3
   set _CPUTAPID 0x3ba00477
}
jtag newtap $_CHIPNAME cpu -irlen 4 -ircapture 0x1 -irmask 0xf -expected-id $_CPUTAPID

if { [info exists BSTAPID ] } {
   # FIXME this never gets used to override defaults...
   set _BSTAPID $BSTAPID
} else {
  # See STM Document RM0008
  # Section 29.6.2
  # Low density devices, Rev A
  set _BSTAPID1 0x06412041
  # Medium density devices, Rev A
  set _BSTAPID2 0x06410041
  # Medium density devices, Rev B and Rev Z
  set _BSTAPID3 0x16410041
  set _BSTAPID4 0x06420041
```

```
  # High density devices, Rev A
  set _BSTAPID5 0x06414041
  # Connectivity line devices, Rev A and Rev Z
  set _BSTAPID6 0x06418041
  # XL line devices, Rev A
  set _BSTAPID7 0x06430041
}
jtag newtap $_CHIPNAME bs -irlen 5 -expected-id $_BSTAPID1 \
               -expected-id $_BSTAPID2 -expected-id $_BSTAPID3 \
               -expected-id $_BSTAPID4 -expected-id $_BSTAPID5 \
               -expected-id $_BSTAPID6 -expected-id $_BSTAPID7

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

$_TARGETNAME configure -work-area-phys 0x20000000 -work-area-size $_WORKAREASIZE -work-area-backup 0

# flash size will be probed
set _FLASHNAME $_CHIPNAME.flash
flash bank $_FLASHNAME stm32f1x 0x08000000 0 0 0 $_TARGETNAME

# if srst is not fitted use SYSRESETREQ to perform a (soft) reset, see “reset_config” parameter above
# this kind of reset will activate reset pin of MCU for > 20µs via JTAG signaling
cortex_m3 reset_config sysresetreq
```

openocd.cfg

4.2 Serial bootloader

STM32 devices can also be programmed by use the integrated bootloader. This might be useful for production, update in the field or low cost development purposes. The bootloader of the "STM32F103RBT6" device (STM32F10xxx family) supports only the USART1 interface.
The hardware that is required to bring the STM32 into System memory boot mode can consists of any circuitry capable of holding the **BOOT0** pin high and the **BOOT1** pin low during reset. To connect the STM32 during "System memory boot mode" to the programming host, **a (LV)TTL-level** RS-232 serial interface directly linked to the **USART1_RX** (PA10) and **USART1_TX** (PA9) pins must be used. Pins USART1_RX and USART1_TX of "STM32F103RBT6" are 5V tolerant, for other devices or pins see datasheet. USART1_CK, USART1_CTS and USART1_RTS pins are not used in this mode, so these pins are available for other peripherals or GPIO's.
For more details about hardware recommendations, refer to application note "AN2586 - STM32 hardware development: getting started". For further details about the bootloader please see application note "AN2606 - STM32TM microcontroller system memory boot mode". Both available at the ST Microelectronics website: http://www.st.com
To upload the program from the PC to the device a serial uploader is necessary. There exist two programs, one for GNU/Linux and one for Windows operating systems.

4.2.1 stm32flash

Program for GNU/Linux, developed by Geoffrey McRae (2). It can be downloaded from:
http://code.google.com/p/stm32flash/
It supports raw binary and Intel HEX files for flashing. To get the source of the program, by the use of subversion, run following commands:

```
cd ~/temp/
svn checkout http://stm32flash.googlecode.com/svn/trunk/ stm32flash
cd ~/temp/stm32flash/
```

Compile the program:

```
make
```

Install the utility into "/usr/local/bin":

```
sudo make install
```

Following example command downloads file "main.bin" via serial port "ttyS0" to the MCU:

```
stm32flash -w main.bin -v /dev/ttyS0
```

Replace "ttyS0" by the port in use ("ttyUSB0" for example). To find out the ports available use:

```
dmesg | grep tty
```

This command will show a list of the ports.

4.2.2 Flash loader demonstrator

In case a PC with GNU/Linux OS is not available, flashing can be done by use of a windows PC.
This program is for MS-Windows, developed by ST Microelectronics. It can be downloaded from:
http://www.st.com/internet/com/SOFTWARE_RESOURCES/SW_COMPONENT/SW_DEMO/um0462.zip
Documentation can be downloaded from:
http://www.st.com/internet/com/TECHNICAL_RESOURCES/TECHNICAL_LITERATURE/USER_MANUAL/CD00171488.pdf
See the manual for information about install and use of the program.

4.3 GCC toolchain

In order to keep installation simple a free, unlimited and up to date version of “Sourcery CodeBench”, based on the GNU tools is used.
Information about “Sourcery CodeBench” can be found at:
http://www.codesourcery.com/sgpp/lite

4.3.1 Download

Be sure to download the embedded-application binary interface (EABI) version, it is built to produce a raw binary that will run stand-alone on the device without an operating system.
In order to make installation easy and clear download the tarball version (IA32 GNU/Linux TAR) and make some manual settings afterwards.

Link to site where the package can be downloaded from after some registration:
http://www.mentor.com/embedded-software/sourcery-tools/sourcery-codebench/editions/lite-edition/
Direct link to download area:
http://www.codesourcery.com/sgpp/lite/arm/portal/subscription?@template=lite

4.3.2 Installation

These instructions will install version “Sourcery CodeBench Lite 2012.03-56 for ARM EABI” .
Included are:

- GNU Binary Utilities (2.21.53-sg++)
- GNU C & C++ Compilers (4.6.3-sg++)
- GNU Debugger (7.2.50-sg++)
- Newlib C Library (1.18.0-sg++)

Extract the tarball (arm-2012.03-56-arm-none-eabi-i686-pc-linux-gnu.tar.bz2) and then copy the extracted contents to “/opt” as root.
Add path “/opt/arm-2012.03/bin” to your environment, therefore append following 2 lines at end of file “.profile” located at your home directory.

```
…
# set PATH to Sourcery CodeBench EABI ARM-Compiler
export PATH=$PATH:/opt/arm-2012.03/bin
```

The arm-none-eabi compiler is now ready to use.

Documentation of the toolchain in PDF format is available on disk in directory:
“/opt/arm-2012.03/share/doc/arm-arm-none-eabi/pdf “
Documentation of the toolchain in HTML format is available on disk in directory:
“/opt/arm-2012.03/share/doc/arm-arm-none-eabi/html”

To verify that PATH is set up correctly enter following command:
`arm-none-eabi-gcc -v`
The last line of the output should contain the actual compiler version.

5 Basic projects

To check whether the compiler works, we should create and compile some small programs/projects. Therefore it it a good idea to create a directory within the home directory to store the coming glamorous projects in:
mkdir -p ~/22_ARM-Firmware

5.1 0001_Test_Blink

This project is just for test purposes. We will show how a program can be compiled, flashed to the device and debugged via GDB. Do not use it as sample or template for future work.
The content of this project will be downloaded from Olimex, some hints are in a file called projects.txt.

First create a directory for this project:
mkdir -p ~/22_ARM-Firmware/0001_Test_Blink

Copy your OpenOCD configuration file openocd.cfg to the project directory
cp ~/temp/openocd.cfg ~/22_ARM-Firmware/0001_Test_Blink

Now download the Olimex files and copy them to the project directory:
mkdir -p ~/temp
cd ~/temp
wget http://olimex.com/dev/soft/arm/STR/STM32-BLINK-LED-GCC-ECLIPSE-projects.rar
unrar x STM32-BLINK-LED-GCC-ECLIPSE-projects.rar
cp ~/temp/projects/projects.txt ~/22_ARM-Firmware/0001_Test_Blink
cp ~/temp/projects/stm_h103/* ~/22_ARM-Firmware/0001_Test_Blink
cd ~/22_ARM-Firmware/0001_Test_Blink

5.1.1 Compile

And now let's make a clean, recompile and see what we get.
make clean
Now you will see some messy output – this project should not be used as a template.
Recompile program:
make
Again you will see some messy output and the version of the used compiler too.
The program should have compiled without generating error messages.
Let's check what happened:
ls --sort=time -1 -l
The directory is displayed, newest files first:
main.list
main.bin
main.out
stm32f10x_gpio.o
stm32f10x_rcc.o
main.o
arm_comm.h
…

The binary code to be flashed on the MCU is “main.bin”.
The file with the strange name “main.out” is the ELF-file that normally should be named “main.elf”.
The ELF-file contains debug information and is used for debug and other purposes.

For a closer look at the ELF-file use command “readelf” from “binutils”:
```
arm-none-eabi-readelf -A main.out
```
Now you should see something like this:
```
Attribute Section: aeabi
File Attributes
  Tag_CPU_name: "Cortex-M3"
  Tag_CPU_arch: v7
  Tag_CPU_arch_profile: Microcontroller
  Tag_THUMB_ISA_use: Thumb-2
  Tag_ABI_PCS_wchar_t: 4
  Tag_ABI_FP_denormal: Needed
  Tag_ABI_FP_exceptions: Needed
  Tag_ABI_FP_number_model: IEEE 754
  Tag_ABI_align_needed: 8-byte
  Tag_ABI_align_preserved: 8-byte, except leaf SP
  Tag_ABI_enum_size: small
  Tag_ABI_optimization_goals: Aggressive Debug
  Tag_CPU_unaligned_access: v6
```
It is shown that we produced code for “Cortex-M3” and lots of other information too.

Use objdump to look at the ELF-file with the -S flag:
```
arm-none-eabi-objdump -S main.out
```

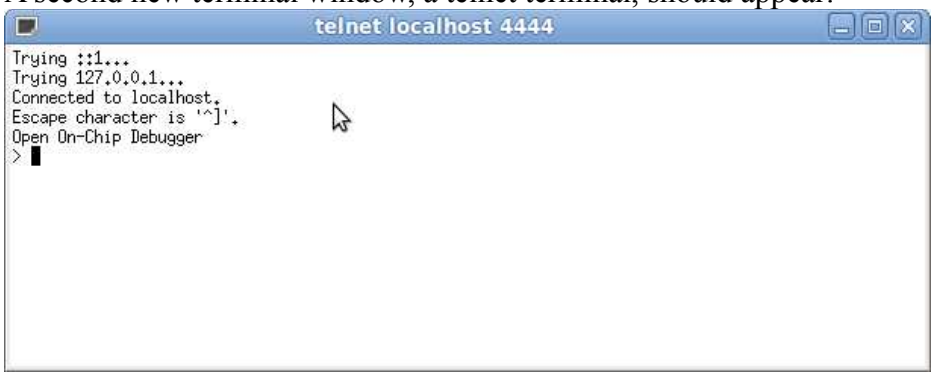
The makefile also created a file called “main.list”. Included is assembler and interleaved C-code interleaved. Open file by use of an editor:
```
gedit main.list
```
This is the result of the compiling. If you want to have a closer look at the previous steps, see (1). There you will also find some explanatory notes about the program and the associated makefile.

5.1.2 Flash and run

Now we will flash the MCU and check whether the green LED will start blinking.

Start the OpenOCD server:
`xterm -geometry 100x16+100+100 -e "openocd -f openocd.cfg" &`
A new terminal window should appear:

Then start a telnet session, so that you can talk to the OpenOCD server:
`xterm -geometry 100x16+100+350 -e "telnet localhost 4444" &`
A second new terminal window, a telnet terminal, should appear:



At the prompt of the telnet terminal enter following commands:
`reset halt`
`flash probe 0`
`stm32f1x mass_erase 0`
`flash write_bank 0 main.bin 0`
`reset run`

Every command will produce some output in the 2 terminal windows. If the green led is blinking after the last command, compiling and download worked properly.
For a description of the commands used, see the OpenOCD manual.

5.1.3 Debug

Now let's check if the program can be debugged by use of GDB. Resize your terminal window to a very big one.

At your telnet terminal enter:
`reset halt`
The device will stop blinking.

At you terminal window enter:
`arm-none-eabi-gdbtui --eval-command="target remote localhost:3333" main.out`

Follow the instructions shown in the terminal window (You may have to press return).
Inputs to GDB have to be confirmed by pressing the enter key.
Step through the program by typing (commands always followed by enter):

```
s
s
…
```

Set 2 breakpoints at main:

```
break main.c:60
break main.c:62
```

Now lets run the program from breakpoint to breakpoint by entering:

```
c
c
…
```

This way the green LED can be switched on and off...
If this works, debugging with GDB also works properly. There exist a lot of useful graphical frontends to GDB like KDbg or DDD (Data Display Debugger). If you want to know more about these frontends, see (1). Later on we will install an IDE (Integrated Desktop Environment) which also includes a graphical frontend to GDB, so for our purposes a stand alone graphical frontend is not needed.

5.1.4 Make & Flash

This section shows how to automate the flash procedure by using a Perl telnet script and how to integrated it into the make process.
Install Perl telnet:

```
sudo apt-get install libnet-telnet-perl
```

Create a file in the in the project directory called “do_flash.pl” and paste the following content into it:

```
#!/usr/bin/perl
use Net::Telnet;

$numArgs = $#ARGV + 1;
if($numArgs != 1){
    die( "Usage ./do_flash.pl [main.bin] \n");
}

$file   = $ARGV[0];

$ip = "127.0.0.1";
$port = 4444;

$telnet = new Net::Telnet (
    Port   => $port,
    Timeout=>10,
    Errmode=>'die',
    Prompt =>'/>/');

$telnet->open($ip);

print $telnet->cmd('reset halt');
print $telnet->cmd('flash probe 0');
print $telnet->cmd('stm32f1x mass_erase 0');
print $telnet->cmd('flash write_bank 0 '.$file.' 0');
print $telnet->cmd('reset halt');
print $telnet->cmd('exit');

print "\n";
```

do_flash.pl

Now lets modify file “makefile” by adding following text:

```
flash: all
	./do_flash.pl main.bin
```

File “makefile” now looks like this:

```
NAME   = demoh103_blink_rom

CC      = arm-none-eabi-gcc
LD      = arm-none-eabi-ld -v
AR      = arm-none-eabi-ar
AS      = arm-none-eabi-as
CP      = arm-none-eabi-objcopy
OD                      = arm-none-eabi-objdump

CFLAGS  =  -I./ -c -fno-common -O0 -g -mcpu=cortex-m3 -mthumb
AFLAGS  = -ahls -mapcs-32 -o crt.o
LFLAGS  = -Tstm_h103_blink_rom.cmd -nostartfiles
CPFLAGS = -Obinary
ODFLAGS             = -S

all: test

clean:
	-rm crt.lst a.lst main.lst crt.o main.o main.out main.hex main.map stm32f10x_rcc.o stm32f10x_gpio.o

test: main.out
	@ echo "...copying"
	$(CP) $(CPFLAGS) main.out main.bin
	$(OD) $(ODFLAGS) main.out > main.list

main.out: main.o stm32f10x_rcc.o stm32f10x_gpio.o stm_h103_blink_rom.cmd
	@ echo "..linking"
	$(LD) $(LFLAGS) -o main.out  main.o stm32f10x_rcc.o stm32f10x_gpio.o

crt.o: crt.s
	@ echo ".assembling"
	$(AS) $(AFLAGS) crt.s > crt.lst

stm32f10x_rcc.o: stm32f10x_rcc.c
	@ echo ".compil"
	 $(CC) $(CFLAGS) stm32f10x_rcc.c

stm32f10x_gpio.o: stm32f10x_gpio.c
	@ echo ".compili"
	 $(CC) $(CFLAGS) stm32f10x_gpio.c

main.o: main.c
	@ echo ".compiling"
	 $(CC) $(CFLAGS) main.c

flash: all
	./do_flash.pl main.bin
```

makefile

Now we have an additional make command available. Close the telnet terminal window. At the terminal window enter:
make flash
Have a look at the OpenOCD terminal window and the terminal window and see that build and flash have been executed.

To clean up you project enter following command:
make clean

5.2 0002_Test_Template

This project is for test purposes too. It shows how a project can be structured, which libraries should be included and provides some makefiles to build the project. This project can be used as a **template** for future projects.

First create a directory for this project:
mkdir -p ~/22_ARM-Firmware/0002_Test_Template

Copy your OpenOCD configuration file openocd.cfg to the project directory
cp ~/temp/openocd.cfg ~/22_ARM-Firmware/0002_Test_Template

5.2.1 Librarys

The “STM32F10x standard peripheral library” contains device drivers for all standard device peripherals, including functions covering full peripheral functionality. The C-source is documented and tested. It contains all defines and structures needed for coding with the STM32. The library is provided by ST Microelectronics:
http://www.st.com/internet/com/SOFTWARE_RESOURCES/SW_COMPONENT/FIRMWARE/stm32f10x_stdperiph_lib.zip
This file contains version 3.5.0 of the library including example code. Documentation that comes with the library is in CHM format, so a reader like xCHM or ChmSee has to be installed to read the documentation.

The “USB full-speed device library” enables building applications including USB functionality. The library is provided by ST Microelectronics:
http://www.st.com/internet/com/SOFTWARE_RESOURCES/SW_COMPONENT/FIRMWARE/stm32_usb-fs-device_lib.zip
This file contains the USB library and the “STM32F10x standard peripheral library” version 3.6.1. without example code.

5.2.1.1 Install StdPeriph_Lib_V3.5.0

Make a new directory in the project called "libs":
mkdir -p ~/22_ARM-Firmware/0002_Test_Template/libs
This is the location to store all third party libraries and headers required for the project. Extract the STM32 library to this directory.
The package is big (> 30 MB), so it may be a good idea to store some content of this directory (like the CHM documentation and examples) not in the project directory but elsewhere. Otherwise duplication of data will waste a lot of disk space every time the template is copied when starting with a new project.
Now following path should exist:
"~/22_ARM-Firmware/0002_Test_Template/libs/STM32F10x_StdPeriph_Lib_V3.5.0/Libraries".

5.2.1.2 Install USB library and StdPeriph_Lib_V3.6.1

Decompress the “USB full-speed device library” and replace the V3.5.0 content in the “Libraries” directory of the “STM32F10x standard peripheral library” with the content included in the “USB full-speed device library”. Rename path to:
"~/22_ARM-Firmware/0002_Test_Template/libs/STM32F10x_StdPeriph_Lib_V3.6.1/...".

If USB functionality of interest copy the content of the “Project” directory (USB examples) to a separate directory.
Later on we will use the CMSIS (Cortex Microcontroller Software Interface Standard) headers and helper functions from these libs.

Note: CMSIS structure is not compatible to structure contained in older librarys. Makefiles described later on will base on StdPeriph_Lib_V3.6.1. If you use a different version of the library you will need to update the path and name settings in the makefile “Makefile.common” to reflect the change.

5.2.1.3 Content

The package is worth to be examined in detail, there is a lot of example code and information inside. For building working projects not using USB functionality only the following paths are needed:

- Libraries/CMSIS
- Libraries/STM32F10x_StdPeriph_Driver/inc
- Libraries/STM32F10x_StdPeriph_Driver/src

The CMSIS directory contains the defines and data structures for every peripheral in the STM32, as well as the defines for the configuration registers and values.
The other paths contain the helper functions that try to make programming the STM32 simple. They add a layer of abstraction – feel free to use them or not.

5.2.2 Basic Makefiles

The build process, compiling and linking source code, can be simplified and automated by the use of Makefiles. The use of Makefiles may look complicated, using make has many advantages:

- Results become predictable and reproducible.
- Makefiles can (should) contain hints and comments.
- Makefiles are stored within the project.
- The use of make offers much more possibilities and a better clarity than configuring a build via a predefined input menu structure inside a compiler specific IDE (everybody will agree who was already fishing for “this special compiler switch” used some years or projects ago).

For a full description please see the “GNU Make manual” at http://www.gnu.org/software/make/manual/make.pdf or have a web search.

5.2.2.1 Common Makefile

This makefile has to be included into all other makefiles. It contains variable setup for the build procedure. Create a file at the top level of the project called “Makefile.common” and paste the following text into it:

```
TOP=$(shell readlink -f "$(dir $(lastword $(MAKEFILE_LIST)))")
PROGRAM=main
LIBDIR=$(TOP)/libs
#Adust the following line to the library in use
STMLIB=$(LIBDIR)/STM32F10x_StdPeriph_Lib_V3.6.1/Libraries

TC=arm-none-eabi
CC=$(TC)-gcc
LD=$(TC)-ld -v
OBJCOPY=$(TC)-objcopy
AR=$(TC)-ar
GDB=$(TC)-gdb

INCLUDE=-I$(TOP)/inc
INCLUDE+=-I$(STMLIB)/CMSIS/Include
INCLUDE+=-I$(STMLIB)/CMSIS/Device/ST/STM32F10x/Include
INCLUDE+=-I$(STMLIB)/CMSIS/Device/ST/STM32F10x/Source/Templates
INCLUDE+=-I$(STMLIB)/STM32F10x_StdPeriph_Driver/inc
INCLUDE+=-I$(STMLIB)/STM32_USB-FS-Device_Driver/inc

#Compiler optimize settings:
# -O0 no optimize, reduce compilation time and make debugging produce the expected results (default).
# -O1 optimize, reduce code size and execution time, without much increase of compilation time.
# -O2 optimize, reduce code execution time compared to ‘O1’, increase of compilation time.
# -O3 optimize, turns on all optimizations, further increase of compilation time.
# -Os optimize for size, enables all ‘-O2’ optimizations that do not typically increase code size and other code size optimizations.
# default settings for release version: COMMONFLAGS=-O3 -g -mcpu=cortex-m3 -mthumb
# default settings for debug version: COMMONFLAGS=-O0 -g -mcpu=cortex-m3 -mthumb
#COMMONFLAGS=-O3 -g -mcpu=cortex-m3 -mthumb
COMMONFLAGS=-O0 -g -mcpu=cortex-m3 -mthumb
CFLAGS+=$(COMMONFLAGS) -Wall -Werror $(INCLUDE)
#Adust the following line to the type of MCU used
CFLAGS+=-D STM32F10X_MD
CFLAGS+=-D VECT_TAB_FLASH
```

Makefile.common

The makefile will use library “STM32F10x_StdPeriph_Lib_V3.6.1”. Code optimization is turned off to make debugging produce the expected results. File is configured for a “STM32 Medium density device” cause the MCU “STM32F103RBT6” belongs to this device class, change "STM32F10X_MD" to another setting if another microcontroller belonging to a different class is used. An explanation of this define can be found in file:
/libs/STM32F10x_StdPeriph_Lib_V3.6.1/Libraries/CMSIS/Device/ST/STM32F10x/Include/stm32f10x.h
inside your current project directory.

5.2.2.2 Libs Makefile

When building the STM32 library as a static library, changes to the application do not induce a complete re-compile of the library and this speeds up the build process. Create another Makefile named “Makefile” in the libs directory with the following contents by use of a text editor:

```
include ../Makefile.common
LIBS+=libstm32.a
CFLAGS+=-c

all: libs

libs: $(LIBS)

libstm32.a:
	@echo -n "Building $@ ..."
	@cd $(STMLIB)/CMSIS/Device/ST/STM32F10x/Source/Templates && \
		$(CC) $(CFLAGS) \
			system_stm32f10x.c
	@cd $(STMLIB)/STM32F10x_StdPeriph_Driver/src && \
		$(CC) $(CFLAGS) \
			-D"assert_param(expr)=((void)0)" \
			-I../../CMSIS/Include \
			-I../../CMSIS/Device/ST/STM32F10x/Include \
			-I../inc \
			*.c
	@cd $(STMLIB)/STM32_USB-FS-Device_Driver/src && \
		$(CC) $(CFLAGS) \
			-D"assert_param(expr)=((void)0)" \
			-I../../CMSIS/Include \
			-I../../CMSIS/Device/ST/STM32F10x/Include \
			-I../inc \
			*.c
	@$(AR) cr $(LIBDIR)/$@ \
		$(STMLIB)/CMSIS/Device/ST/STM32F10x/Source/Templates/system_stm32f10x.o \
		$(STMLIB)/STM32F10x_StdPeriph_Driver/src/*.o \
		$(STMLIB)/STM32_USB-FS-Device_Driver/src/*.o
	@echo "done."

.PHONY: libs clean

clean:
	rm -f $(STMLIB)/CMSIS/Device/ST/STM32F10x/Source/Templates/system_stm32f10x.o
	rm -f $(STMLIB)/STM32F10x_StdPeriph_Driver/src/*.o
	rm -f $(STMLIB)/STM32_USB-FS-Device_Driver/src/*.o
	rm -f $(LIBS)
```

Makefile

To test that everything is OK, execute following commands from the “libs” directory:
make clean
make
Now you should see the STM32 library get compiled, and a new file called “libstm32.a” appear in the current projects “libs” directory. If not, be sure that your cross compiler is installed properly.

5.2.3 Linker Script

The project build will run stand alone without an operating system, so hardware and memory has to be initialized manually. The binary that will be created has to run without a loader such as ELF. For successful execution, the program entry point has to be at a certain address. When dealing with embedded devices a linker script, which tells GCC exactly how to build the binary, is necessary. Create a file called “linker.ld” and paste the following text into it:

```
ENTRY(Reset_Handler)

MEMORY {
	/*Adust in following line variable LENGTH to RAM size of target MCU*/
	RAM     (RWX) : ORIGIN = 0x20000000     , LENGTH = 20K
	EXTSRAM  (RWX) : ORIGIN = 0x68000000     , LENGTH = 0
	/*Adust in following line variable LENGTH to (FLASH size - 2K) of target MCU*/
	FLASH    (RX)  : ORIGIN = 0x08000000     , LENGTH = 126K
	/*Adust in following line variable ORIGIN to (0x08000000 + FLASH size - 2K) of target MCU*/
	EEMUL    (RWX) : ORIGIN = 0x08000000+126K, LENGTH = 2K
}

_estack      = ORIGIN(RAM)+LENGTH(RAM);     /* end of the stack */
_seemul     = ORIGIN(EEMUL);                  /* start of the eeprom emulation area */
_min_stack      = 0x100;                      /* minimum stack space to reserve for the user app */

/* check valid alignment for the vector table */
ASSERT(ORIGIN(FLASH) == ALIGN(ORIGIN(FLASH), 0x80), "Start of memory region flash not aligned for startup vector table");

SECTIONS {
	/* vector table and program code goes into FLASH */
	.text : {
		. = ALIGN(0x80);
		_isr_vectors_offs = . - 0x08000000;
		KEEP(*(.isr_vectors))
		. = ALIGN(4);
		CREATE_OBJECT_SYMBOLS
		*(.text .text.*)
	} >FLASH

	.rodata : ALIGN (4) {
		*(.rodata .rodata.*)

		. = ALIGN(4);
		KEEP(*(.init))

		. = ALIGN(4);
		__preinit_array_start = .;
		KEEP (*(.preinit_array))
		__preinit_array_end = .;

		. = ALIGN(4);
		__init_array_start = .;
		KEEP (*(SORT(.init_array.*)))
		KEEP (*(.init_array))
		__init_array_end = .;

		. = ALIGN(4);
		KEEP(*(.fini))

		. = ALIGN(4);
		__fini_array_start = .;
		KEEP (*(.fini_array))
		KEEP (*(SORT(.fini_array.*)))
		__fini_array_end = .;

		*(.init .init.*)
		*(.fini .fini.*)

		PROVIDE_HIDDEN (__preinit_array_start = .);
		KEEP (*(.preinit_array))
		PROVIDE_HIDDEN (__preinit_array_end = .);
		PROVIDE_HIDDEN (__init_array_start = .);
		KEEP (*(SORT(.init_array.*)))
		KEEP (*(.init_array))
		PROVIDE_HIDDEN (__init_array_end = .);
```

```
		PROVIDE_HIDDEN (__fini_array_start = .);
		KEEP (*(.fini_array))
		KEEP (*(SORT(.fini_array.*)))
		PROVIDE_HIDDEN (__fini_array_end = .);

		. = ALIGN (8);
		*(.rom)
		*(.rom.b)
		_etext = .;
		_sidata = _etext; /* exported for the startup function */
	} >FLASH

	/*
		this data is expected by the program to be in ram
		but we have to store it in the FLASH otherwise it
		will get lost between resets, so the startup code
		has to copy it into RAM before the program starts
	*/
	.data : ALIGN (8) {
		_sdata = . ; /* exported for the startup function */
		. = ALIGN(4);
		KEEP(*(.jcr))
		*(.got.plt) *(.got)
		*(.shdata)
		*(.data .data.*)
		. = ALIGN (8);
		*(.ram)
		*(.ramfunc*)
		 . = ALIGN(4);
		_edata = . ; /* exported for the startup function */
	} >RAM AT>FLASH

	/* This is the uninitialized data section */
	.bss (NOLOAD): {
		. = ALIGN(4);
		_sbss = . ; /* exported for the startup function */
		*(.shbss)
		*(.bss .bss.*)
		*(COMMON)
		. = ALIGN (8);
		*(.ram.b)
		. = ALIGN(4);
		 _ebss = . ; /* exported for the startup function */
		_end = .;
		__end = .;
	} >RAM AT>FLASH
	/* ensure there is enough room for the user stack */
	._usrstack (NOLOAD): {
		. = ALIGN(4);
		_susrstack = . ;
		. = . + _min_stack ;
		. = ALIGN(4);
		_eusrstack = . ;
	} >RAM

	/* Stabs debugging sections.  */
	.stab           0 : { *(.stab) }
	.stabstr        0 : { *(.stabstr) }
	.stab.excl      0 : { *(.stab.excl) }
	.stab.exclstr   0 : { *(.stab.exclstr) }
	.stab.index     0 : { *(.stab.index) }
	.stab.indexstr 0 : { *(.stab.indexstr) }
	.comment        0 : { *(.comment) }
	/* DWARF debug sections.
		Symbols in the DWARF debugging sections are relative to the beginning
		of the section so we begin them at 0.  */
	/* DWARF 1 */
	.debug          0 : { *(.debug) }
	.line           0 : { *(.line) }
	/* GNU DWARF 1 extensions */
	.debug_srcinfo  0 : { *(.debug_srcinfo) }
	.debug_sfnames  0 : { *(.debug_sfnames) }
	/* DWARF 1.1 and DWARF 2 */
	.debug_aranges  0 : { *(.debug_aranges) }
	.debug_pubnames 0 : { *(.debug_pubnames) }
	/* DWARF 2 */
	.debug_info     0 : { *(.debug_info .gnu.linkonce.wi.*) }
	.debug_abbrev   0 : { *(.debug_abbrev) }
	.debug_line     0 : { *(.debug_line) }
```

```
        .debug_frame    0 : { *(.debug_frame) }
        .debug_str      0 : { *(.debug_str) }
        .debug_loc      0 : { *(.debug_loc) }
        .debug_macinfo  0 : { *(.debug_macinfo) }
        /* SGI/MIPS DWARF 2 extensions */
        .debug_weaknames 0 : { *(.debug_weaknames) }
        .debug_funcnames 0 : { *(.debug_funcnames) }
        .debug_typenames 0 : { *(.debug_typenames) }
        .debug_varnames  0 : { *(.debug_varnames) }
        /* DWARF 3 */
        .debug_pubtypes 0 : { *(.debug_pubtypes) }
        .debug_ranges   0 : { *(.debug_ranges) }

        .ARM.attributes 0 : { KEEP (*(.ARM.attributes)) KEEP (*(.gnu.attributes)) }
        .note.gnu.arm.ident 0 : { KEEP (*(.note.gnu.arm.ident)) }
        /DISCARD/ : { *(.note.GNU-stack) *(.gnu_debuglink) }
}
```

linker.ld

This linker script is for a Medium density (MD) device with 128K of flash and 20K of RAM. IF another device is used adjust lines at the top beginning with “RAM”, “FLASH” and “EEMUL”.

2K flash memory is reserved for EEPROM emulation (2 pages x 1K → usable size < 1K), see “AN2594 -EEPROM emulation” for available memory size and access mechanism.

5.2.4 Startup Code

Following code is responsible for device initialization. Static values are copied into RAM. Memory, interrupt vectors and device is initialized. The reset handler is initialized and function main(), starting point for our future program, is called.
Create two directories, one named “scr” and one named “inc” at the top level of the project.
Create a file called “startup.c” in the “src” directory and paste the following text into it:

```
#include "stm32f10x.h"

typedef void( *const intfunc )( void );

#define WEAK __attribute__ ((weak))

/* provided by the linker script */
//extern unsigned long _etext; /* start address of the static initialization data */
extern unsigned long _sidata; /* start address of the static initialization data */
extern unsigned long _sdata; /* start address of the data section */
extern unsigned long _edata; /* end address of the data section */
extern unsigned long _sbss; /* start address of the bss section */
extern unsigned long _ebss; /* end address of the bss section */
extern unsigned long _estack;

void Reset_Handler(void) __attribute__((__interrupt__));
void __Init_Data(void);
void Default_Handler(void);

extern int main(void);

void WEAK NMI_Handler(void);
void WEAK HardFault_Handler(void);
void WEAK MemManage_Handler(void);
void WEAK BusFault_Handler(void);
void WEAK UsageFault_Handler(void);
void WEAK MemManage_Handler(void);
void WEAK SVC_Handler(void);
void WEAK DebugMon_Handler(void);
void WEAK PendSV_Handler(void);
void WEAK SysTick_Handler(void);

void WEAK WWDG_IRQHandler(void);
void WEAK PVD_IRQHandler(void);
void WEAK TAMPER_IRQHandler(void);
void WEAK RTC_IRQHandler(void);
void WEAK FLASH_IRQHandler(void);
void WEAK RCC_IRQHandler(void);
void WEAK EXTI0_IRQHandler(void);
void WEAK EXTI1_IRQHandler(void);
void WEAK EXTI2_IRQHandler(void);
void WEAK EXTI3_IRQHandler(void);
void WEAK EXTI4_IRQHandler(void);
void WEAK DMA1_Channel1_IRQHandler(void);
void WEAK DMA1_Channel2_IRQHandler(void);
void WEAK DMA1_Channel3_IRQHandler(void);
void WEAK DMA1_Channel4_IRQHandler(void);
void WEAK DMA1_Channel5_IRQHandler(void);
void WEAK DMA1_Channel6_IRQHandler(void);
void WEAK DMA1_Channel7_IRQHandler(void);
void WEAK ADC1_2_IRQHandler(void);
void WEAK USB_HP_CAN1_TX_IRQHandler(void);
void WEAK USB_LP_CAN1_RX0_IRQHandler(void);
void WEAK CAN1_RX1_IRQHandler(void);
void WEAK CAN1_SCE_IRQHandler(void);
void WEAK EXTI9_5_IRQHandler(void);
void WEAK TIM1_BRK_IRQHandler(void);
void WEAK TIM1_UP_IRQHandler(void);
void WEAK TIM1_TRG_COM_IRQHandler(void);
void WEAK TIM1_CC_IRQHandler(void);
void WEAK TIM2_IRQHandler(void);
void WEAK TIM3_IRQHandler(void);
void WEAK TIM4_IRQHandler(void);
void WEAK I2C1_EV_IRQHandler(void);
void WEAK I2C1_ER_IRQHandler(void);
void WEAK I2C2_EV_IRQHandler(void);
void WEAK I2C2_ER_IRQHandler(void);
```

```
void WEAK SPI1_IRQHandler(void);
void WEAK SPI2_IRQHandler(void);
void WEAK USART1_IRQHandler(void);
void WEAK USART2_IRQHandler(void);
void WEAK USART3_IRQHandler(void);
void WEAK EXTI15_10_IRQHandler(void);
void WEAK RTCAlarm_IRQHandler(void);
void WEAK USBWakeUp_IRQHandler(void);
void WEAK TIM8_BRK_IRQHandler(void);
void WEAK TIM8_UP_IRQHandler(void);
void WEAK TIM8_TRG_COM_IRQHandler(void);
void WEAK TIM8_CC_IRQHandler(void);
void WEAK ADC3_IRQHandler(void);
void WEAK FSMC_IRQHandler(void);
void WEAK SDIO_IRQHandler(void);
void WEAK TIM5_IRQHandler(void);
void WEAK SPI3_IRQHandler(void);
void WEAK UART4_IRQHandler(void);
void WEAK UART5_IRQHandler(void);
void WEAK TIM6_IRQHandler(void);
void WEAK TIM7_IRQHandler(void);
void WEAK DMA2_Channel1_IRQHandler(void);
void WEAK DMA2_Channel2_IRQHandler(void);
void WEAK DMA2_Channel3_IRQHandler(void);
void WEAK DMA2_Channel4_5_IRQHandler(void);

__attribute__ ((section(".isr_vectors")))
void (* const g_pfnVectors[])(void) = {
    (intfunc)((unsigned long)&_estack), /* The stack pointer after relocation */
    Reset_Handler,              /* Reset Handler */
    NMI_Handler,                /* NMI Handler */
    HardFault_Handler,          /* Hard Fault Handler */
    MemManage_Handler,          /* MPU Fault Handler */
    BusFault_Handler,           /* Bus Fault Handler */
    UsageFault_Handler,         /* Usage Fault Handler */
    0,                          /* Reserved */
    0,                          /* Reserved */
    0,                          /* Reserved */
    0,                          /* Reserved */
    SVC_Handler,                /* SVCall Handler */
    DebugMon_Handler,           /* Debug Monitor Handler */
    0,                          /* Reserved */
    PendSV_Handler,             /* PendSV Handler */
    SysTick_Handler,            /* SysTick Handler */

    /* External Interrupts */
    WWDG_IRQHandler,            /* Window Watchdog */
    PVD_IRQHandler,             /* PVD through EXTI Line detect */
    TAMPER_IRQHandler,          /* Tamper */
    RTC_IRQHandler,             /* RTC */
    FLASH_IRQHandler,           /* Flash */
    RCC_IRQHandler,             /* RCC */
    EXTI0_IRQHandler,           /* EXTI Line 0 */
    EXTI1_IRQHandler,           /* EXTI Line 1 */
    EXTI2_IRQHandler,           /* EXTI Line 2 */
    EXTI3_IRQHandler,           /* EXTI Line 3 */
    EXTI4_IRQHandler,           /* EXTI Line 4 */
    DMA1_Channel1_IRQHandler,   /* DMA1 Channel 1 */
    DMA1_Channel2_IRQHandler,   /* DMA1 Channel 2 */
    DMA1_Channel3_IRQHandler,   /* DMA1 Channel 3 */
    DMA1_Channel4_IRQHandler,   /* DMA1 Channel 4 */
    DMA1_Channel5_IRQHandler,   /* DMA1 Channel 5 */
    DMA1_Channel6_IRQHandler,   /* DMA1 Channel 6 */
    DMA1_Channel7_IRQHandler,   /* DMA1 Channel 7 */
    ADC1_2_IRQHandler,          /* ADC1 & ADC2 */
    USB_HP_CAN1_TX_IRQHandler,  /* USB High Priority or CAN1 TX */
    USB_LP_CAN1_RX0_IRQHandler, /* USB Low  Priority or CAN1 RX0 */
    CAN1_RX1_IRQHandler,        /* CAN1 RX1 */
    CAN1_SCE_IRQHandler,        /* CAN1 SCE */
    EXTI9_5_IRQHandler,         /* EXTI Line 9..5 */
    TIM1_BRK_IRQHandler,        /* TIM1 Break */
    TIM1_UP_IRQHandler,         /* TIM1 Update */
    TIM1_TRG_COM_IRQHandler,    /* TIM1 Trigger and Commutation */
    TIM1_CC_IRQHandler,         /* TIM1 Capture Compare */
    TIM2_IRQHandler,            /* TIM2 */
    TIM3_IRQHandler,            /* TIM3 */
    TIM4_IRQHandler,            /* TIM4 */
    I2C1_EV_IRQHandler,         /* I2C1 Event */
    I2C1_ER_IRQHandler,         /* I2C1 Error */
```

```
    I2C2_EV_IRQHandler,         /* I2C2 Event */
    I2C2_ER_IRQHandler,         /* I2C2 Error */
    SPI1_IRQHandler,            /* SPI1 */
    SPI2_IRQHandler,            /* SPI2 */
    USART1_IRQHandler,          /* USART1 */
    USART2_IRQHandler,          /* USART2 */
    USART3_IRQHandler,          /* USART3 */
    EXTI15_10_IRQHandler,       /* EXTI Line 15..10 */
    RTCAlarm_IRQHandler,        /* RTC Alarm through EXTI Line */
    USBWakeUp_IRQHandler,       /* USB Wakeup from suspend */
    TIM8_BRK_IRQHandler,
    TIM8_UP_IRQHandler,
    TIM8_TRG_COM_IRQHandler,
    TIM8_CC_IRQHandler,
    ADC3_IRQHandler,
    FSMC_IRQHandler,
    SDIO_IRQHandler,
    TIM5_IRQHandler,
    SPI3_IRQHandler,
    UART4_IRQHandler,
    UART5_IRQHandler,
    TIM6_IRQHandler,
    TIM7_IRQHandler,
    DMA2_Channel1_IRQHandler,
    DMA2_Channel2_IRQHandler,
    DMA2_Channel3_IRQHandler,
    DMA2_Channel4_5_IRQHandler,
    0, 0, 0, 0, 0,
    0, 0, 0, 0, 0,
    0, 0, 0, 0, 0,
    0, 0, 0, 0, 0,
    0, 0, 0, 0, 0,
    0, 0, 0, 0, 0,
    0, 0, 0, 0, 0,
    0, 0, 0, 0, 0,
    0, 0, 0,
    (intfunc)0xF1E0F85F         /* @0x1E0. This is for boot in RAM mode for STM32F10x High Density devices. */
};

void __Init_Data(void) {
        unsigned long *src, *dst;
        /* copy the data segment into ram */
        src = &_sidata;
        dst = &_sdata;
        if (src != dst)
                while(dst < &_edata)
                        *(dst++) = *(src++);

        /* zero the bss segment */
        dst = &_sbss;
        while(dst < &_ebss)
                *(dst++) = 0;
}

void Reset_Handler(void) {
        __Init_Data(); /* Initialize memory, data and bss */
        extern u32 _isr_vectors_offs; /* the offset to the vector table in ram */
        SCB->VTOR = 0x08000000 | ((u32)&_isr_vectors_offs & (u32)0x1FFFFF80); /* set interrupt vector table address */
        SystemInit(); /* configure the clock */
        main(); /* start execution of the program */
        while(1) {}
}

#pragma weak MMI_Handler                = Default_Handler
#pragma weak MemManage_Handler                  = Default_Handler
#pragma weak BusFault_Handler                   = Default_Handler
#pragma weak UsageFault_Handler                 = Default_Handler
#pragma weak SVC_Handler                = Default_Handler
#pragma weak DebugMon_Handler                   = Default_Handler
#pragma weak PendSV_Handler             = Default_Handler
#pragma weak SysTick_Handler            = Default_Handler
#pragma weak WWDG_IRQHandler                    = Default_Handler
#pragma weak PVD_IRQHandler                     = Default_Handler
#pragma weak TAMPER_IRQHandler                  = Default_Handler
#pragma weak RTC_IRQHandler                     = Default_Handler
#pragma weak FLASH_IRQHandler                   = Default_Handler
#pragma weak RCC_IRQHandler                     = Default_Handler
#pragma weak EXTI0_IRQHandler                   = Default_Handler
#pragma weak EXTI1_IRQHandler                   = Default_Handler
```

```
#pragma weak EXTI2_IRQHandler                   = Default_Handler
#pragma weak EXTI3_IRQHandler                   = Default_Handler
#pragma weak EXTI4_IRQHandler                   = Default_Handler
#pragma weak DMA1_Channel1_IRQHandler           = Default_Handler
#pragma weak DMA1_Channel2_IRQHandler           = Default_Handler
#pragma weak DMA1_Channel3_IRQHandler           = Default_Handler
#pragma weak DMA1_Channel4_IRQHandler           = Default_Handler
#pragma weak DMA1_Channel5_IRQHandler           = Default_Handler
#pragma weak DMA1_Channel6_IRQHandler           = Default_Handler
#pragma weak DMA1_Channel7_IRQHandler           = Default_Handler
#pragma weak ADC1_2_IRQHandler                  = Default_Handler
#pragma weak USB_HP_CAN1_TX_IRQHandler      = Default_Handler
#pragma weak USB_LP_CAN1_RX0_IRQHandler     = Default_Handler
#pragma weak CAN1_RX1_IRQHandler    = Default_Handler
#pragma weak CAN1_SCE_IRQHandler    = Default_Handler
#pragma weak EXTI9_5_IRQHandler                 = Default_Handler
#pragma weak TIM1_BRK_IRQHandler    = Default_Handler
#pragma weak TIM1_UP_IRQHandler                 = Default_Handler
#pragma weak TIM1_TRG_COM_IRQHandler        = Default_Handler
#pragma weak TIM1_CC_IRQHandler                 = Default_Handler
#pragma weak TIM2_IRQHandler                    = Default_Handler
#pragma weak TIM3_IRQHandler                    = Default_Handler
#pragma weak TIM4_IRQHandler                    = Default_Handler
#pragma weak I2C1_EV_IRQHandler                 = Default_Handler
#pragma weak I2C1_ER_IRQHandler                 = Default_Handler
#pragma weak I2C2_EV_IRQHandler                 = Default_Handler
#pragma weak I2C2_ER_IRQHandler                 = Default_Handler
#pragma weak SPI1_IRQHandler                    = Default_Handler
#pragma weak SPI2_IRQHandler                    = Default_Handler
#pragma weak USART1_IRQHandler                  = Default_Handler
#pragma weak USART2_IRQHandler                  = Default_Handler
#pragma weak USART3_IRQHandler                  = Default_Handler
#pragma weak EXTI15_10_IRQHandler   = Default_Handler
#pragma weak RTCAlarm_IRQHandler    = Default_Handler
#pragma weak USBWakeUp_IRQHandler   = Default_Handler
#pragma weak TIM8_BRK_IRQHandler    = Default_Handler
#pragma weak TIM8_UP_IRQHandler                 = Default_Handler
#pragma weak TIM8_TRG_COM_IRQHandler        = Default_Handler
#pragma weak TIM8_CC_IRQHandler                 = Default_Handler
#pragma weak ADC3_IRQHandler                    = Default_Handler
#pragma weak FSMC_IRQHandler                    = Default_Handler
#pragma weak SDIO_IRQHandler                    = Default_Handler
#pragma weak TIM5_IRQHandler                    = Default_Handler
#pragma weak SPI3_IRQHandler                    = Default_Handler
#pragma weak UART4_IRQHandler                   = Default_Handler
#pragma weak UART5_IRQHandler                   = Default_Handler
#pragma weak TIM6_IRQHandler                    = Default_Handler
#pragma weak TIM7_IRQHandler                    = Default_Handler
#pragma weak DMA2_Channel1_IRQHandler           = Default_Handler
#pragma weak DMA2_Channel2_IRQHandler           = Default_Handler
#pragma weak DMA2_Channel3_IRQHandler           = Default_Handler
#pragma weak DMA2_Channel4_5_IRQHandler         = Default_Handler

void Default_Handler(void) {
        while (1) {}
}
```

startup.c

5.2.5 Final steps

These steps are creating a source file “main.c” and finish works on the Makefiles.

5.2.5.1 Source main.c

Create a file called “main.c” in the “src” directory, and paste the following demo code into it:

```
/*****************************************************************************
 * Test-program for Olimex “STM32-H103”, header board for “STM32F103RBT6”.
 * After program start green LED (LED_E) will blink.
 *
 * Program has to be compiled with optimizer setting "-O0".
 * Otherwise delay via while-loop will not work correctly.
 * Setup compiler optimizer setting to "-O0" in file "Makefile.common"
 *****************************************************************************/

#include "stm32f10x.h"
#include "stm32f10x_rcc.h"
#include "stm32f10x_gpio.h"

int main(int argc, char *argv[])
{
	GPIO_InitTypeDef GPIO_InitStructure;
	u32 delay;

	/* GPIOC Periph clock enable */
	RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC, ENABLE);

	/* Configure PC12 to mode: slow rise-time, pushpull output */
	GPIO_InitStructure.GPIO_Pin = GPIO_Pin_12; // GPIO No. 12
	GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz; // slow rise time
	GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP; // push-pull output
	GPIO_Init(GPIOC, &GPIO_InitStructure); // GPIOC init

	while(1)
	{
		/* make some float calculations */
		float x = 42, y = 23, z = 7;
		int i = 0;
		for ( i = 0; i < 6; i++ )
		{
			z = (x*y)/z;
		};

		/* GPIO PC12 set, pin=high, LED_E off */
		GPIOC->BSRR = GPIO_BSRR_BS12;
		/*GPIO_WriteBit(GPIOC,GPIO_Pin_12,Bit_SET);*/

		/* delay --> compiler optimizer settings must be "-O0" */
		delay=500000;
		while(delay)
			delay--;

		/* GPIO PC12 reset, pin=low, LED_E on */
		GPIOC->BSRR  = GPIO_BSRR_BR12;
		/*GPIO_WriteBit(GPIOC,GPIO_Pin_12,Bit_RESET);*/

		/* delay --> compiler optimizer settings must be "-O0" */
		delay=500000;
		while(delay)
			delay--;
	}
}
```

main.c

This code is just sample code for test purposes, for playing around with compiler options and for debugger tests.

5.2.5.2 Source Makefile

Application is build into a static library named “app.a”.
Create a file called “Makefile” in the “src” directory and paste the following text into it:

```
include ../Makefile.common

OBJS+=startup.o
OBJS+=main.o

all: src

src: app.a

app.a: $(OBJS)
	$(AR) cr app.a $(OBJS)

.PHONY: src clean
clean:
	rm -f app.a $(OBJS)
```

Makefile

5.2.5.3 Final Makefile

Create a file called “Makefile” in your projects top level directory and paste the following text into it:

```
include Makefile.common
LDFLAGS=$(COMMONFLAGS) -fno-exceptions -ffunction-sections -fdata-sections -L$(LIBDIR) -nostartfiles -Wl,--gc-sections,-Tlinker.ld

LDLIBS+=-lm
LDLIBS+=-lstm32

STARTUP=startup.c
all: libs src
	$(CC) -o $(PROGRAM).elf $(LDFLAGS) \
		-Wl,--whole-archive \
			src/app.a \
		-Wl,--no-whole-archive \
			$(LDLIBS)
	$(OBJCOPY) -O ihex   $(PROGRAM).elf $(PROGRAM).hex
	$(OBJCOPY) -O binary $(PROGRAM).elf $(PROGRAM).bin
#Extract info contained in ELF to readable text-files:
	arm-none-eabi-readelf -a $(PROGRAM).elf > $(PROGRAM).info_elf
	arm-none-eabi-size -d -B -t $(PROGRAM).elf > $(PROGRAM).info_size
	arm-none-eabi-objdump -S $(PROGRAM).elf > $(PROGRAM).info_code
	arm-none-eabi-nm -t d -S --size-sort -s $(PROGRAM).elf > $(PROGRAM).info_symbol

.PHONY: libs src clean
libs:
	$(MAKE) -C libs $@

src:
	$(MAKE) -C src $@

clean:
	$(MAKE) -C src $@
	$(MAKE) -C libs $@
	rm -f $(PROGRAM).elf $(PROGRAM).hex $(PROGRAM).bin $(PROGRAM).info_elf $(PROGRAM).info_size
	rm -f $(PROGRAM).info_code
	rm -f $(PROGRAM).info_symbol
```

Makefile

5.2.6 Build project

Now all is prepared for the first built. Execute following commands:

```
make clean
make
```

Elf, bin and hex file should have been built:

- “main.elf”, ELF file containing debug information, used for debug and other purposes.
- “main.bin”, raw binary file for flashing the MCU.
- “main.hex”, the binary file in Intel HEX for flashing the MCU.
- “main.info_*” are files containing additional information about headers, sections, size ...

5.2.7 Flash, run and debug

See page 14 and following pages for this topic.

6 Additional Tools

There exist some smart tools that ease programmers task after some practice. The tools and it's documentation can be installed via the package manager of your GNU/Linux system. This may not install the latest versions, but these versions should be usable and easy to maintain.

The use of these tools is recommended.

6.1 Doxygen

A tool that can generate documentation from source code in HTML, hyper-linked PDF and some other formats.

Postulate is that some documentation is done when the code is created, at the point of time you now your code best and this kind of work takes least time...

6.2 Git

A distributed revision control system not dependent on network access or a central server.
The use of a revision control system:

- facilitates to keep an overview about changes and revisions of software projects
- stores the sources and changes in a data base called repository
- enables teams to work on software projects

Introduction about revision control and Git:
http://tom.preston-werner.com/2009/05/19/the-git-parable.html

Documentation:
http://git-scm.com/documentation

Tutorial:
http://schacon.github.com/git/gittutorial.html

Pro Git:
http://progit.org/book/

6.3 Terminal emulation

For communicating with the MCU use a serial terminal emulation program like picocom (Minicom or CuteCom, ...).

Create a file called “run_picocom.sh” in your lokal script directory and paste the following text into it:

```
(xterm -geometry 40x50+0+0 -e 'picocom -b 9600 -d 8 -f n /dev/ttyACM0; bash' &)
```

run_picocom.sh

To run terminal emulation call script. Adjust picocom parameters to needs if necessary. Devicename “ttyACM0” refers to a STM32 VCP (Virtual Com Port) device.

7 IDE

An **I**ntegrated **D**evelopment **E**nvironments to handle all those tools described before may be:

- installed and configured.

or

- build on our own by making scripts and by arranging terminal windows and favorite programs on the screen.

It is just a matter of taste and habit. Below see about how to install and configure a popular IDE.

7.1 Eclipse

Eclipse has lots of features, all previous mentioned packages can be integrated. After some familiarization the user interface will appear well-arranged.
Writing and debugging C-code and simultaneously having a look at the assembler level and registers is possible, even the ability to set breakpoints at assembler level.

Other IDE's like Codeblocks , Codelite or Geany at the moment (january 2012) do not have all these abilities. KDevelop and Anjuta were not tested because the installation of a somewhat up to date system was not possible on Ubuntu with reasonable expense.
If debugging via a discrete frontend to GDB is an option, KDbg or DDD may also be a alternative in alliance with any of the above mentioned IDE's.

7.1.1 Install

For installing the IDE a Java Runtime Environment (JRE) has to exist on the system.
Only settings deviating from defaults are mentioned, for install user privileges are sufficient.

Three compressed files must be downloaded (filenames given may be out of date, the download of the Doxygen frontend is for archive purposes only):
Eclipse IDE for C/C++ Developers

- eclipse-cpp-indigo-SR1-incubation-linux-gtk.tar.gz
  available at: http://www.eclipse.org/downloads/

Eclipse CDT (C/C++ Development Tooling)

- cdt-master-8.0.1.zip
  available at: http://eclipse.org/cdt/

Doxygen frontend plug-in for eclipse

- eclox_0.8.0.zip
  available at: http://home.gna.org/eclox/

First step → install Eclipse IDE (from file):

1. Unzip file containing compressed Eclipse IDE to an empty directory and copy it's content to ~/eclipse.
2. Create a shortcut to ~\eclipse\eclipse or add program to the GNOME-menue.
3. Start eclipse
4. Eclipse will ask for a working directory. Enter your projects directory “~/22_ARM-Firmware”.

Second step → install some extensions available via Eclipse CDT (from file):

5. Help → Install New Software → Add
6. Enter CDT in field “Name:” and path to Eclipse CDT archive in field “Location:”.
7. Set checkbox “Hide items that are already installed” and open selection trees.
8. Set checkbox “C/C++ Debugger Services Framework (DSF) Examples”.
9. Set checkbox “C/C++ GDB Hardware Debugging”.
10. Set checkbox “C99 LR Parser SDK”.
11. Set checkbox “Eclipse Debugger for C/C++”.
12. Confirm your choices, accept license agreement.

13. Re-start eclipse.

Third step → install Doxygen frontend (via direct download):

14. → Help → Install New Software
15. Enter “http://download.gna.org/eclox/update” in field “Work with:”
16. Press enter and open selection tree.
17. Set checkbox “Eclox”.
18. Confirm your choices, accept license agreement.
19. Re-start eclipse.
20. File → Exit

7.1.2 Create project

Create project for use with “make” inside the IDE.

1. Start eclipse
2. Select “Workbench” icon → left mouse-click
3. Window → Open Perspective → Other → C/C++
4. File → New → C Project
5. “Project name”: 0003_Test_Eclipse, “Project Type”: Makefile.. Empty.., “Toolchains”: other.. → Finish

7.1.3 Configure workspace

Global settings concerning the workspace and all projects.

1. Window → Preferences → General → Workspace → set checkbox “Save automatically before build” → Apply → OK
2. There exist lot of settings that may be useful. Enable text folding, tab-with, code style settings, code templates, colours, perspectives, spell checking, keys (shotcuts), template default values, code analysis… Collect some experience and use or ignore them.

7.1.4 Configure project

Make project settings for comfortable use of the IDE and import files.

First step → configure

1. Project → clear checkbox at menu item “Build Automatically”
2. Project → Properties → C/C++ Build → Settings → set checkbox “GNU Elf Parser” (no other checkboxes set) → OK

Second step → import existing files into project

3. File → Import → General → File System → Next
4. “From directory:” Browse → select “0002_Test_Template” → set checkbox at appearing project → Finish
5. A double click on any file inside the “Project Explorer” window will open a file in the editor.

Third step → make

6. Project → Clean → Clean projects selected.. → set checkbox at “0003_Test_Eclipse” → clear checkbox “Start a build immediately” → OK
7. Project → Build All
8. See result of the previous operation inside the “Console” window. A make should have happened.

7.1.5 Configure external tools

Input settings to start and abort OpenOCD from within the IDE.

1. Run → External Tools → External Tools Configurations
2. Double click on “Program”, some more input boxes will appear.
3. Set “Name:” to “OpenOCD_0003”.
4. Goto Tab Main
5. At input area “Location” click on “Browse File System” and select path to OpenOCD (i.e. “/usr/local/bin/openocd”).
6. At input area “Working Directory” click on “Browse Workspace...” select “0003_Test_Eclipse” → OK.
7. At input area “Arguments:” enter “-f openocd.cfg”.
8. Goto Tab Build → clear checkbox “Build before launch”.
9. Goto Tab Common → at input area “Display in favourites..” set checkbox “External Tools” → Apply
10. Double click on “Program”, some more input boxes will appear.
11. Set “Name:” to “Terminal emulation”.
12. Goto Tab Main
13. At input area “Location” click on “Browse File System” and select path to shell-script (i.e. “/home/userA/scripts/run_picocom.sh”).
14. Goto Tab Build → clear checkbox “Build before launch”.
15. Goto Tab Common → at input area “Display in favourites..” set checkbox “External Tools” → Apply → Close

7.1.6 Configure debugger

The debugger inside the IDE needs some project specific settings. These settings also depend on the way OpenOCD is configured. See previous chapter about OpenOCD configuration for appropriate settings and the OpenOCD manual for pros and cons.

7.1.6.1 Hardware reset

This is the **recommended** setting. It should be used when the reset signal of the MCU is available at the JTAG connector of the device in development (OpenOCD setting “reset_config srst_only”). The device is **reset via hardware signaling or “SYSRESETREQ” interrupt**.

When reset is done via hardware use following settings:

1. Window → Open Perspective → Other... → Debug
2. Run → Debug Configurations
3. Double click on “GDB Hardware Debugging”, some more input boxes will appear.
4. Select “Name:” → enter “**0003_Debug**”
5. At “C/C++ Applikation:” click on “Search Project” → select “main.elf” → Apply
6. At bottom see tex“Using GDB … Hardware Debugging Launcher..” click on “Select other...”.
7. Set checkbox “Use configuration specific settings”
8. Select “Standard GDB Hardware Debugging Launcher” if not already selected → OK
9. Select the “Debugger” tab → at “GDB Setup” → Browse → select path to debugger program that should be used, set to “/opt/arm-2012.03/bin/arm-none-eabi-gdb” ; set listbox “Command Set:” to “Standard”
10. At “Remote Target” set checkbox “Use remote target”
11. Set “JTAG Device:” to “Generic TCP/IP”, set “Host name..:” to “localhost” and “Port number:” to “3333” → Apply
12. Select the “Startup” tab

13. At “Initialization Commands” clear checkbox “Reset and Delay..:” and clear checkbox “Halt”, in text box enter following line:
    monitor reset init
14. At “Run Commands:” in text box enter following line (optional):
    break main
15. Select “Common” tab → at input area “Display in favourites..” set checkbox “Debug”.
16. Apply

Note (install step 11.): when trying to set “JTAG Device:” to “OpenOCD (via pipe)” debugging was not possible, searching the web yielded no result. So this may be a topic for future improvements.
Duplicate previous debug configuration by right click on this configuration and selecting “Duplicate”.

a) Select “Name:” → Enter “**0003_Flash+Debug**”
b) Select the “Startup” tab
c) At “Initialization Commands” in text box enter following lines:
   monitor reset init
   monitor flash probe 0
   monitor stm32f1x mass_erase 0
   monitor flash write_bank 0 main.bin 0
d) Apply → Close

7.1.6.2 Software reset

This setting is **not recommended**. It should be used when the reset signal of the MCU is not available at the JTAG connector of the device in development and the “SYSRESETREQ” interrupt also is not used. The device is **reset via JTAG commands**.

When reset is done via JTAG commands replace steps 13. and c) in previous section:

13. At “Initialization Commands” clear checkbox “Reset and Delay..:” and clear checkbox “Halt”, in text box enter following line:
    monitor soft_reset_halt

c) At “Initialization Commands”, in text box enter following lines:
   monitor soft_reset_halt
   monitor flash probe 0
   monitor stm32f1x mass_erase 0
   monitor flash write_bank 0 main.bin 0

7.1.7 First steps

1. Switch to the C/C++ perspective
2. Open file main.c
3. Set breakpoint at line 60 and line 62
4. Switch to the Debug perspective
5. Run → External Tools → OpenOCD (or click on icon)
6. Click on arrow near icon Debug (the bug) → select “0003_Flash+Debug”
7. Press F8 → Press F8 → ...(stepping through the code, the green LED should toggle)

7.1.8 Hints

There exists a lot of documentation and hints about Eclipse – and it seems to be necessary. This chapter intends to contain useful know-how about first and standard steps. Hope it's content will increase by time and with successful usage of the IDE.

Debuggen mit GDB (Gnu DeBugger) unter Eclipse :
http://homepages.thm.de/~bbdw58/anleit/debuggen.pdf

Dokumentieren mit Eclipse und Doxygen:
http://homepages.thm.de/~bbdw58/anleit/doxygen.pdf

EGit/User Guide:
http://wiki.eclipse.org/EGit/User_Guide

Git with Eclipse (EGit) - Tutorial
http://www.vogella.de/articles/EGit/article.html

8 Bugs and Workarounds

8.1 GCC toolchain

Bug in Newlib C Library (1.18.0-sg++) (stdlib.h) included in “Sourcery CodeBench Lite” – when including function “strtoull” in program code, communication via USART switches to baudrate x 4, sometimes HardFault_Handler interrupt occurs. Function “strtoul” seems to be not affected.
Workaround:

1. Do not use corrupted functions from Newlib C Library or do not use this lib at all.

The same effect occurs when adding following content to the source code:
long long test1 = 10, test2 = 4, test3;
test3 = test1 / test2;
As soon as the code “division of a long long variable” exists inside the program (no matter where), the USART behaves strange. Maybe the above mentioned function “strtoull” contains a 64-bit division too. The manual "Using the GNU Compiler Collection, chapter 6.9" tells that long long divisions are open coded and are available only on machines 'providing special support'.
Workaround:

1. **do not use long long division.**

It would be just of interest how the code of the division can alter the program in a way that:
+ everything still works and debug functionality is fine
+ even the result of the long long division is OK
- the baudrate changes to baudrate x 4 (which is defined by the content of a CPU-register, not by the translated code itself)

8.2 IDE -eclipse

Bug 351549 - IDE shows source has errors, but it builds without errors. Errors in “Problems” terminal look like: “Type xxx could not be resolved ... Semantic Error”. This error happens on “Eclipse Platform 3.7.1.r37” with “Eclipse CDT 8.0.0.”. The IDE can come into this (buggy) state, but it must not – seems to be depending on making project or workspace settings.
Two workarounds known:

1. Switch code analysis off (**recommended**): Window → Preferences → C/C++ → Code Analysis → clear all checkboxes → Apply → OK → delete all errors in “Problems” terminal.
2. Add include path to project (not recommended): Project → Properties → C/C++ General → Paths and Symbols → Includes → GNU C → Add “usr/include” → Apply → OK

9 To do's

The big challenge now is to fill “main.c” with some useful code and improve the toolchain when things are not satisfying or buggy.

10 Credits and Reference

(1) How-to by Johan Simonsson:
http://fun-tech.se/stm32/index.php

(2) How-to by Geoffrey McRae (unfortunately this page is not on-line any more):
http://stm32.spacevs.com/index.php

11 Revision history

Document revision history:

Date	Revision	Changes
2012-01-27	0.7.2	Initial release.
2012-02-16	0.7.3	Added external tool “Run terminal emulation” to eclipse.
2012-02-28	0.7.4	Changes in OpenOCD install and configuration.
2012-08-16	0.7.5	Extended “Bugs and Workaround” chapter. GCC toolchain update.
2012-08-29	0.7.6	Added content to “Bugs and Workaround, GCC toolchain”.
2012-11-09	0.8.1	Upgrade to StdPeriph_Lib_V3.6.1, include USB full-speed device library.
2013-01-18	0.8.2	Upgrade chapter “Install JTAG device”, Ubuntu 12.04

12 Appendix

12.1 Cortex-M3

Collection of Cortex-M3 related documents.

12.1.1 Intro's

The Insider's Guide To The STM32 ARM Based Microcontroller
http://www.hitex.com/fileadmin/pdf/insiders-guides/stm32/isg-stm32-v18d-scr.pdf

Getting started with STM32F10xxx hardware development - AN2586
http://www.st.com/internet/com/TECHNICAL_RESOURCES/TECHNICAL_LITERATURE/APPLICATION_NOTE/CD00164185.pdf

Discovering the STM32 Microcontroller , Geoffrey Brown
http://homes.soic.indiana.edu/geobrown/index.cgi/teaching
http://www.cs.indiana.edu/~geobrown/book.pdf

12.1.2 Architecture

Cortex-M3 Technical Reference Manual
http://infocenter.arm.com/help/topic/com.arm.doc.ddi0337i/DDI0337I_cortexm3_r2p1_trm.pdf

ARMv7-M Architecture Reference Manual
http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0403c/index.html

Errata: Cortex-M3/Cortex-M3 with ETM (AT420/AT425)
http://infocenter.arm.com/help/topic/com.arm.doc.eat0420d/Cortex-M3-Errata-r2p1-v3.pdf

Cortex-M System Design Kit Technical Reference Manual
http://infocenter.arm.com/help/topic/com.arm.doc.ddi0479b/DDI0479B_cortex_m_system_design_kit_r0p0_trm.pdf

AN179 - Cortex-M3 Embedded Software Development
http://infocenter.arm.com/help/topic/com.arm.doc.dai0179b/AppsNote179.pdf

12.1.3 MCU

Datasheet STM32F103RB – DS5319
http://www.st.com/internet/com/TECHNICAL_RESOURCES/TECHNICAL_LITERATURE/DATASHEET/CD00161566.pdf

Errata sheet STM32F103x8/B medium-density device limitations - ES096
http://www.st.com/internet/com/TECHNICAL_RESOURCES/TECHNICAL_LITERATURE/ERRATA_SHEET/CD00190234.pdf

Reference manual STM32F103xx advanced ARM-based 32-bit MCU - RM0008
http://www.st.com/internet/com/TECHNICAL_RESOURCES/TECHNICAL_LITERATURE/REFERENCE_MANUAL/CD00171190.pdf

Cortex-M3 programming manual - PM0056
http://www.st.com/internet/com/TECHNICAL_RESOURCES/TECHNICAL_LITERATURE/PROGRAMMING_MANUAL/CD00228163.pdf

STM32F10xxx Flash memory microcontrollers programming manual - PM0075
http://www.st.com/internet/com/TECHNICAL_RESOURCES/TECHNICAL_LITERATURE/PROGRAMMING_MANUAL/CD00283419.pdf

AN2594 - EEPROM emulation
http://www.st.com/internet/com/TECHNICAL_RESOURCES/TECHNICAL_LITERATURE/APPLICATION_NOTE/CD00165693.pdf
http://www.st.com/internet/com/SOFTWARE_RESOURCES/SW_COMPONENT/FIRMWARE/an2594.zip

AN2824 – I2C optimized examples
http://www.st.com/internet/com/TECHNICAL_RESOURCES/TECHNICAL_LITERATURE/APPLICATION_NOTE/CD00209826.pdf
http://www.st.com/internet/com/SOFTWARE_RESOURCES/SW_COMPONENT/FIRMWARE/an2824.zip

12.2 Links

Coding Style - how the boss likes the C code in the kernel to look
http://www.kernel.org/doc/Documentation/CodingStyle

Eclipse example project for ST STM32F103RB – blinking LED, simplified printf_() function
http://www.freddiechopin.info/index.php/en/download/

Q&A for professional and enthusiast programmers
http://stackoverflow.com/

www.mikrocontroller.net
http://www.mikrocontroller.net/articles/STM32F10x_Standard_Peripherals_Library
http://www.mikrocontroller.net/articles/STM32

About OpenOCD
http://elk.informatik.fh-augsburg.de/pub/epjournal-1/oocd.html

embedded projects GmbH - embedded journal, tools & shop
http://shop.embedded-projects.net/

The Open Development Environment for embedded application (ODeV)
http://www.stf12.org/developers/Home.html