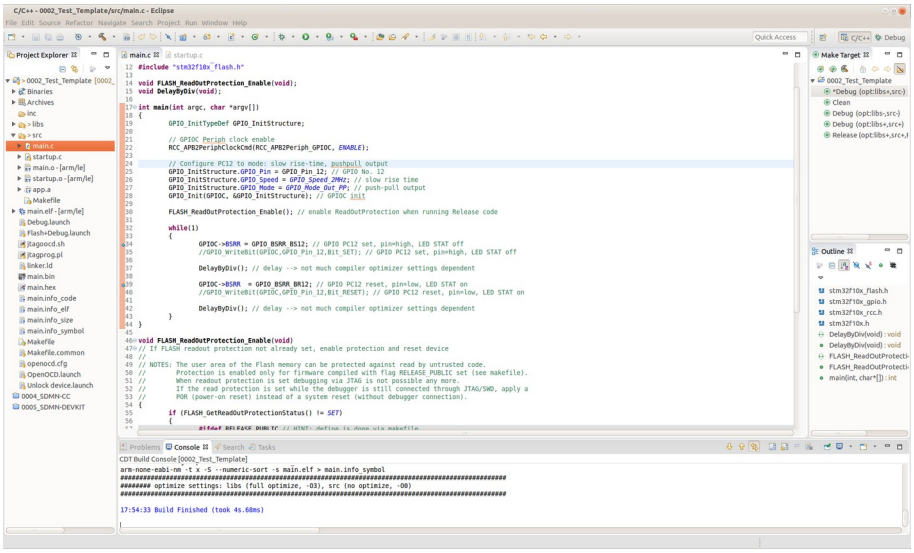
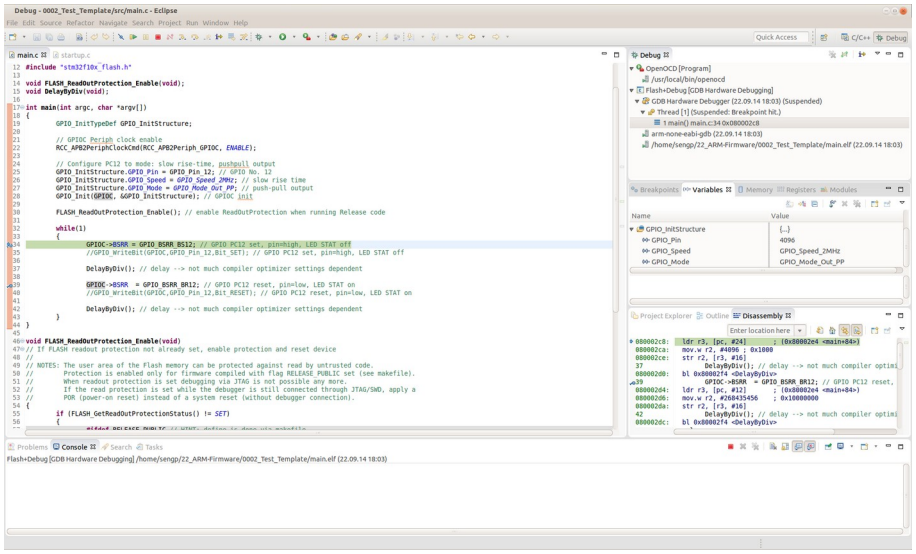
How-to manual

Installing a toolchain for Cortex-M3/STM32 on GNU/Linux

Version 1.0.4, 2020-10-28





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 GNU/Linux (installed and tested on *Ubuntu 10.04*, *Ubuntu 12.04*, *LinuxMint 17*, *Linux Mint 20*).
All packages used, except the GCC toolchain, are open source.
For this part a free, unlimited and up to date version of “Sourcery CodeBench” or “GNU Tools for ARM Embedded Processors” (both based on the GNU tools) are used in order to ease the install and build procedure. *LinuxMint17* includes the full GCC toolchain in it's repository, *LinuxMint20* not.
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 (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
The current version of this manual is available at : http://www.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, …
- STM32F103RET6 (512KB Flash, 64KB RAM) mounted on “STM32-H103” board.

The example code in this manual is adapted to STM32F103RBT6 with comments for the ...RET6.

3 Software

The toolchain consists of following packages:

- OpenOCD
- stm32flash by Geoffrey McRae (2)
- GCC toolchain for build and debug
- STM32F10x standard peripheral library
- Project template and makefile by Geoffrey McRae (2)
- Eclipse IDE and some utitlities
- (Git)
- (Doxygen)

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.

When using Linux Mint 20 install OpenOCD using the repository and continue at 4.1.2

Else OpenOCD downloads and documentation can be found at:
http://openocd.org/documentation/
http://sourceforge.net/projects/openocd/files/openocd/

4.1.1 Download, build and install

Create a temporary directory:
```
mkdir ~/temp/stm32/ -p
cd ~/temp/stm32/
```

Download the documentation using one of the above mentioned links. Think about a **structure to store the documentation** of this toolchain in. Be aware to get the manual version that fits to the program version.
Down see how to install the software out of a repository. From above links download of source releases is also possible.

Install some packages that are needed to build the program (this steps are not mandatory, maybe some packages are already installed, maybe some packages are still missing, look at the errors and hints that may occur when the program is compiled and installed) :
```
sudo apt-get install libusb-1.0-0 libusb-1.0-0-dev \
   libtool pkg-config \
   autoconf automake texinfo
```

Get and compile the program (these instructions will install version 0.8.0 of the package):
```
mkdir -p ~/temp/stm32/stm32-tools
cd ~/temp/stm32/stm32-tools
git clone git://git.code.sf.net/p/openocd/code OpenOCD
cd OpenOCD
#git tag
git reset --hard v0.8.0
./bootstrap
./configure --enable-ftdi
make
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`

Using Linux Mint 20 you should see something like this:
...
`Bus 001 Device 009: ID 15ba:002b Olimex Ltd. ARM-USB-OCD-H JTAG+RS232`
...
everything is OK and continue at 4.1.3.

But if you see following result:
...
`Bus 001 Device 010: ID 15ba:002b Olimex Ltd.`
…
a rules file has to be added to the system.

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.
That's why the device can not be identified automatically by Ubuntu 10.04. **To make the JTAG and RS232 serial port of the device usable, a rules file has to be added to the system.** Content of file and further 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

After copying the file to the file system detach an re-connect the JTAG device, to enable the detection of the changed rules by the operating system.

check with:
`dmesg | grep usb`

that the device 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/ftdi/olimex-arm-usb-ocd-h.cfg”

The board section content for Olimex STM32-H103 is copied 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
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 included as a cross reference in:
“/usr/local/share/openocd/scripts/board/olimex_stm32_h103.cfg” and refers to:
“/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
# http://www.olimex.com/dev/arm-usb-ocd-h.html
#
interface ftdi
ftdi_device_desc "Olimex OpenOCD JTAG ARM-USB-OCD-H"
ftdi_vid_pid 0x15ba 0x002b
ftdi_layout_init 0x0c08 0x0f1b
ftdi_layout_signal nSRST -oe 0x0200
ftdi_layout_signal nTRST -data 0x0100 -noe 0x0400
ftdi_layout_signal LED -data 0x0800

#board configuration#############################################################
# Adjust Work-area size (RAM size) according to MCU in use:
# STM32F103RB --> 20KB
set WORKAREASIZE 0x5000
# STM32F103RE --> 64KB
#set WORKAREASIZE 0x10000

# 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
reset_config none

#target configuration############################################################
source [find target/stm32f1x.cfg]
```

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 - Getting started with STM32F 10xxx hardware development". For further details about the bootloader please see application note "AN2606 - STM32 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, originally written 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/stm32/stm32-tools/
svn checkout http://stm32flash.googlecode.com/svn/trunk/ stm32flash
cd ~/temp/stm32/stm32-tools/stm32flash/
```

Compile the program:

```
make
```

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

```
sudo make install
```

Following example command would download the 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/st-web-ui/static/active/en/st_prod_software_internet/resource/technical/software/demo_and_example/stsw-mcu005.zip
Documentation can be downloaded from:
http://www.st.com/st-web-ui/static/active/en/resource/technical/document/user_manual/CD00171488.pdf
See the manual for information about install and use of the program.

4.3 GCC toolchain

4.3.1 Repository install, Linux Mint17/18 only

In recent distributions such as *LinuxMint 17* the gcc cross-compiler toolchain is contained in the repository. If available, install by:

```
sudo apt-get install gcc-arm-none-eabi binutils-arm-none-eabi \
   gdb-arm-none-eabi libnewlib-arm-none-eabi
```

Cause of an error of the distribution (status 2014-09-16) the *gdb-arm-none-eabi* package must be installed forcing an overwrite. Check if this matters on the destination system before install:

```
sudo apt-get -o Dpkg::Options::="--force-overwrite" install gdb-arm-none-eabi
```

This might be the fastest way to install the gcc toolchain, the force overwrite error will clear away in the future. Hope the quality of the provided documentation will reach the level of the external installs soon.

4.3.2 External install (e.g. Linux Mint 20)

In order to keep installation simple a free, pre-built, well documented and unlimited version of "**GNU Tools for ARM Embedded Processors**", based on the GNU tools may be used when the repository does not contain the toolchain or the above mentioned drawbacks are intolerable.
Information and downloads can be found at:
https://developer.arm.com/tools-and-software/open-source-software/developer-tools/gnu-toolchain/gnu-rm/downloads
Download "Linux x86_64 Tarball". This tool suite also includes FPU support, that is not needed when using STM32F103 devices.

Alternatively "Sourcery CodeBench Lite Edition" can be used. Information and downloads can be found at:
http://www.mentor.com/embedded-software/sourcery-tools/sourcery-codebench/editions/lite-edition
Installation and usage is identical to "GNU Tools for ARM Embedded Processors".

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).

Installation
These instructions will install version "GNU Arm Embedded Toolchain 2020-q2-update" .
Included are:

- GNU Binary Utilities
- GNU C & C++ Compilers
- GNU Debugger
- Newlib C Library

Extract the tarball (e.g.: arm-2012.09-63-arm-none-eabi-i686-pc-linux-gnu.tar.bz2) and then copy the extracted contents to "/opt" as root.
Add path e.g.: "/opt/gcc-arm-none-eabi-9-2020-q2-update/bin" to your environment, therefore append following lines at end of file ".profile" located at your home directory. Adjust pathname to name of extracted content.

```
…
# Product Begin: arm Developer ARM-Compiler
# set PATH to arm Developer ARM-Compiler
# see: https://developer.arm.com/tools-and-software/open-source-software/developer-tools/gnu-toolchain/gnu-rm/downloads
export PATH=$PATH:/opt/gcc-arm-none-eabi-9-2020-q2-update/bin
# Product End: arm Developer ARM-Compiler
```

After exporting the path reload the ".profile" file without the need for logging out and back in again by:
. ~/.profile
If this may not work → log out and back in again.

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

Documentation of the "Sourcery CodeBench" toolchain in PDF format is available in directory:
"/opt/arm-2012.09/share/doc/arm-arm-none-eabi/pdf "
Documentation of the "Sourcery CodeBench" toolchain in HTML format is available in directory:
"/opt/arm-2012.09xx/share/doc/arm-arm-none-eabi/html"

4.3.3 Check installation

To verify that install was successful and 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.

When following error occurs:
"error while loading shared libraries: libncurses.so.5: cannot open shared object file: No such file or directory"
install library:
sudo apt install libncurses5

and check if install is successful now by entering:
arm-none-eabi-gcc -v
The last line of the output should contain the actual compiler version.

Be sure to modify the path to the build and debug suite within eclipse (or other IDE) when updating the GCC toolchain if the toolchain was not installed via the repository or path was not exported.

5 Basic project

To check whether the compiler works, we should create and compile a small program/project. 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 0002_Test_Template

This project is for test purposes. 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

5.1.1 Libraries

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/st-web-ui/static/active/en/st_prod_software_internet/resource/technical/software/firmware/stsw-stm32054.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/st-web-ui/static/active/en/st_prod_software_internet/resource/technical/software/firmware/stsw-stm32121.zip
This file contains the USB library version 4.0.0 and the “STM32F10x standard peripheral library” version 3.6.1. without example code.

5.1.1.1 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".

5.1.1.2 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/...".

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

Note: 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.

5.1.1.3 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.

5.1.2 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.

5.1.2.1 Common Makefile

This makefile has to be included into all other makefiles. It contains variable setup for the build procedure.
Optimization and conditional compiling of libraries (OptLIB) and sources (OptSRC) can be controlled by use of parameters. Invoke make by following statement:

make OptLIB=x OptSRC=y all tshow

x	y	Description
0	0	no optimize, reduce compilation time and make debugging produce the expected results (default).
1	1	optimize, reduce code size and execution time, without much increase of compilation time.
2	2	optimize, reduce code size and execution time, without much increase of compilation time.
3	3	optimize, turns on all optimizations, further increase of compilation time.
s	s	optimize for size, enables all ‘-O2’ optimizations that do not typically increase code size and other code size optimizations.
	4	Same as 3, additionally a define for conditional compiling is set: *-D RELEASE_PUBLIC*. Define may be used to automatically include readout protection code when compiling release version.

Create a file at the top level of the project called “Makefile.common” and paste the following text into it:

```
# include Makefile

#This file is included in the general Makefile, the libs Makefile and the src Makefile
#Different optimize settings for library and source files can be realized by using arguments
#Compiler optimize settings:
# -O0 no optimize, reduce compilation time and make debugging produce the expected results.
# -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.
#Recommended optimize settings for release version: -O3
#Recommended optimize settings for debug version: -O0
#Valid parameters :
# OptLIB=0 --> optimize library files using the -O0 setting
# OptLIB=1 --> optimize library files using the -O1 setting
# OptLIB=2 --> optimize library files using the -O2 setting
# OptLIB=3 --> optimize library files using the -O3 setting (default)
# OptLIB=s --> optimize library files using the -Os setting
# OptSRC=0 --> optimize source files using the -O0 setting
# OptSRC=1 --> optimize source files using the -O1 setting
# OptSRC=2 --> optimize source files using the -O2 setting
# OptSRC=3 --> optimize source files using the -O3 setting
# OptSRC=s --> optimize source files using the -Os setting
# OptSRC=4 --> optimize source files using the -O3 setting, conditional compiling by use of define RELEASE_PUBLIC (default)
# all --> build all
# libs --> build libs only
# src --> build src only
# clean --> clean project
# tshow --> show optimize settings
#Example:
# make OptLIB=3 OptSRC=0 all tshow

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

#Adjust TypeOfMCU in use, see CMSIS file "stm32f10x.h"
#STM32F103RBT (128KB FLASH, 20KB RAM) --> STM32F10X_MD
TypeOfMCU=STM32F10X_MD
#STM32F103RET (512KB FLASH, 64KB RAM) --> STM32F10X_HD
#TypeOfMCU=STM32F10X_HD

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

COMMONFLAGS=-g -mcpu=cortex-m3 -mthumb
COMMONFLAGSlib=$(COMMONFLAGS)

#Commands for general Makefile and src Makefile
ifeq ($(OptSRC),0)
	COMMONFLAGS+=-O0
	InfoTextSrc=src (no optimize, -O0)
else ifeq ($(OptSRC),1)
	COMMONFLAGS+=-O1
	InfoTextSrc=src (optimize time+ size+, -O1)
else ifeq ($(OptSRC),2)
	COMMONFLAGS+=-O2
	InfoTextSrc=src (optimize time++ size+, -O2)
else ifeq ($(OptSRC),s)
	COMMONFLAGS+=-Os
	InfoTextSrc=src (optimize size++, -Os)
```

```
else ifeq ($(OptSRC),3)
		COMMONFLAGS+=-O3
		InfoTextSrc=src (full optimize, -O3)
else
		COMMONFLAGS+=-O3
		CFLAGS += -D RELEASE_PUBLIC
		InfoTextSrc=src (full optimize and readout protected, -O4)
endif
CFLAGS+=$(COMMONFLAGS) -Wall -Werror $(INCLUDE)
CFLAGS+=-D $(TypeOfMCU)
CFLAGS+=-D VECT_TAB_FLASH

#Commands for libs Makefile
ifeq ($(OptLIB),0)
		COMMONFLAGSlib+=-O0
		InfoTextLib=libs (no optimize, -O0)
else ifeq ($(OptLIB),1)
		COMMONFLAGSlib+=-O1
		InfoTextLib=libs (optimize time+ size+, -O1)
else ifeq ($(OptLIB),2)
		COMMONFLAGSlib+=-O2
		InfoTextLib=libs (optimize time++ size+, -O2)
else ifeq ($(OptLIB),s)
		COMMONFLAGSlib+=-Os
		InfoTextLib=libs (optimize size++, -Os)
else
		COMMONFLAGSlib+=-O3
		InfoTextLib=libs (full optimize, -O3)
endif
CFLAGSlib+=$(COMMONFLAGSlib) -Wall -Werror $(INCLUDE)
CFLAGSlib+=-D $(TypeOfMCU)
CFLAGSlib+=-D VECT_TAB_FLASH
```

Makefile.common

The makefile will use library “STM32F10x_StdPeriph_Lib_V3.6.1”. Code optimization must be 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. **Enable the correct define setting in following file** (an explanation of this define can also be found there):
/libs/STM32F10x_StdPeriph_Lib_V3.6.1/Libraries/CMSIS/Device/ST/STM32F10x/Include/**stm32f10x.h**
inside your current project directory.

5.1.2.2 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:

```
# libs Makefile

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

all: libs

libs: $(LIBS)

libstm32.a:
	@echo -n "Building $@ ..."
	@cd $(STMLIB)/CMSIS/Device/ST/STM32F10x/Source/Templates && \
		$(CC) $(CFLAGSlib) \
			system_stm32f10x.c
	@cd $(STMLIB)/STM32F10x_StdPeriph_Driver/src && \
		$(CC) $(CFLAGSlib) \
			-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) $(CFLAGSlib) \
#			-D"assert_param(expr)=((void)0)" \
#			-I../../CMSIS/Include \
#			-I../../CMSIS/Device/ST/STM32F10x/Include \
#			-I../inc \
#			*.c
	@$(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 tshow

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)

tshow:
	@echo "#########################################################################################################"
	@echo "######## optimize settings: $(InfoTextLib), $(InfoTextSrc)"
	@echo "#########################################################################################################"
```

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.

5.1.3 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 LENGTH to RAMsize of target MCU:*/
	/*STM32F103RBT --> 20K*/
	RAM	(RWX) : ORIGIN = 0x20000000	, LENGTH = 20K
	/*STM32F103RET --> 64K*/
	/*RAM	(RWX) : ORIGIN = 0x20000000	, LENGTH = 64K*/

	EXTSRAM  (RWX) : ORIGIN = 0x68000000	, LENGTH = 0

	/*Adust LENGTH to (FLASHsize - FeePROMsize) of target MCU:*/
	/*STM32F103RBT --> 126K*/
	FLASH	(RX)  : ORIGIN = 0x08000000	, LENGTH = 126K
	/*STM32F103RET --> 508K*/
	/*FLASH	(RX)  : ORIGIN = 0x08000000	, LENGTH = 508K*/

	/*Adust ORIGIN to (0x08000000 + (FLASHsize-FeePROMsize)) of target MCU*/
	/*and adust LENGTH to FeePROMsize allocated:*/
	/*STM32F103RBT --> 0x08000000+126K, 2K*/
	EEMUL	(RWX) : ORIGIN = 0x08000000+126K, LENGTH = 2K
	/*STM32F103RET --> 0x08000000+508K, 4K*/
	/*EEMUL	(RWX) : ORIGIN = 0x08000000+508K, LENGTH = 4K*/
}

_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

Lines at the top beginning with “RAM”, “FLASH” and “EEMUL” specify MCU memory.

This linker script includes examples for medium density (MD) devices with 128K flash and 20K RAM (STM32F103RBT) and high density (HD) devices with 512K flash and 64K RAM (STM32F103RET), it is setup for an STM32F103RBT.

2K (MD) or 4K (HD) flash memory is reserved for EEPROM emulation (2 pages x 1K/2K), see “AN2594 -EEPROM emulation” for available memory size and access mechanism.

5.1.4 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; /* end address of the stack section */

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

5.1.5 Final steps

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

5.1.5.1 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 (STAT) will blink, when jumper LED_E is closed.
 *
 * Running Release code will set ReadOutProtection (see down) via function FLASH_ReadOutProtection_Enable().
 * Do not run Release code until you know how to set back ReadOutProtection!
 *****************************************************************************************************************/

#include "stm32f10x.h"
#include "stm32f10x_rcc.h"
#include "stm32f10x_gpio.h"
#include "stm32f10x_flash.h"

void FLASH_ReadOutProtection_Enable(void);
void DelayByDiv(void);

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

	// 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

	FLASH_ReadOutProtection_Enable(); // enable ReadOutProtection when running Release code

	while(1)
	{
		GPIOC->BSRR = GPIO_BSRR_BS12; // GPIO PC12 set, pin=high, LED STAT off
		//GPIO_WriteBit(GPIOC,GPIO_Pin_12,Bit_SET); // GPIO PC12 set, pin=high, LED STAT off

		DelayByDiv(); // delay --> not much compiler optimizer settings dependent

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

		DelayByDiv(); // delay --> not much compiler optimizer settings dependent
	}
}

void FLASH_ReadOutProtection_Enable(void)
// If FLASH readout protection not already set, enable protection and reset device
//
// NOTES: The user area of the Flash memory can be protected against read by untrusted code.
//	Protection is enabled only for firmware compiled with flag RELEASE_PUBLIC set (see makefile).
//	When readout protection is set debugging via JTAG is not possible any more.
//		If the read protection is set while the debugger is still connected through JTAG/SWD, apply a
//	POR (power-on reset) instead of a system reset (without debugger connection).
{
	if (FLASH_GetReadOutProtectionStatus() != SET)
	{
		#ifdef RELEASE_PUBLIC // HINT: define is done via makefile
			FLASH_Unlock();
			if (FLASH_ReadOutProtection(ENABLE) != FLASH_COMPLETE) // set readout protection
			{
				// ERROR: could not program read protection
			}
			else
				NVIC_SystemReset(); // protection set --> reset device to enable protection
		#else
				// output warning message
		#endif
	}
}
```

```
void DelayByDiv(void)
// delay implemented by floating division
// not much compiler optimizer settings dependent
{
	float x=50.0f;

	while (x > 0.0001f)
		x = x/1.0001f; // delay loop
}
```

main.c

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

5.1.5.2 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:

```
# src Makefile

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 tshow

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

tshow:
	@echo "#######################################################################################################"
	@echo "######## optimize settings: $(InfoTextLib), $(InfoTextSrc)"
	@echo "#######################################################################################################"
```

Makefile

5.1.5.3 Final Makefile

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

```
# general Makefile

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 x -S --numeric-sort -s $(PROGRAM).elf > $(PROGRAM).info_symbol

.PHONY: libs src clean tshow

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

tshow:
	@echo "#########################################################################################################"
	@echo "######## optimize settings: $(InfoTextLib), $(InfoTextSrc)"
	@echo "#########################################################################################################"

#flash:
#	./jtagprog.pl
```

Makefile

5.2 Build project

Now all is prepared for the first built. Execute following commands:
make clean
make OptLIB=0 OptSRC=0 all tshow

5.3 Check results

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.

Text, elf, bin and hex file should have been built:

- “main.info_symbol”, file containing readable information about symbols.
- “main.info_code”, file containing the generated assembler code and interleaved C source-code.
- “main.info_size”, file containing information about the size of the generated code.
- “main.info_elf”, file containing readable information contained in the elf file.
- “main.bin”, raw binary file for flashing the MCU.
- “main.hex”, the binary file in Intel HEX format for flashing the MCU.
- “main.elf”, ELF file containing debug information, used for debug and other purposes.

Open the main.info_* files by use of an editor (gedit, pluma, ...), i.e.:
```
gedit main.info_size
```

For a closer look at the ELF-file use command “readelf” from “binutils”:
```
arm-none-eabi-readelf -A main.elf
```
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.elf
```

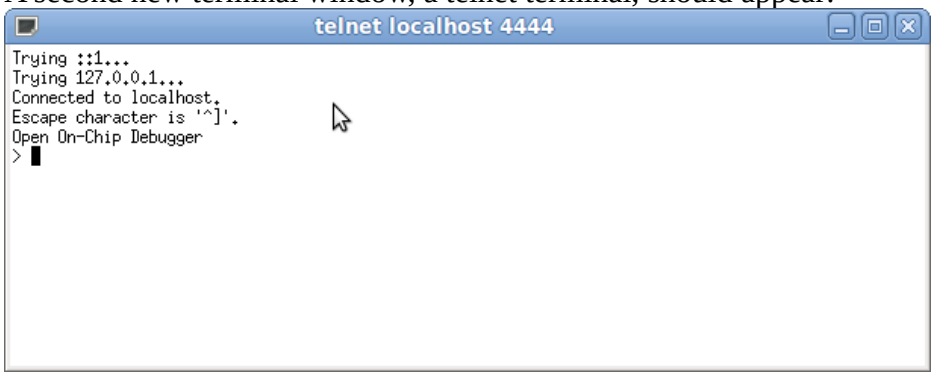
If you want to have a closer look at the previous steps, see (1). There you will also find some explanatory notes.

5.4 Flash and run

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

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.

5.5 Read protection

The user area of the Flash memory can be protected against read by untrusted code. The read protection is activated by setting the RDP option byte and then, by applying a system reset to reload the new RDP option byte.
Note: If the read protection is set while the debugger is still connected through JTAG/SWD, apply a POR (power-on reset) instead of a system reset (without debugger connection). **Do not read protect a device that still needs further debugging.**

The OpenOCD command for enabling the protection is:
`stm32f1x lock 0`
When device is read protected (locked) debugging and programming is not possible any more.

The OpenOCD command for disabling the protection is:

```
stm32f1x unlock 0
```

The unlock procedure first erases the flash-memory content and then unlocks the device. **If a device behaves strange during programming, first check if it is locked.**

Hint: enabling of the readout protection can also be done by software (see Standard Peripherals Library functions:

```
FLASH_ReadOutProtection(ENABLE);
FLASH_GetReadOutProtectionStatus();
```

The feature will be enabled at the first boot of the MCU. If handled this way it makes sense to create a special release target in the makefile that includes the code enabling protection, by use of conditional compiling in the release version only, please also see 5.1.2.1 Common Makefile

5.6 Debug

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

At the telnet terminal enter:

```
reset halt
```

The device will stop blinking.

At the terminal window enter:

```
arm-none-eabi-gdb -tui --eval-command="target remote localhost:3333" main.elf
```

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:38
break main.c:47
```

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.

Exit GDB by entering:

```
quit
y
```

5.7 Automate Flash

This section shows how to automate the flash procedure by using a Perl programming script.
Optionally we show how to integrate the flash procedure into the make process.
Install Perl telnet:
sudo apt-get install libnet-telnet-perl
sudo apt-get install libswitch-perl
Create a file in the in the project directory called “jtagprog.pl” and paste the following content into it:

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

print "**********************************\n";
print "* jtagprog for use with OpenOCD *\n";
print "*   unlock/flash/lock-utility   *\n";
print "**********************************\n";

$lock = 0;
$flash = 0;
$unlock = 0;
$help = 0;
$run = 0;
$erase = 0;

if($#ARGV == -1)
{
	# no arguments given
	print "INFO: using defaults - unlock, flash with file 'main.bin', run program\n";
	$unlock = 1;		# default --> do unlock
	$flash = 1; # default --> do flash
	$run = 1;   # default --> run program
}
elsif($#ARGV > 2)
{
	# more than 3 arguments given
	die( "ERROR: execution aborted, more than 3 arguments given!\n\n")
}
else
{
	# arguments given
	switch ($ARGV[0])
	{
		case "-u"	{ $unlock = 1 }
		case "-f"	{ $flash = 1 }
		case "-l"	{ $lock = 1 }
		case "-r"	{ $run = 1 }
		case "-e"	{ $erase = 1 }
		case "-h"	{ $help = 1 }
		else
		{
			print "ERROR: execution aborted, argument 1 not valid!\n\n";
			$help = 1;
		}
	}
	switch ($ARGV[1])
	{
		case "-u"	{ $unlock = 1 }
		case "-f"	{ $flash = 1 }
		case "-l"	{ $lock = 1 }
		case "-r"	{ $run = 1 }
		case "-e"	{ $erase = 1 }
		else
		{
			if($#ARGV >= 1)
			{
				print "ERROR: execution aborted, argument 2 not valid!\n\n";
				$help = 1;
			}
		}
	}
	switch ($ARGV[2])
	{
		case "-u"	{ $unlock = 1 }
```

```
                        case "-f"   { $flash = 1 }
                        case "-l"   { $lock = 1 }
                        case "-r"   { $run = 1 }
                        case "-e"   { $erase = 1 }
                        else
                        {
                                if($#ARGV >= 2)
                                {
                                        print "ERROR: execution aborted, argument 3 not valid!\n\n";
                                        $help = 1;
                                }
                        }
                }
}

if ($erase != 0)
{
        # erase option given, ignore other options
        $unlock = 0;
        $flash = 0;
        $lock = 0;
        $run = 0;
}

if ($help == 1)
{
        print "program usage:\n";
        print "    ./jtagprog.pl [-options ...]\n";
        print "where options include:\n";
        print "    -u       unlock device\n";
        print "    -e       erase device (all other options given will be ignored)\n";
        print "    -f       flash device with file 'main.bin' (including prior erase)\n";
        print "    -l       lock device\n";
        print "    -r       run program (no effect when -l option is given)\n";
        print "    -h       print out this message\n";
        print "examples:\n";
        print "    ./jtagprog.pl -u -f -l\n";
        print "    ./jtagprog.pl -l\n";
        print "    ./jtagprog.pl -u\n";
        print "    ./jtagprog.pl -u -f -r\n";
        print "info:\n";
        print "    - when running program OpenOCD must already be started\n";
        print "    - when -l option is given adjacent power down cycle is mandatory\n";
        print "    - for sole erase of unlocked device use -e option, -u has no effect\n\n";
        exit 0;
}

$filename = './main.bin';
if (($flash == 1) && !(-e $filename))
{
        # flash requested, but file does not exist
        die( "ERROR: execution aborted, file 'main.bin' does not exist!\n\n")
}

$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 $telnet->cmd('flash probe 0');
if ($unlock == 1)
{
        print "INFO: unlock device\n";
        print $telnet->cmd('stm32f1x unlock 0');
        print $telnet->cmd('reset halt');
}
if ($flash == 1)
{
        print "INFO: flash device with file 'main.bin'\n";
        print $telnet->cmd('stm32f1x mass_erase 0');
        print $telnet->cmd('flash write_bank 0 main.bin 0');
```

```
}
if ($erase == 1)
{
		print "INFO: erase device\n";
		print $telnet->cmd('stm32f1x mass_erase 0');
}
if ($lock == 1)
{
		print "INFO: lock device\n";
		print $telnet->cmd('stm32f1x lock 0');
}
print $telnet->cmd('reset halt');
if ($run == 1)
{
		print "INFO: run program\n";
		print $telnet->cmd('reset run');
}
print $telnet->cmd('exit');
print "\n";
```

jtagprog.pl

Close the telnet terminal window. Start the Perl programming script and see how the telnet session starts automatically:
./jtagprog.pl
The Pearl script by default first resets the device, unlocks it, programs the flash-memory with file 'main.bin' and then runs the program.

Optionally modify file “Makefile” in your projects top level directory by adding following content at the end of the file:

```
flash:
		./jtagprog.pl
```

After doing so, an additional make command is available. At the terminal window enter:
make OptLIB=0 OptSRC=0 all tshow flash
Have a look at the OpenOCD terminal window and at the terminal window and see that build and programming script have been executed.

To clean up you project enter following command:
make clean

5.7.1 Production programming

For production purposes efficient programming of the flash device can be done via a terminal window.

If not already running start the OpenOCD server:
xterm -geometry 100x16+100+100 -e "openocd -f openocd.cfg" &
The OpenOCD terminal window should appear, do not enter anything here.

For each device to be programmed do the following steps:

1.) Connect the JTAG device to the board to be programmed and power on the board
2.) Start the Perl programming script:
./jtagprog.pl -u -f -l
The Pearl script first brings the device to be programmed into the unlocked state, then programs the flash-memory with file 'main.bin' and at last locks the device again to protect it against read by untrusted code
3.) When output at terminal window stops, check terminal window if programming and locking succeded
4.) Power down the board and disconnect the JTAG device from the board

After all devices are programmed terminate the OpenOCD server:
pkill openocd

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://git-scm.com/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 115200 -d 8 -f n /dev/ttyACM0; bash' &)
```

run_picocom.sh

To run terminal emulation call script. Adjust picocom parameters to needs if necessary. Device name “ttyACM0” refers to a STM32 VCP (Virtual Com Port) device.
Hint: Beware that permissions for using the ttyXXX device must exist. Get permission for /dev/ttyACM0 permanently by adding yourself to the dialout group. You will have to logout and then log back in before the group change is recognized.

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 and stable.
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 of implementation (january 2012) did 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 Copy Template

Duplicate the content of
“~/22_ARM-Firmware/0002_Test_Template” to
“~/22_ARM-Firmware/0002_Test_Eclipse_Backup”.

7.1.2 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.

One compressed files must be downloaded (filename given may be out of date):

- **Eclipse IDE for C/C++ Developers**, release for linux, version “eclipse-cpp-2020-09-R-linux-gtk-x86_64” or newer. Package is available at: http://www.eclipse.org/downloads/.

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” or add program to the desktop-menu.
3. Start eclipse
4. Eclipse will ask for a working directory. Enter the directory you store your projects in, for example “~/22_ARM-Firmware”.

Second step → install extension “**C/C++ GDB Hardware Debugging**”:

5. Help → Install New Software → Available Software Sites → Set checkboxes at all sites shown → OK
6. Set checkbox “Show only the latest versions...” and disable all other checkboxes
7. Enter “gdb hardware debugging” in field “type filter text”, wait a while ...
8. Set checkbox “C/C++ GDB Hardware Debugging”.
9. Confirm your choices, accept license agreement.
10. Re-start eclipse.

Hint: the eclipse IDE is very comfortable, highly configurable and can be expanded by installation of

plugins. Using the IDE, we had no reason to expand the IDE with external tools, except one thing: the toolbar does not contain undo / redo buttons. These can be installed by copying a jar file (undoredo_1.0.2.jar) to the “dropin” directory inside the eclipse folder.

7.1.3 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 → Debug
4. File → New → Project → C/C++ → Makefile Project with Existing Code → Next
5. At “Project Name” enter “0002_Test_Template”, at “Existing Code Location” click “Browse” and select “0002_Test_Template”, at “Toolchain for Indexer Settings” select “none” → Finish

7.1.4 Configure workspace

Global settings concerning the workspace and all projects.

1. Window → Preferences → General → Workspace → set checkbox “Save automatically before build” → Apply
2. Click on “Startup and Shutdown” → disable checkbox “RSE UI” → Apply → OK
3. 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.

7.1.5 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. In the Project Explorer window make a single click on “0002_Test_Template”
3. Project → Properties → C/C++ Build → Settings → set checkbox “Elf Parser” (no other checkboxes set) → Apply → OK

Ignore second step if GNU toolchain (build and debug suite) was installed via the repository.

Second step (add path to compiler suite)

4. Project → Properties → C/C++ Build → Environment → Add → Name “PATH”, Value “/opt/arm-2012.09/bin ” (for path value see 4.3.2) → set checkbox “Add to all configurations” → OK → OK

Third step (make)

5. Project → Clean → Clean projects selected.. → set checkbox at “0002_Test_Template” → clear checkbox “Start a build immediately” → OK
6. Project → Build All
7. See result of the previous operation inside the “Console” window. A make should have happened.

7.1.6 Configure external tools

Input settings to start OpenOCD and a terminal window 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”.
4. Select “Main” tab, at input area “Location:” click on “Browse File System” and select path to OpenOCD (i.e. “/usr/local/bin/openocd” or “/usr/bin/openocd” when installed via repository).
5. At input area “Working Directory:” click on “Browse Workspace” select “0002_Test_Template” → OK
6. At input area “Arguments:” enter “-f openocd.cfg” (yes, the argument is NOT in quotes).
7. Select “Build” tab → clear checkbox “Build before launch”.
8. Select “Common” tab → at input area “Display in favorites..” set checkbox “External Tools”, at input area “Save as” select “Shared file:” and set to “/0002_Test_Template”.
9. Double click on “Program”, some more input boxes will appear.
10. Set “Name:” to “Terminal emulation”.
11. Goto Tab Main
12. At input area “Location” click on “Browse File System” and select path to shell-script (i.e. “/home/userA/scripts/run_picocom.sh”).
13. Goto Tab Build → clear checkbox “Build before launch”.
14. Goto Tab Common → at input area “Display in favourites..” set checkbox “External Tools” → Apply → Close

7.1.7 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.

7.1.7.1 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”) or the device is reset via JTAG commands (OpenOCD setting “reset_config none”). The device is **reset via hardware signaling or “SYSRESETREQ” interrupt**.

When reset is done via hardware use following settings:

1. Window → Open Perspective → Debug
2. Run → Debug Configurations
3. Double click on “GDB Hardware Debugging”, some more input boxes will appear.
4. Select “Name:” → enter “**Debug**”
5. At “Project”: click on “Browse” → select “0002_Test_Template”
6. At “C/C++ Applikation:” click on “Search Project” → select “main.elf” → Apply
7. At bottom see text “Using GDB … Hardware Debugging Launcher..” click on “Select other...”.
8. Set checkbox “Use configuration specific settings”
9. Select “**Legacy (Standard) GDB Hardware Debugging Launcher**” → OK
10. Set checkbox “Disable auto build”
11. Select the “Debugger” tab → at “GDB Command:
    when GNU toolchain was **installed via the repository** or when **path already set in file “.profile”** (see 4.3.2); :
        input “**arm-none-eabi-gdb**”;
    when GNU toolchain was **not installed via the repository** and when **path not already set in file “.profile”** (see 4.3.2); → Browse and set to full path to e.g. “**/opt/arm-2012.09/bin/arm-none-eabi-gdb**”;

12. set listbox “Command Set:” to “Standard”
13. At “Remote Target” set checkbox “Use remote target”
14. Set “JTAG Device:” to “Generic TCP/IP”, set “Host name..:” to “localhost” and “Port number:” to “3333” → Apply
15. Select the “Startup” tab
16. At “Initialization Commands” clear checkbox “Reset and Delay..:” and clear checkbox “Halt”, in text box enter following line:
    monitor reset init
17. At “Run Commands:” in text box enter following line (optional, not recommended cause it always adds one more breakpoint when starting debug):
    break main → Apply
18. Select “Common” tab → at input area “Display in favorites..” set checkbox “Debug”, at input area “Save as” select “Shared file:” and set to “/0002_Test_Template”.
19. Apply

Note (install step 14.): 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 debug configuration “Debug” by right click on this configuration and selecting “Duplicate”.

a) Select “Name:” → Enter “**Flash+Debug**”
b) Select the “Startup” tab
c) At “Initialization Commands” in text box delete content and enter following **4** lines:
   monitor reset init
   monitor flash probe 0
   No EEPROM emulation: ***monitor stm32f1x mass_erase 0***
   or EEPROM emulation (2 KB) on STM32F103RBT: ***monitor flash erase_sector 0 0 125***
   or EEPROM emulation (4 KB) on STM32F103RET: ***monitor flash erase_sector 0 0 253***
   (Hint: Using “erase_sector” prevents erase of emulated EEPROM memory during flash erase.)
   monitor flash write_bank 0 main.bin 0
d) Apply → Close

Duplicate debug configuration “Flash+Debug” by right click on this configuration and selecting “Duplicate”.

a) Select “Name:” → Enter “**Unlock device**”
b) Select the “Startup” tab
c) At “Initialization Commands” in text box delete content and enter following **4** lines:
   monitor reset halt
   monitor flash probe 0
   monitor stm32f1x unlock 0
   monitor reset halt
d) At section “Load Image and Symbols” remove checkbox at “Load image” and “Load symbols”
e) Apply → Close

7.1.7.2 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 16. and c) in previous section:

16. 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 **4** lines:
   Instead of *monitor reset init* enter line ***monitor soft_reset_halt***

The other 3 lines remain the same as above.

7.1.8 Configure Make Target Window

Handling the make process is controlled by use of a Make Target window.

1. Window → Open Perspective → C/C++
2. Window → Show View → Make Target
3. Inside “Make Target”, single click on the “Hide Empty Folders” icon till “0002_Test_Template” is visible
4. Make a click on “0002_Test_Template”, but do not expand it
5. Make a click on the “New Make Target” icon
6. Disable ckeckbox “Same as the target name”
7. Disable checkbox “Use builder settings”
8. Target name → enter “***Debug (opt:libs+,src-)**”
9. Hint: “Make target” input box stays empty
10. Build command → enter “**make OptLIB=3 OptSRC=0 all tshow**” → OK
11. Make a click on the expand triangle left to “0002_Test_Template”
12. Make a click on the “Hide Empty Folders” icon

13. Make a click on “0002_Test_Template”, but do not expand it
14. Make a click on the “New Make Target” icon
15. Disable ckeckbox “Same as the target name”
16. Disable checkbox “Use builder settings”
17. Target name → enter “**Debug (opt:libs-,src-)**”
18. Hint: “Make target” input box stays empty
19. Build command → enter “**make OptLIB=0 OptSRC=0 all tshow**” → OK

20. Make a click on the “New Make Target” icon
21. Disable ckeckbox “Same as the target name”
22. Disable checkbox “Use builder settings”
23. Target name → enter “**make all --> Debug (full optimize)**”
24. Hint: “Make target” input box stays empty
25. Build command → enter “**Debug (opt:libs+,src+)**” → OK

26. Make a click on the “New Make Target” icon
27. Disable ckeckbox “Same as the target name”
28. Disable checkbox “Use builder settings”
29. Target name → enter “**Release (opt:libs+,src+,ReadOutProtected)**”
30. Hint: “Make target” input box stays empty
31. Build command → enter “**make OptLIB=3 OptSRC=4 all tshow**” → OK

32. Make a click on the “New Make Target” icon
33. Disable ckeckbox “Same as the target name”
34. Disable checkbox “Use builder settings”
35. Target name → enter “**Clean**”
36. Hint: “Make target” input box stays empty
37. Build command → enter “**make clean**” → OK

7.1.9 Code analysis setup

Eclipse implements code analysis during creation of the sources, so errors may be shown before compile time. This is a nice feature but it requires some version dependent setup or throws lots of errors … → do following setup and see the errors fade away or just disable code analysis completely.

Indigo 3.7 32-Bit

1. Window → Preferences → C/C++ → Code Analysis → Restore Defaults → Apply → disable checkboxes at “Symbol is not resolved” and “Type cannot be resolved” → Apply
2. Indexer → Restore Defaults → Apply → OK
3. Project → Properties → C/C++ General → Paths and Symbols → Restore Defaults → Apply → OK

Juno 4.2 32-Bit and Luna 4.4 32-Bit/64-Bit

1. Window → Preferences → C/C++ → Code Analysis → Restore Defaults → Apply
2. Indexer → Restore Defaults → Apply → OK
3. Project → Properties → C/C++ General → Paths and Symbols → Restore Defaults → Apply
4. Preprocessor Include Paths, Macros etc. → Restore Defaults → Yes → Apply → Providers → enable checkboxes of: “CDT User Settings Entries”, “CDT Managed Build Settings Entries” and “CDT GCC Built-in Compiler Settings”, disable checkboxes of all others, sort in following order (top to down): “**CDT User Settings Entries**”, “**CDT Managed Build Settings Entries**”, “**CDT GCC Built-in Compiler Settings**” → Apply → OK
5. Paths and Symbols → Restore Defaults → Apply → Yes → Symbols → GNU C → Add → Name: **uint8_t**, Value: u8, enable checkbox “Add to all configurations” → OK → Add → Name: **uint16_t**, Value: u16, enable checkbox “Add to all configurations” → OK → Add → Name: **uint32_t**, Value: u32, enable checkbox “Add to all configurations” → OK → Apply → Yes → OK

When switching between different versions of eclipse, sharing the same workspace, these settings may be overwritten and lots of errors are thrown again … Solution: re-enter the setup or do not use different versions.

7.1.10 Setup Perspectives

Allocate all components the way you want and the way scree-size allows on the screen. Do this for the C/C++ and the Debug perspective. Do not miss to save the perspectives (and a backup too).

7.1.11 First debug steps

1. Switch to the C/C++ perspective
2. Open file main.c
3. Set breakpoint at line 34 and line 39
4. Make Target window → double click “Clean”
5. Make Target window → double click “*Debug (opt:libs+,src-)”
6. Switch to the Debug perspective
7. Run → External Tools → OpenOCD (or click on icon)
8. Click on arrow near icon Debug (the bug) → select “Flash+Debug”
9. Press F8 → Press F8 → … (stepping through the code, the green LED should toggle)

7.1.12 Eclipse setup files

During setup eclipse made made some hidden additions to the workspace:

1. a directory named ".metadata" located at the root of the workspace
2. an XML document named ".cproject" located at the root of the project directory
3. an XML document named ".project" located at the root of the project directory
4. some XML documents named *.launch located at the root of the project directory, containing GDB hardware debugging and OpenOCD setup

7.1.13 Clone project

Once you have created a functional setup you may want to create other projects without worrying about workspace and project settings. Just use your favorite project as a template, erase all or some oft the contained code and fill it with your new code. Clone the project with some clicks:

1. Switch to the C/C++ perspective
2. Go to the Project Explorer window
3. Right click on the project you want to use as a template e.g. "0002_Test_Template" → Open Project
4. Right click on the project again → Copy
5. Right click on the project once more → Paste → at "Project name:" insert the new projects name (must not contain spaces) e.g. "AbCd" → OK
6. Right click on the project you used as a template → Close Project

The template project now is closed and "AbCd" project is opened. But it won't work, cause the setup files in the "AbCd" project still include the templates project name.

7. Go to the Project Explorer window
8. Make a click on project "AbCd"
9. In the program menue select "Search" → File → File Search
10. At "Containing text:" input the template projects name e.g. "0002_Test_Template" → at "File name patterns.." input "*" → enable checkbox "Case sensitive → at "Scope" select "Workspace" → Replace → With: "AbCd" → Preview → check if automatic selections are fine (only files ".cproject and *.launch of project "AbCd" should be concerned) → OK
11. Right click on the project "AbCd" → Close Project
12. Right click on the project "AbCd" → OpenClose Project
13. Delete backup directory, "~/22_ARM-Firmware/0002_Test_Eclipse_Backup"
14. **Hint:** its about time to make a backup

Modify file "main.c" and include every feature in *your* project – based upon the template project, with everything already in place.

7.1.14 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.

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 Target device type setup

Adapting an MCU type with different memory layout (e.g. STM32F103RBT → STM32F103RET) to the toolchain requires several steps:

1. Uncomment the line specifying the device type in file
   /libs/STM32F10x_StdPeriph_Lib_V3.6.1/Libraries/CMSIS/Device/ST/STM32F10x/Include/stm32f10x.h
   according to the target STM32 device (e.g. MD or HD) used in the application. A high density device for example needs the definition: “#define STM32F10X_HD”.
2. In “Makefile.common” set TypeOfMCU accordingly to step 1.
3. In file “linker.ld” adapt the lines specifying RAM, FLASH and EEMUL to target.
4. In file “openocd.cfg” adapt WORKAREASIZE to target.
5. Configure eclipse debugger to correct chip or page erase behavior, see “7.1.7 Configure debugger”.

9 Bugs and Workarounds

9.1 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)

9.2 IDE-eclipse

9.2.1 Juno release

Juno Service Release 2 (4.2.2) behaves somewhat “sticky” when editing and produces “ticks” via the audio output when entering characters, CR or backspace (?!).
Indigo Service Release 2 (3.7.2) behaves fine and fast and does not show these effects.

9.3 OpenOCD

9.3.1 STM32F103RET

Supply for STM32F103RET was implemented using OpenOCD version 0.5.0. “Flash+Debug” did not work correctly (Target request failed: …) when compiling was done without optimization and code size was > 128KB → Chip erase and download worked but debug did not start. “Debug” worked correctly. Updating to OpenOCD 0.6.1 solved the problem.

9.3.2 Single step failure

On OpenOCD version 0.6.1 when reaching a breakpoint, instantly continuing debugging with single steps on some breakpoints failed. Instead of single stepping a resume till the next breakpoint occurred. When the breakpoint was disabled before single-stepping, single stepping behaved normally. Updating to OpenOCD 0.8.0 solved the problem.

9.4 MCU

9.4.1 I2C peripheral

I2Cx configuration and use is somewhat tricky, cause:

1. I2C interface of STM32F10xxx has some severe bugs and sometimes tends to hang, see errata sheet.
    - Supervision of the I2C interface by software watchdog is indispensable
    - To leave hanging state a reconfiguration of the interface is mandatory
    - I2C analog filter may provide wrong value
    - The errata sheet and the associated sample program itself consists severe errors
2. I2C peripheral device may hang and freeze the SDA line low for infinity (or till power loss)
    - Supervision of the I2C peripheral device by software watchdog is indispensable
    - Toggle the SCL line in GPIO mode during configuration to unfreeze the SDA line is mandatory
3. Almost the whole I2C functionality and it's configuration must run in interrupt mode in order to not slow down the system by use of wait or while() loops.

Proper functionality of the I2C interface can be achieved by implementing a state machine for I2C configuration and watchdogs monitoring the functionality.

10 To do's

- Make files should be improved. Changes in header files do not lead to an automatic re-compilation of the c-files including the headers.
- Keep this document up to date

11 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

12 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
2013-04-06	0.8.3	Implemented STM32F103RET. Update to OpenOCD 0.6.1 and Eclipse 4.2
2013-04-15	0.8.4	Update build toolchain and links.
2013-xx-xx	0.8.5	Starting GDB via terminal corrected.
2014-05-31	0.8.6	Added Automate Flash content, corrected bugs. Removed 0001_Test_Blink example.
2014-06-09	0.8.7	Added conditional compiling to makefiles
2014-06-25	0.8.8	Update to OpenOCD 0.8.0
2014-09-17	0.8.9	Changes in OpenOCD, GNU and eclipse toolchain, LinuxMint support
2014-09-21	1.0.0	Eclipse setup, external tools configuration, streamlining
2014-09-24	1.0.1	Changed eclipse project properties
2014-10-21	1.0.2	Minor bug fixing
2015-06-16	1.0.3	Bug fixing in chapter 7.1.7.1
2020-10-28	1.0.4	Linux Mint 20 compatibility issues implemented

13 Appendix

13.1 Cortex-M3

Collection of Cortex-M3 related documents.

13.1.1 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/st-web-ui/static/active/en/resource/technical/document/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

13.1.2 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

13.1.3 MCU

Datasheet STM32F103RB - DS5319
http://www.st.com/st-web-ui/static/active/en/resource/technical/document/datasheet/CD00161566.pdf

Errata sheet STM32F103x8/B medium-density device limitations - ES096
http://www.st.com/st-web-ui/static/active/en/resource/technical/document/errata_sheet/CD00190234.pdf

Datasheet STM32F103RE - DS5792
http://www.st.com/st-web-ui/static/active/en/resource/technical/document/datasheet/CD00191185.pdf

Errata sheet STM32F103xC/D/E high-density device limitations - ES0104
http://www.st.com/st-web-ui/static/active/en/resource/technical/document/errata_sheet/CD00197763.pdf

Reference manual STM32F103xx advanced ARM-based 32-bit MCU - RM0008
http://www.st.com/st-web-ui/static/active/en/resource/technical/document/reference_manual/CD00171190.pdf

Cortex-M3 programming manual - PM0056
http://www.st.com/st-web-ui/static/active/en/resource/technical/document/programming_manual/CD00228163.pdf

STM32F10xxx Flash memory microcontrollers programming manual - PM0075
http://www.st.com/st-web-ui/static/active/en/resource/technical/document/programming_manual/CD00283419.pdf

AN2594 - EEPROM emulation
http://www.st.com/st-web-ui/static/active/en/resource/technical/document/application_note/CD00165693.pdf
http://www.st.com/st-web-ui/static/active/en/st_prod_software_internet/resource/technical/software/firmware/stsw-stm32010.zip

AN2824 – I2C optimized examples
http://www.st.com/st-web-ui/static/active/en/resource/technical/document/application_note/CD00209826.pdf
http://www.st.com/st-web-ui/static/active/en/st_prod_software_internet/resource/technical/software/firmware/stsw-stm32020.zip

UM0424 - STM32 USB-FS-Device development kit
http://www.st.com/st-web-ui/static/active/en/resource/technical/document/user_manual/CD00158241.pdf?s_searchtype=keyword

STM32 Virtual COM Port Driver – STSW-STM32102
http://www.st.com/web/en/catalog/tools/PF257938
Windows only, not needed when MCU is connected to a GNU-Linux host.

13.2 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

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

STM32LAB
http://elk.informatik.fh-augsburg.de/hhweb/labor/arm/stm32lab

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

System Workbench for STM32 - free IDE on Windows and Linux
The System Workbench toolchain, called SW4STM32, is a free multi-OS software development environment based on Eclipse, which supports the full range of STM32 microcontrollers and associated boards.

The SW4STM32 toolchain may be obtained from the website www.openstm32.org, which includes forums, blogs, and trainings for technical support. Once registered to this site, users will get installation instructions at the Documentation > System Workbench page to proceed with the download of the free toolchain.

The System Workbench toolchain and its collaborative website have been built by AC6, a service company providing training and consultancy on embedded systems.

http://www.openstm32.org/HomePage