GCC ARM 7.3 + GDB ARM 8.1

Vast set of ARM processors can be easily programmed with usage of GCC ARM tool, cross compiled for CM0+ microarchitecture. This tutorial is intended for Cortex M4F/armv7-m (with float co-processor). Optimal hardware requirement is Nucleo-F411RE board, with provided programmer.

1) compile from sources some compiler for CM0+/CM4F-based MCU's. It will be based on provided terminal wizard thanks to "iwasz" website. Please do note, that you can set your own tool name instead default "arm-none-eabi":

```
#sudo aptitude install -y build-essential flex bison libgmp3-dev \
libncurses5-dev libmpc-dev autoconf texinfo libtool libftdi-dev libusb-1.0-0-dev \
zlib1g zlib1g-dev python-yaml openocd ncurses-dev build-essential git \
libgmp-dev libmpfr-dev libmpc-dev zlib1g-dev p7zip-full lxterminal srecord wmctrl
#cd stm/ && sudo cp stm32f4x_rstInit.cfg /usr/share/openocd/scripts/target/
```

compile native mcu compiler, as described in 34_bareMetalCompilers

```
#cd ../34_bareMetalCompilers
#cat build_nativeCompiler.sh
```

2) plug in stm32F411 board and run provided trivial blink example:

```
#./00_empty/RUN_COMMANDS.sh
```

3) consider learning processor capabilities with C programming language (folder 00_doc).

4) C standard libs should be used with care, and on-going program benchmarks – for example: trivially implementable standard sprintf() uses 16kiB of Flash. As for example some complex float functions could be easily aproximated by polynomials. Moreover consider usage of precomputed Look Up Tables in program,

5) source file is always conditionally preprocessed via temporary file for GDB arguments passing, then program is compiled in one of two possible ways:

-for normal memory erase & programming,

-or for GDB debugging, running on OPENOCD ST-LINKv2 client debugger (__BKPT, and __GDB(option1; option2; …; optionN); source file macros). GDB TUI broken by stdout display can be fixed with file ~/.gdbinit (Ctrl+L):

```
define hook-next
   refresh
end
```