Developing for *LimiFrog* under Linux using gcc and Makefiles

Rel. 0.2 – March 2016

A suitable gcc compiler is the *gcc-arm-none-eabi* suite. It is available on Launchpad, here: *https://launchpad.net/gcc-arm-embedded/+download*
You have the option to use Linux, Windows or OS X pre-built binaries.

To flash the STM32, one option under Linux is to use the ST-Link utility developed by 'Texane' available through GitHub : *https://github.com/texane/stlink*.
Or, under Windows, once you have an .hex or a .bin executable you can program the STM32 using ST's ST-Link utility. With it you can download your code into the STM32 program memory – and do a number of things like viewing its contents, comparing it against a bin reference file etc. It is freely available from ST under reference STSW-LINK004, here : *www.st.com/web/en/catalog/tools/PF258168*
The Makefile provided inside each *LimiFrog* sample project invokes the arm-none-eabi toolchain (to build) and Texane's ST-Link utility under Linux (to run). The Makefile is written so that it can be used mostly as is for any project that follows the same file organization as the sample projects. If you want to add some libraries, or want to change compilation options, etc., there should be just a few lines to modify. At the time of writing, the proposed Makefile is used in conjunction with gcc 4.8 for ARM (specifically: *gcc-arm-none-eabi-4_8-2014q2*), under Linux.

If you wish to replicate this environment, here is a « journal » of the steps I followed to install under Linux (Mint) the toolchain that was used to develop the LimiFrog sample projects.

INSTALLING THE GCC COMPILER FOR ARM:

- downloaded the gcc-arm-none-eabi suite from Lanchpad, here:
  https://launchpad.net/gcc-arm-embedded/+download
  (pre-built binaries for Linux, Windows and OS X are available).

- chose to create a directory /ARM under the root on my PC and install there
  gcc-arm-none-eabi-<version number>

- therefore added /ARM to my search paths -- specifying that in ~/.bashrc in my case:
```
# For ARM toolchain
export PATH=$PATH:/ARM/gcc-arm-none-eabi-xxx/bin/
```

- Being on 64-bit Linux, for compatibility with a 32-bit executable, installed ia32_libs :
```
> sudo apt-get install ia32-lib
```

- At this stage, was able to compile a test program.
  For example, run "make build" within one of the sample project folders provided with LimiFrog -- e.g., .../projects/LimiFrog-BringUp0

INSTALLING THE STM32 PROGRAMMING SOFTWARE:

Using work done by 'Texane' available through GitHub.

- downloaded stlink_master.zip from *https://github.com/texane/stlink*
- unzipped it and installed it under /ARM too

There is an automatic build procedure using Autotools. For it to work as expected, had to do the following :
- installed autoconf and automake

```
> sudo apt-get install autoconf
> sudo apt-get install automake
```

- modified permissions of directory stlink_master and its contents (...that was rather sloppy work using 777 but you can be more subtle!)

```
> sudo chmod -R 777 stlink_master
```

- also installed the following :

```
> sudo apt-get install libc6-dev
> sudo apt-get install pkg-config
> sudo apt-get install libusb-1.0-0.dev
```

Then , could successfully conclude the build as directed on the README of Texane's GitHub page (github.com/texane/stlink) :

```
> ./autogen.sh
> ./configure
> make
```

Finally, to be able to actually program the STM32 :

- added the following search path into my ~/.bashrc :

```
# For ST-Link software
export PATH=$PATH:/ARM/stlink-master
```

and, as suggested on the GitHub page :

- installed st-util :

```
> make ./st-util
```

from directory /ARM/stlink-master
- copied file 49-stlinkv*.rules, provided in the stlink-master repository, to /etc/udev/rules.d :

```
> sudo cp 49-stlinkv2.rules /etc/udev/rules.d
```

- and ran :

```
> sudo udevadm control --reload-rules
> sudo udevadm trigger
```

At this point, was able to load through the STLink-V2 programmer a test program
For example, execute "make run" within one of the sample project folders provided with LimiFrog -- e.g., .../projects/0_LimiFog_UnitTest_LED

OPTIONAL **- USING GDB:**

GDB is the GNU debugger utility, which allows to set breakpoints, step into the code, etc.
Do not be afraid if you do not know GDB, you can write code and program LimiFrog without launching GDB.
Its use can however be handy to chase bugs etc., even though it is in part a matter of personal preferences. You can always learn it in due time.

Important Note*:* to be able to properly debug it is recommended to turn off compile optimizations (OPT = 0 in the Makefile, ie. Set compile flag -O0) and to include flag "-g "in compile flags (CFLAGS list in Makefile)

With the board connected to the host PC through the ST-Link/V2 probe, did the following:

- from project file, for example LimiFrog-SW/projects/0_LimiFog_UnitTest_LED, launched st-util:

```
> st-util
```

(Note: typing `st-util --help` would provide information on possible options)

Got the following response:

```
(...)
 INFO gdbserver/gdb-server.c: Listening at *:4242...
```

- Opened a new terminal window and from project file typed :

```
> arm-none-eabi-gdb obj/limifrog.elf
```

Got the following response:

```
(...)
 Reading symbols from
 (...)/projects/0_LimiFog_UnitTest_LED/obj/limifrog.elf...done
```

- Then typed :

```
> target extended-remote :4242
```

Got the following response:

```
Remote debugging using :4242
0x080001dc in Reset_Handler ()
```

- Followed by :

```
> load
```

Got the following response:

```
Loading section ( ... )
( ... )
 Start address 0x80001dc, load size 38520
 Transfer rate: 17 KB/sec, 4815 bytes/write.
```

- Then could use gdb commands (a simple web search will provide the full list).

(*Tip: if you 'run' or 'continue' a program that has no breakpoint set, you don't get the gdb prompt. Type CTRL-C to stop the program and get the prompt back).*