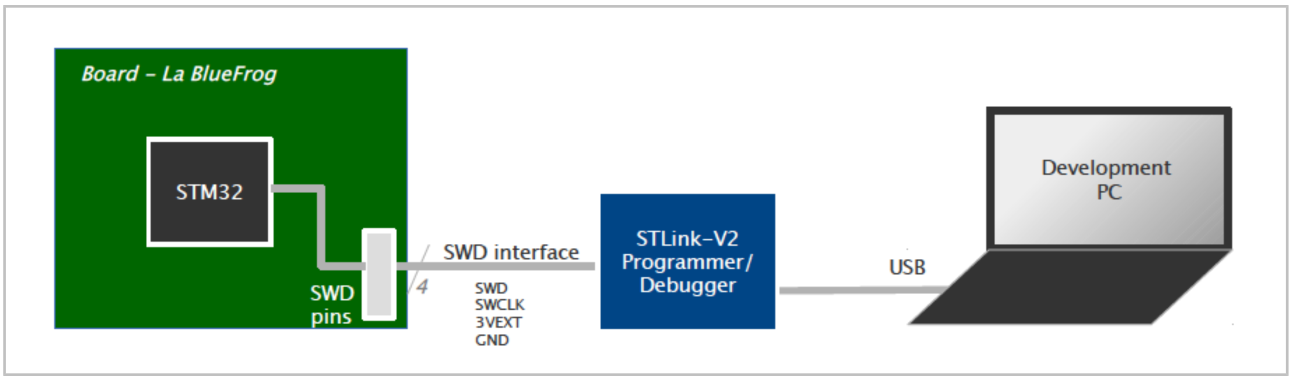
DEVELOPMENT CHAIN
FOR *LA BLUEFROG*

Hardware set-up

To program the STM32 present on *La BlueFrog*, you need a development PC running a suitable toolchain and a programmer/debugger hardware module (that ST calls STLink-V2) - see figure below. This module connects on one side to the development PC via USB and on the other side to *La BlueFrog* via an SWD interface. Everything except the development PC (really?) is provided when you acquire *La BlueFrog*.
The SWD interface is used as a standard for ARM core programming and debug ; it consists of a serial data line (SWD) and a clock line (SWCLK), plus ground (GND) and power supply (3VEXT).



Development toolchains

Option A - Command Line Based Development with Makefiles

A straightforward approach is to just go with a gcc compiler for ARM cores and a loader for the STM32L1. This is the route we have taken to develop the sample projects provided with *La BlueFrog.*

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 La BlueFrog sample project invoke 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 (refer to readme file in provided STM32-Projects-xxx package). If you're departing a little bit from this structure (say, adding some libraries), or want to change compilation options, etc., there should be just a few lines to modify in the Makefile.

If you wish to replicate this environment, an exact list of the steps that were required on our side to do this is provided in the Appendix at the end of this document.

Option B - Using an Integrated Development Environment (IDE)

Commercial IDEs

There are a number of commercial Integrated Development Enviroment (IDE) solutions available to compile C code for the ARM Cortex-M3 and program/debug the STM32L1.
There are, from examples, the tools from IAR, Keil, Raisonance, Hitex, Atollic (Eclipse-based), MikroElektronica. They run under Windows or, for some, Mac OS X too.
Most of them can be obtained for free in a limited version (typically, with code size capped to 32KB).

Open / Free IDEs

Coocox is a free IDE that runs under Windows, the user interface is Eclipse.

emIDE is open-source, the UI is not based on Eclipse. It runs on Windows, their web site says « Linux and Mac version is planned for some time in the future » and « since the code is based on Code::Blocks and wxWidgets, it is mostly easily portable to Mac and Linux Systems ».

There is also an initiative called OpenSTM32 which looks interesting. It will provide a free System Workbench for STM32. Have a look at *test.openstm32.org*.

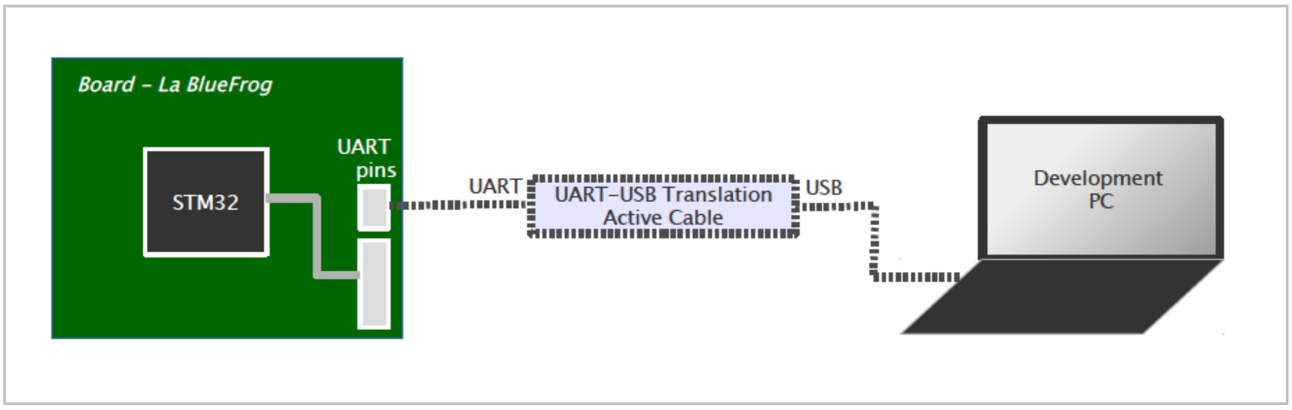
OpenOCD is an Open On-Chip Debugger, not a full IDE itself.
Some have combined Eclipse, gcc, OpenOCD and FreeRTOS to build a cross-platform IDE for Windows, Linux and Mac. If you are into that kind of approach, you may want to check the ODeV project : *www.stf12.org/developers/ODeV.html*

Note:
An interesting source of information on the STM32 is *www.emcu.it* .
They have a page that lists developments tools for STM32 with links to some tutorials :
http://www.emcu.it/STM32/What_should_I_use_to_develop_on_STM32/What_should_I_use_to_develop_on_STM32.html

Optional – UART as debug and information port

La BlueFrog provides a UART port that can be used for different purposes, one of them being debug and diagnosis.
A typical set-up in that case is depicted below. A UART-to-USB cable (such as FTDI's or any similar solution – not provided with *La BlueFrog*) enables seeing the UART interface from PC as a virtual COM port (virtual serial interface over USB) and exchangeing messages between La BlueFrog and the PC.

Then, a terminal emulator (for example , PuTTY) running on the host PC can be used to display data received over the serial interface. The libraries provided with La BlueFrog include a function that sets up the UART for that job, as well as a string handler function for user console interface which implements functions such as xprintf, xgets or put-dump to print a formatted string, acquire a string or dump binary data, etc. Refer to the sample projects for a simple use example.



APPENDIX – Installation of a toolchain under Linux

The following is a « journal » of the steps I followed to install under Linux (Mint) the toolchain that was used to develop the BlueFrog 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-4_8-2014q2/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 La BlueFrog -- e.g., .../projects/BlueFrogV1-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 La BlueFrog -- e.g., .../projects/BlueFrogV1-BringUp0