Using LimiFrog's software package with the STM32 Workbench Eclipse-based IDE

V0.3

Keywords : STM32, IDE, Makefile, OpenOCD, gcc, Windows

INTRODUCTION

The software package provided with LimiFrog includes Makefiles which can be launched from a terminal window using GNU tools (gcc etc.). However, it is also possible to use the Makefiles with gcc-based IDEs (Integrated Development Environments).

This document explains how to compile and debug LimiFrog Makefile-based projects on the *STM32 System Workbench*, which is a free Eclipse-based IDE relying on gcc for compilation and OpenOCD for debug.

Some of the information contained here may also be useful to people willing to work with similar, although not identical configurations (for example with a different Eclipse-based IDE or with OpenOCD outside the proposed IDE).

The procedure described here is an example. There are other solutions ; also, obviously, you may prefer to use file names and paths different from those suggested here.

Note : this is a 'recipe' that worked for the author, intended to help others, it is clearly not « the definitive guide » to using Eclipse. If you are an Eclipse expert and would like to propose an optimal solution, please get in touch.

A- Initialization and Set-Up

1. Obtain the IDE : « System Workbench for STM32 »

This IDE has been developed by AC6 in partnership with ST. It is based on the Eclipse framework. It can be freely downloaded from the OpenSTM32.org website, here :

http://www.openstm32.org/System+Workbench+for+STM32

Java must be installed on your PC.

Both a Microsoft Windows and a Linux version of this workbench are available.

2. Copy LimiFrog software package to your PC

You may download the full package from GitHub.

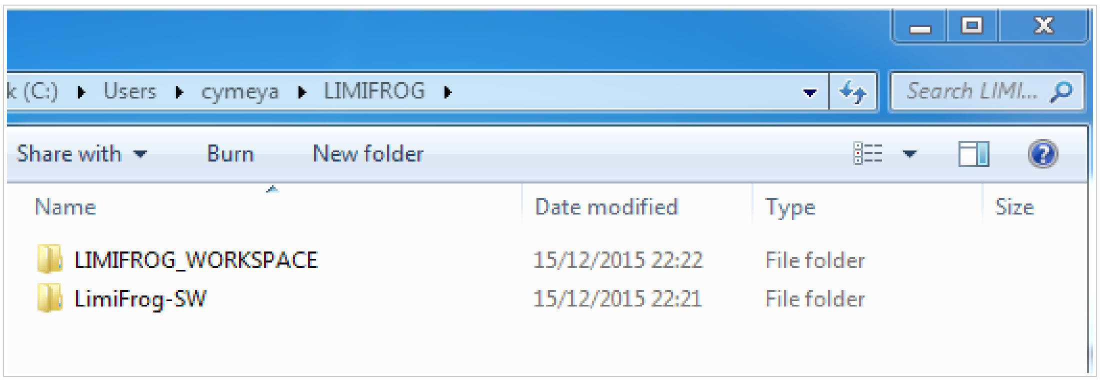
Create a directory /LIMIFROG at a suitable location, for example in the home directory. In this directory we're putting /LimiFrog-SW which is the full software package.

3. Set up Eclipse

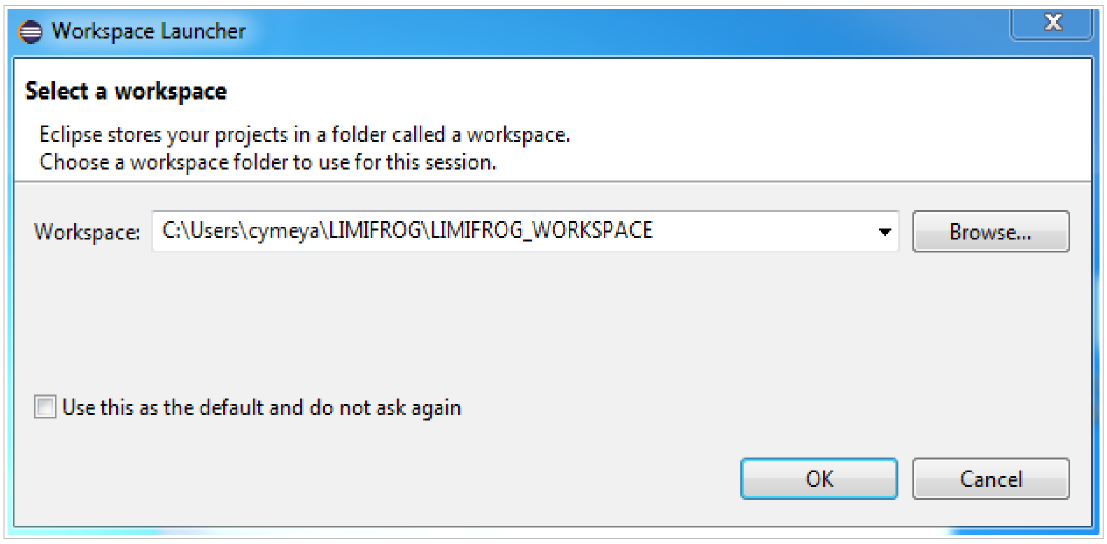
> Launch the STM32 System Workbench.

At first launch after installation, Eclipse asks where to create a workspace. A workspace is a set of consistent projects : for example, in our case, all LimiFrog-related developments

Here we are creating a directory LIMIFROG_WORKSPACE in the /LIMIFROG directory, alongside the /LimiFrog-SW software package and we are selecting this as Eclipse workspace.



Name	Date modified	Type	Size
LIMIFROG_WORKSPACE	15/12/2015 22:22	File folder	
LimiFrog-SW	15/12/2015 22:21	File folder	


Workspace Launcher

Select a workspace

Eclipse stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: C:\Users\cymeya\LIMIFROG\LIMIFROG_WORKSPACE

If needed, close the Welcome tab to switch to the C/C++ project screen (C/C++ « perspective » in Eclipse parlance).

B - Creating and running projects under Eclipse that link to the LimiFrog software package

Say we want to be able to build and debug under Eclipse the project named *0_LimiFrog_UnitTest_LED* in the LimiFrog-SW package : following is one way to do this. The same operation would be done for each project we want to manage under Eclipse.

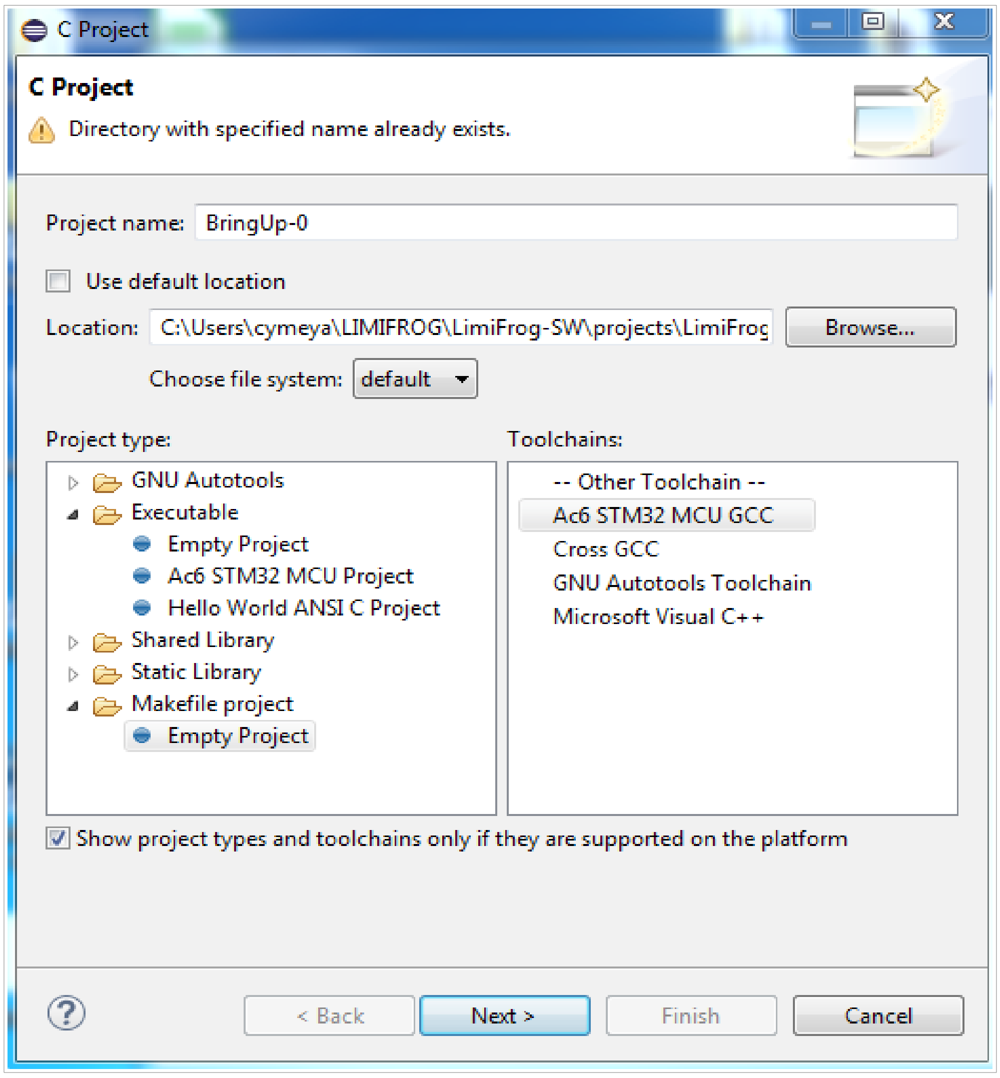
1. Create a new C project under Eclipse :

>> From the top menu bar, select : `File > New > C Project`

A window pops up :

- Untick box '`Use default location`', and (e.g. using the Browse button) select the path to *0_LimiFrog_UnitTest_LED* in the LimiFrog software package.
- Select `Project Type = Makefile project > Empty Project`
- Select `Toolchains = AC6 STM32 MCU GCC`
- Choose Project Name (at top of window) : e.g. `BringUp-0`
  (could also keep the name `0_LimiFrog_UnitTest_LED` or any other)
- Click `Next>`

Here is a snapshot of the pop-up window after having done this :


C Project

C Project

Directory with specified name already exists.

Project name: BringUp-0

[ ] Use default location

Location: C:\Users\cymeya\LIMIFROG\LimiFrog-SW\projects\LimiFrog

Choose file system: default

Project type:
- GNU Autotools
- Executable
  - Empty Project
  - Ac6 STM32 MCU Project
  - Hello World ANSI C Project
- Shared Library
- Static Library
- Makefile project
  - Empty Project

Toolchains:
- -- Other Toolchain --
- Ac6 STM32 MCU GCC
- Cross GCC
- GNU Autotools Toolchain
- Microsoft Visual C++

[✓] Show project types and toolchains only if they are supported on the platform

>> On the next window ('`Select Configurations`') keep default settings and click `Next>`

>> On the next Window ('`MCU Configuration`') :

- if board *LimiFrog* has already been created, select it from the drop-down menu '`Board`'

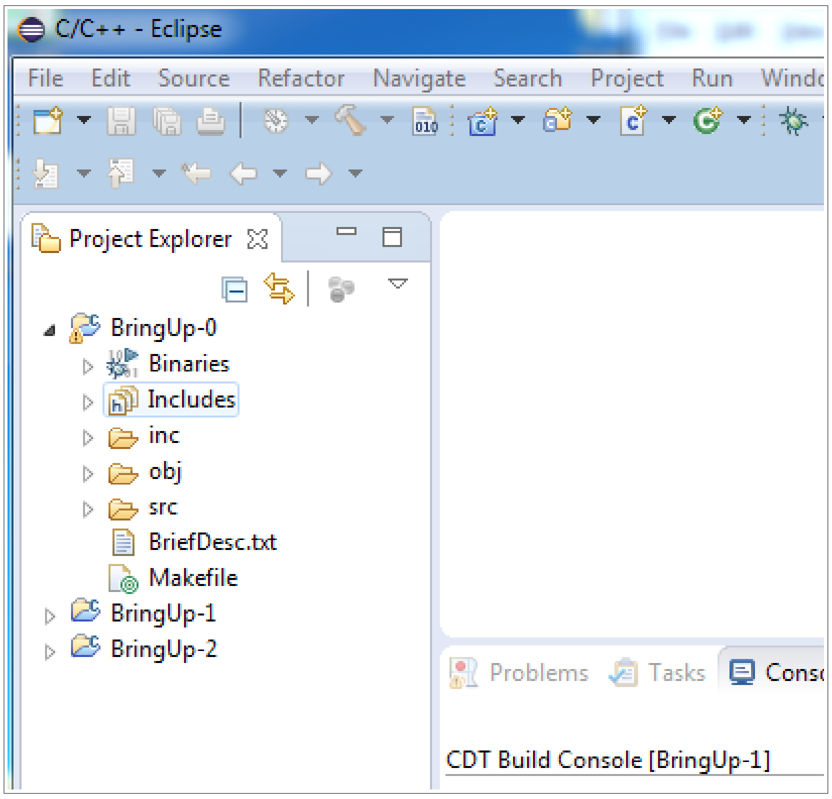
- else, first create it : see Appendix 1

- Click `Finish`

You may get a warning about existing project settings that will be overridden. Click `OK`.

Now project Bring-Up0 is visible in the C/C++ perspective.

You can repeat the operation to import several projects.



2. Building the project(s) under Eclipse

Click on e.g. project BringUp-0 to select it.

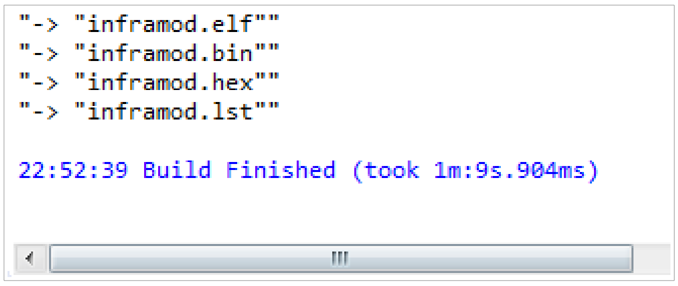
Build the project by :

- either clicking item '`Project`' in the top menu bar and selecting '`Build Project`' from the drop-down menu (sometimes this item is greyed out, if so use next solution – you can also try to first confirm the build configuration by : `Project > Build Configurations > Set active > Default`)

- or, right clicking on project `BringUp-0` on the left and selecting '`Build Project`'.

By doing so the Makefile that is present in project BringUp-0 is invoked.

Results of the build are displayed in the Console tab of the Eclipse screen.



```
"-> "inframod.elf""
"-> "inframod.bin""
"-> "inframod.hex""
"-> "inframod.lst""

22:52:39 Build Finished (took 1m:9s.904ms)
```

If you want to erase previous build resuls to then re-build from scratch you can use the '`Clean Project'` command, similarly to the above « Build Project ».

NOTE :

For the time being, under Windows the Build/Clean operations requires directory **/obj to already exist** prior to launching (even if it is just an empty directory) – else the operation will fail. Some improvements in the Windows-specific portion of the Makefile should allow to fix this issue in the future.

3. Flashing the board and Debugging

This is done through the STLink-V2 programmer/debugger dongle, which connects to a USB port of the PC on one side and to the SWD port of the board on the other side.

The STM32 Workbench IDE relies on OpenOCD (Open On-Chip Debugger software) to exchange information with the STM32's ARM core through the STLink-V2 dongle, using the standardized SWD interface.

To do so, the IDE must be provided with some information about the target board. This information is passed through two specific files. Refer to Appendix 2 for details about these files.

A run/debug session can be conducted as follows :

a) Set up a run/debug configuration

Right -click on the project of interest in the Project Explorer pane and select '`Properties'` from the drop-down menu.

A window opens. No settings for running/debugging the board have been specified yet, so we will create a new « launch » configuration :

- select `Run/Debug Settings` in the left pane,

- click `New...` and select `AC6 STM32 Debugging`

A new window opens with 5 tabs. Go to tab `Debugger`.

The GDB Setup section should read :

```
GDB Command : ${openstm32_compiler_path}\arm-none-eabi-gdb
```

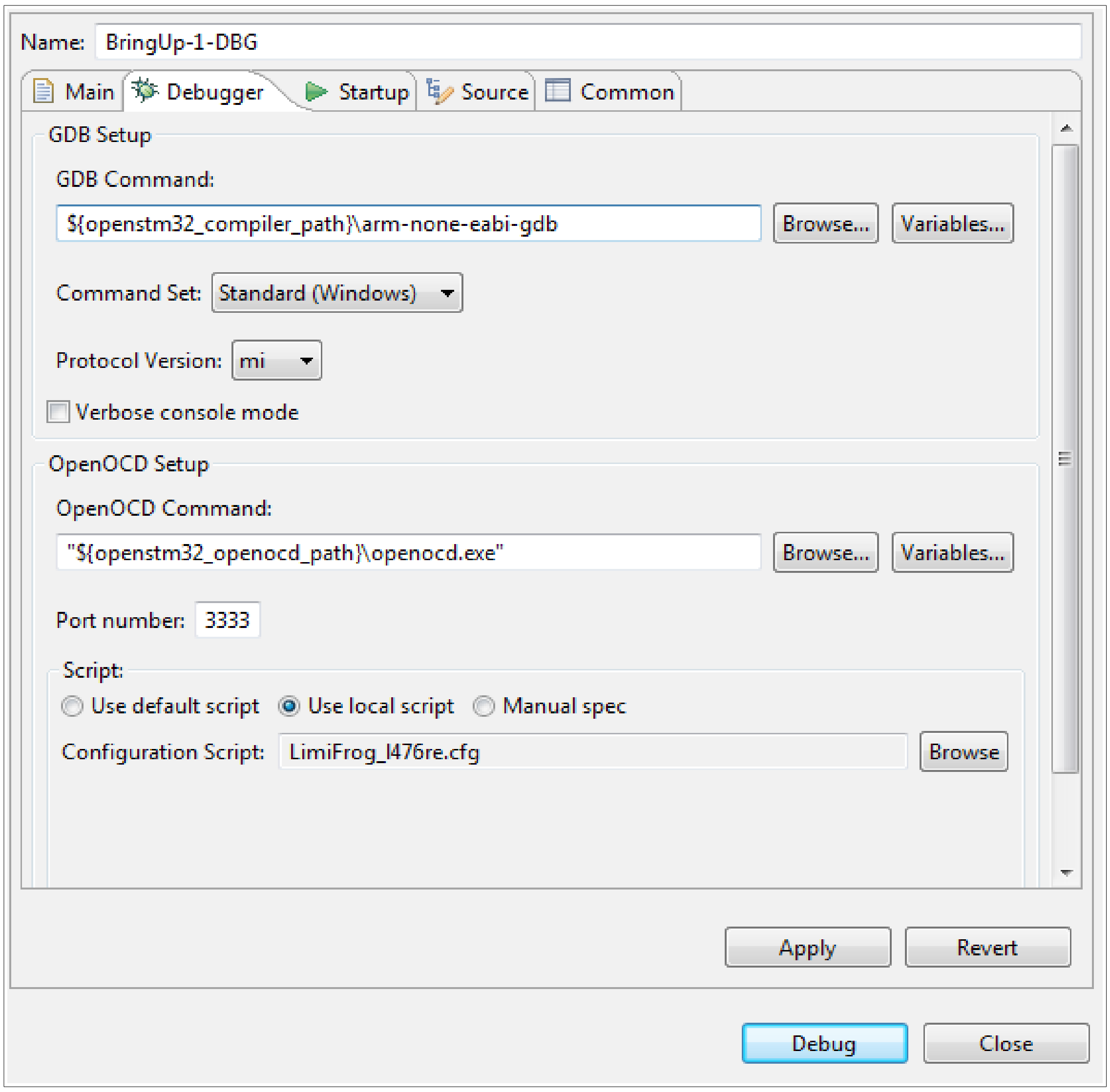
The OpenOCD Setup section should read :

```
"${openstm32_openocd_path}\openocd.exe"
```

Now in the Script section, select option :

`'Use local script'`

and, using the *Browse* feature, specify the path to script *`LimiFrog_l476re.cfg`* (see Appendix 2 for instructions on creating and storing this script the first time).



Name: BringUp-1-DBG

GDB Setup

GDB Command: ${openstm32_compiler_path}\arm-none-eabi-gdb

Command Set: Standard (Windows)

Protocol Version: mi

☐ Verbose console mode

OpenOCD Setup

OpenOCD Command: "${openstm32_openocd_path}\openocd.exe"

Port number: 3333

Script: Use default script [✓ Use local script] Manual spec

Configuration Script: LimiFrog_l476re.cfg

b) (Optional) additional steps for navigating in the code in a debug session

Although the build based on the Makefile is correct, the Eclipse GUI does not show external source code (from the libraries) and reports a lot of 'unresolved symbols', which can be a problem if you want to do some interactive debugging.

[My understanding is that] the 'indexer' of Eclipse, which is supposed to make sense of the full code to ease interactive debugging, does not use the provided Makefile as the compiler does. Therefore it does not know where to find files that are outside the project registered in the Eclipse workspace.

To go round this issue, here's a possibility.

We will make the missing (external) files visible from inside the Eclipse project by creating symbolic links in the project :

First, for all library files used by the LimiFrog software package :

- right-click on project name in project pane,
- select : `New > Folder`

A window opens,

- click on : `Advanced >>`
- select button : `Link to alternate location (Linked Folder)`
- indicate the path to directory */libraries* of the LimiFrog software package
- click : `Finished`

Again for a different path (to the set of files common to all project) :

- right-click on project name in project pane,
- select : `New > Folder`

A window opens,

- click on : `Advanced >>`
- select button : `Link to alternate location (Linked Folder)`
- indicate the path to directory */projects/COMMON* of the LimiFrog software package
- click : `Finished`

Finally, update the indexer by right-clicking on the project and selecting :

- `Index > Rebuild`

This way, the indexer sees (through links) the full code and symbols (although sometimes some of them are still not resolved – Eclipse unstability ??).

NOTE- COMPILE OPTIONS FOR SOURCE CODE VISIBILITY :

Source code will be visible in the debug window only if the compilation was done **with the -g flag set.** Else you get a message « No source available for... »

This can be done e.g. by adding *`-gg`* or *`-ggdb`* to the CFLAG list in the Makefile (list of flags to be used at compilation).

Also, it is preferable to disable compile optimizations to ease debugging (set

OPT=0 in Makefile, i.e. set compile flag -O0).

c) Connect the board

Plug the 4 SWD pins of the ST-Link/V2 device into the SWD port of LimiFrog (4 holes)

d) You can now do various things such as flashing the STM32, running, debugging.

> To download the code and get LimiFrog running :

- enter the Run drop-down menu (at window top) and click : `Run`
- You should see various information in the console window, similar to this :

```
Open On-Chip Debugger 0.9.0-dev-dirty (2015-11-13-11:40)
Licensed under GNU GPL v2
For bug reports, read
      http://openocd.org/doc/doxygen/bugs.html
Info : The selected transport took over low-level target control. The results
might differ compared to plain JTAG/SWD
Warn : use 'STM32L476.cpu' as target identifier, not '0'
adapter_nsrst_delay: 100
none separate
adapter speed: 1800 kHz
Info : clock speed 1800 kHz
Info : STLINK v2 JTAG v24 API v2 SWIM v4 VID 0x0483 PID 0x3748
Info : using stlink api v2
Info : Target voltage: 6.419921
Info : STM32L476.cpu: hardware has 6 breakpoints, 4 watchpoints
adapter speed: 240 kHz
target state: halted
target halted due to debug-request, current mode: Thread
xPSR: 0x01000000 pc: 0x08000318 msp: 0x20018000
adapter speed: 4000 kHz
** Programming Started **
auto erase enabled
Info : STM32L4xx flash size is 1024kb, base address is 0x8000000
target state: halted
target halted due to breakpoint, current mode: Thread
xPSR: 0x21000000 pc: 0x20000068 msp: 0x20018000
wrote 90112 bytes from file obj/limifrog.elf in 2.278596s (38.620 KiB/s)
** Programming Finished **
** Verify Started **
target state: halted
target halted due to breakpoint, current mode: Thread
xPSR: 0x61000000 pc: 0x2000002e msp: 0x20018000
target state: halted
target halted due to breakpoint, current mode: Thread
xPSR: 0x61000000 pc: 0x2000002e msp: 0x20018000
verified 88552 bytes in 2.631242s (32.865 KiB/s)
** Verified OK **
** Resetting Target **
adapter speed: 240 kHz
shutdown command invoked
```

At this point the board is programmed, you can remove the ST-Link connection.

> To enter an interactive debug session :

- Enter the Run drop-down menu (at window top) and click : `Debug`
- You should enter directly or be invited to enter the debug perspective
- unless you changed some defaults, the program is suspended on a breakpoint automatically set on the first instruction of your *main()*.

APPENDIX 1 –

CREATING A CUSTOM « LIMIFROG » BOARD IN THE STM32 WORKBENCH

When creating your first project and reaching the '`MCU Configuration`' window :

- Click '`Create a new custom board`'. A new window pops up.

Select : `Define new board`.

Enter new board name : e.g., `LimiFrog`

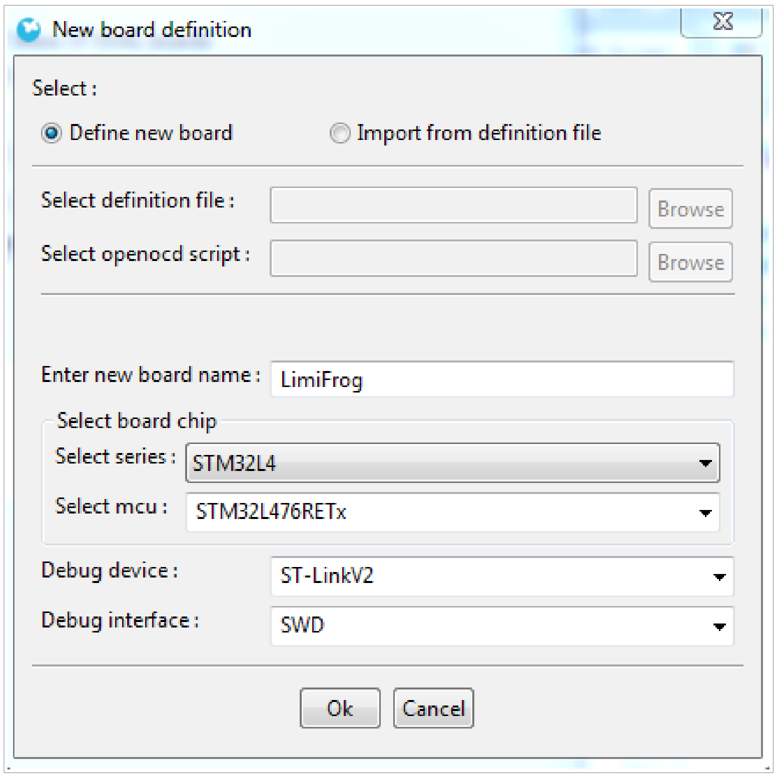
`Select board chip / Select Series : STM32L4`

`Select board chip / Select mcu : STM32L476RETx`

`Debug device :  ST-LinkV2 (or ST-LinkV2-1 if you are using a Nucleo board as programming device)`

`Debug interface : SWD`

`OK`


New board definition

Select :

[✓ Define new board] Import from definition file

Select definition file :

Select openocd script :

Enter new board name : LimiFrog

Select board chip

Select series : STM32L4

Select mcu : STM32L476RETx

Debug device : ST-LinkV2

Debug interface : SWD

Note : in the future LimiFrog could be listed as one of the boards supported 'out of the box' by the STM32 System Workbench and there would then be no need for this step.

APPENDIX 2 –
TARGET SCRIPTS FOR THE DEBUGGER

Two scripts need to be created.

1) One is a board target script which can be put e.g. with pre-defined ST boards in the STM32 Workbench installation.

This script is provided under name *`LimiFrog_l476re.cfg`* in the same directory as this document on GitHub (the name is free, the key is to point to that file as « local script » in the launch configuration).

It can for example be copied into the following location (where scripts for other ST boards are already present) :

`… > Ac6 > SystemWorkbench > plugins > fr.ac6.mcu.debug_`*`[latest_release_number]`*` > ressources > openocd > scripts > st_board`

Its contents are :

```
# This is a LimiFrog board with a single STM32L476RET6 chip.
#

# This is for using the STLINK/V2
source [find interface/stlink-v2.cfg]

# OR, this is for using the STLINK/V2-1 (when using Nucleo as programming
device)
# source [find interface/stlink-v2-1.cfg]

transport select hla_swd

# increase working area to 96KB
set WORKAREASIZE 0x18000

source [find target/LimiFrog_stm32l4.cfg]
```

*** **NOTE** : If you are using a Nucleo board as programming device (which has an ST-Link/V2-1 on-board), you should comment out in the above the line that says '`source [find interface/stlink-v2.cfg]`' and uncomment that which says : '`source [find interface/stlink-v2-1.cfg]`' ***

2) As can be seen, this file includes a « chip target file » *`LimiFrog_stm32l4.cfg`*.

Again, this file is provided in the same directory as this document on GitHub and should now be copied into the following location (where scripts for other ST chip targets are already present) :

`… > Ac6 > SystemWorkbench > plugins >`

fr.ac6.mcu.debug_*[latest_release_number]* > ressources > openocd > scripts > target

(similar to previous path, only last directory is different).

Here are its contents :

```
# script for stm32l4x family

#
# stm32l4 devices support both JTAG and SWD transports.
#
source [find target/swj-dp.tcl]
source [find mem_helper.tcl]

if { [info exists CHIPNAME] } {
   set _CHIPNAME $CHIPNAME
} else {
   set _CHIPNAME stm32l4
}

set _ENDIAN little

# Work-area is a space in RAM used for flash programming
# By default use 64kB
if { [info exists WORKAREASIZE] } {
   set _WORKAREASIZE $WORKAREASIZE
} else {
   set _WORKAREASIZE 0x10000
}


#jtag scan chain
if { [info exists CPUTAPID] } {
   set _CPUTAPID $CPUTAPID
} else {
   if { [using_jtag] } {
      # See STM Document RM0351
      # Section 44.6.3 - corresponds to Cortex-M4 r0p1
      set _CPUTAPID 0x4ba00477
   } else {
      # SWD IDCODE (single drop, arm)
      set _CPUTAPID 0x2ba01477
   }
}

swj_newdap $_CHIPNAME cpu -irlen 4 -ircapture 0x1 -irmask 0xf
-expected-id $_CPUTAPID

if { [info exists BSTAPID] } {
   # FIXME this never gets used to override defaults...
   set _BSTAPID $BSTAPID
} else {
   # See STM Document RM0351 Section 44.6.2
   # Low and medium density
   set _BSTAPID1 0x06415041
}
```

```
if {[using_jtag]} {
   swj_newdap $_CHIPNAME bs -irlen 5 -expected-id $_BSTAPID1
}

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

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

# flash size will be probed
set _FLASHNAME $_CHIPNAME.flash
flash bank $_FLASHNAME stm32l4x 0x08000000 0 0 0 0 $_TARGETNAME

adapter_nsrst_delay 100
if {[using_jtag]} {
 jtag_ntrst_delay 100
}

# use hardware reset, connect under reset
# reset_config srst_only srst_nogate
# LIMIFROG: CHANGED TO
reset_config none


if {![using_hla]} {
   # if srst is not fitted use SYSRESETREQ to
   # perform a soft reset
   cortex_m reset_config sysresetreq
#}

adapter_khz 1800


$_TARGETNAME configure -event reset-start {
     adapter_khz 240
}

$_TARGETNAME configure -event examine-end {
     # Enable debug during low power modes (uses more power)
     # DBGMCU_CR |= DBG_STANDBY | DBG_STOP | DBG_SLEEP
     mmw 0xE0042004 0x00000007 0

     # Stop watchdog counters during halt
     # DBGMCU_APB1_FZ = DBG_IWDG_STOP | DBG_WWDG_STOP
     mww 0xE0042008 0x00001800
}

$_TARGETNAME configure -event trace-config {
     # Set TRACE_IOEN; TRACE_MODE is set to async; when using sync
     # change this value accordingly to configure trace pins
     # assignment
     mmw 0xE0042004 0x00000020 0
}

$_TARGETNAME configure -event reset-init {
     # Configure PLL to boost clock to HSI x 4 (64 MHz)
```

```
      # Set HSION in RCC_CR
      mww 0x40021008 0x00000001   ;# HSI ON RCC_CR

      mww 0x4002100C 0x03020302   ;# RCC_PLLCFGR 16 Mhz /2 (M) * 32 (N) /
4(P) =  64 mhz
#     mww 0x4002100C 0x03028302   ;# RCC_PLLCFGR 16 Mhz /2 (M) * 40 (N) /
4(P) =  80 mhz
      mww 0x40022000 0x00000102   ;# FLASH_ACR = PRFTBE | 2(Latency)
      mmw 0x40021000 0x01000000 0 ;# RCC_CR |= PLLON

      sleep 10                    ;# Wait for PLL to lock
      mmw 0x40021008 0x00001000 0 ;# RCC_CFGR |= RCC_CFGR_PPRE1_DIV2
      mmw 0x40021008 0x00000003 0 ;# RCC_CFGR |= RCC_CFGR_SW_PLL

      # Boost JTAG frequency
      adapter_khz 4000
}
```

As can be seen, the reset configuration is set to : `reset_config none`

This specifies that the optional SRST signal of the SWD interface is not wired on LimiFrog.

APPENDIX 3 –
SYSTEM WORKBENCH AND LINUX

Please pay attention to the warnings provided by AC6 when installing for Linux.

Also, normally during the installation, a file called *99-openocd.rules* is placed under /etc/udev/rules.d

If it happens that these are not present, you may copy file *49-stlinkv2.rules* (provided in the same directory on GitHub as this document) into /etc/udev/rules.d , for example :

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

This is a subset of the above file that focuses on rules useful for the ST-Link/V2 or V2.1 devices, which are those we are interested in.