Using LimiFrog's software package

with the STM32 Workbench Eclipse-based IDE

V0.1

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.

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.

(Incidentally, a Linux version of this workbench is also available).

2. Copy LimiFrog software package to your PC

You may download the full package from GitHub.

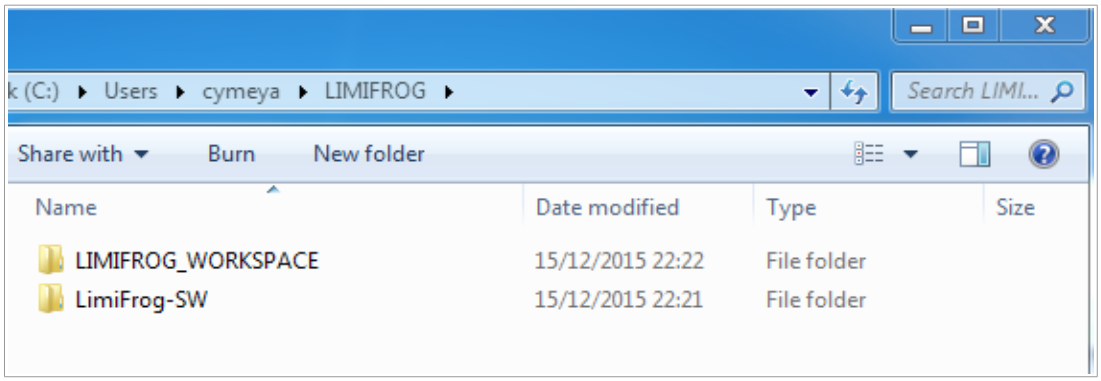
Create a directory /LIMIFROG at a suitable location, for example in the home directory. In this directory we're putting /LimiFrog-SW which is the full software package.

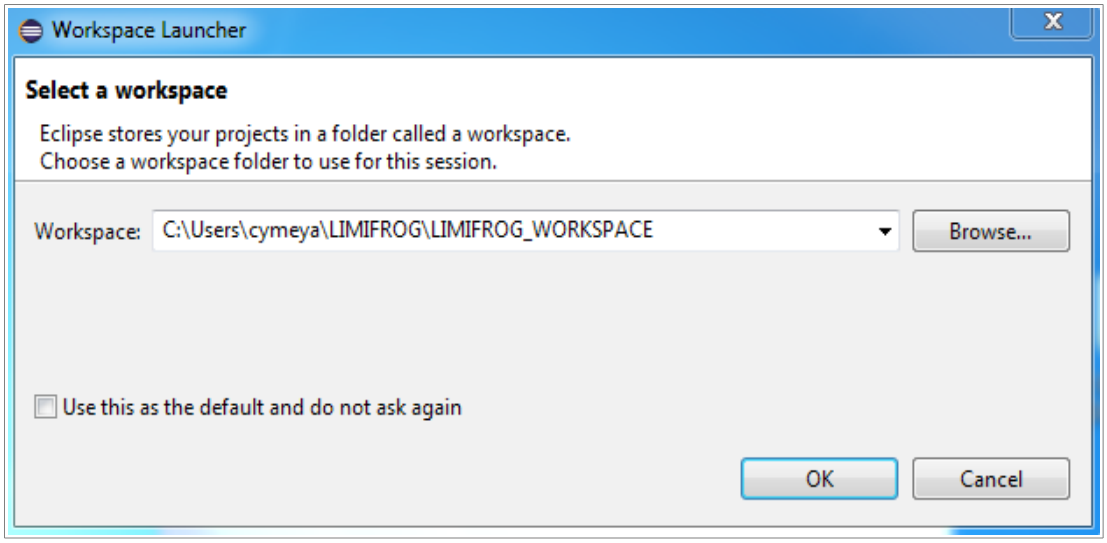
3. Set up Eclipse

> 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.





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 LimiFrog-BringUp0 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 LimiFrog-BringUp0 in the LimiFrog software package.

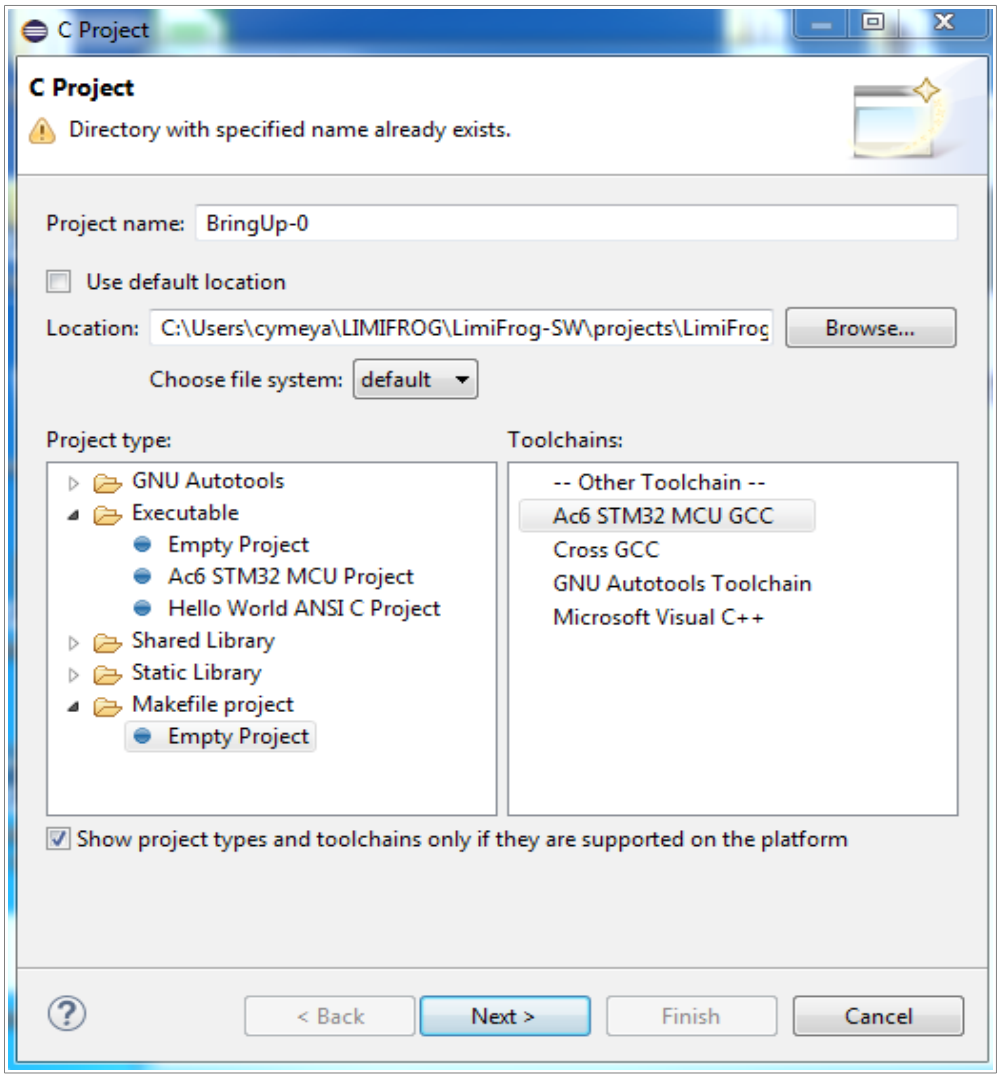
- Select `Project Type = Makefile project > Empty Project`

- Select `Toolchains = AC6 STM32 MCU GCC`

- Choose Project Name (at top of window) : e.g. `BringUp-0`

- Click `Next>`

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



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

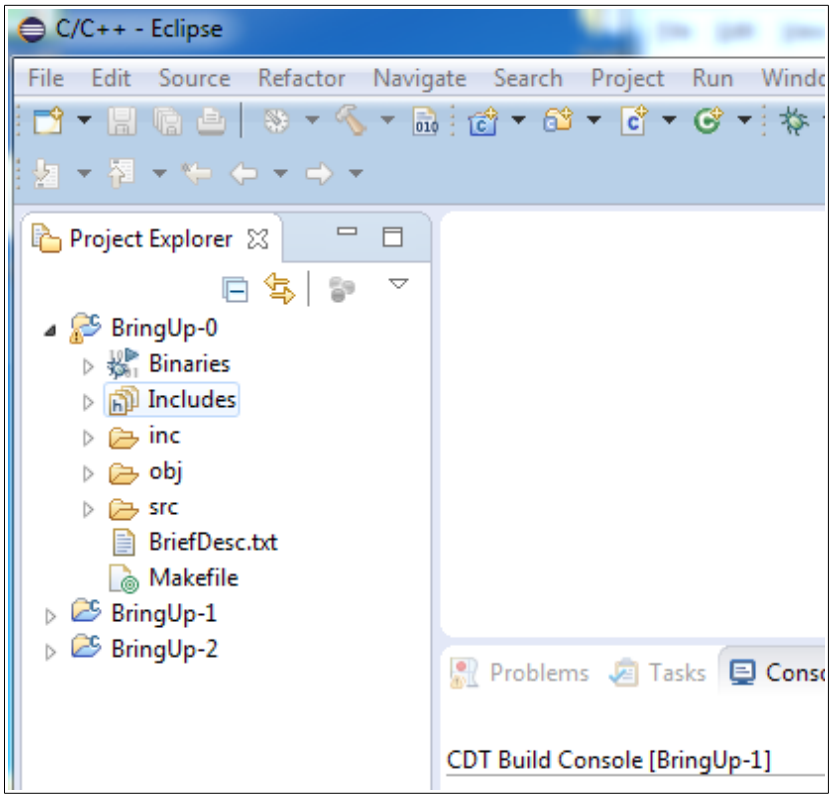
>> On the next Window ('`MCU Configuration`') :

- 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 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 :

- 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
```

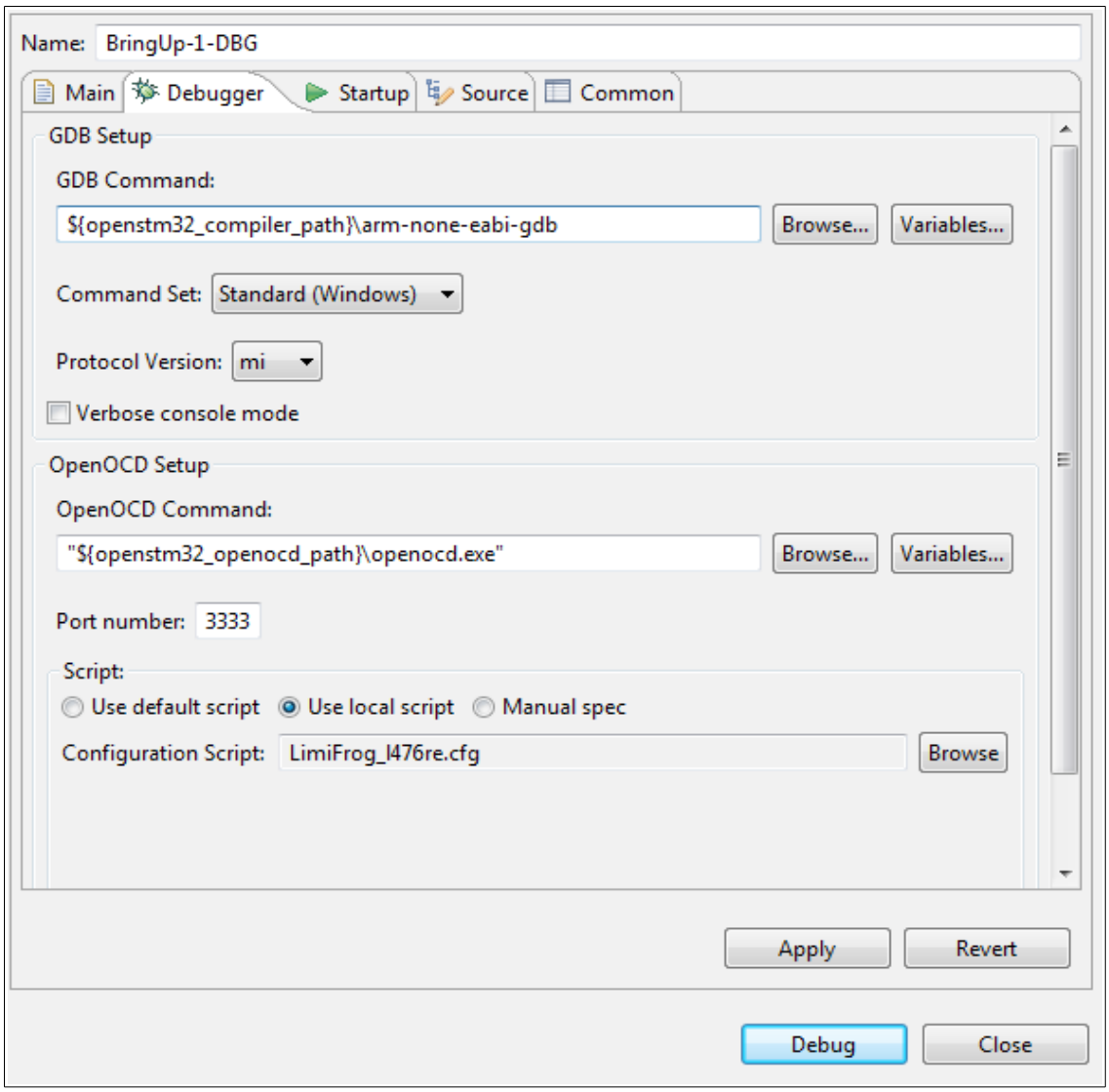
The OpenOCD Setup section should read :

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

Now in the Script section, select option :

`'Use local script'`

and (e.g. using the Browse feature) specify the path you have set to reach script *`LimiFrog_l476re.cfg`* when creating it (see Appendix 2).



b) Connect the board

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

NOTE- 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 `-ggdb` to the CFLAG list in the Makefile (list of flags to be used at compilation).

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

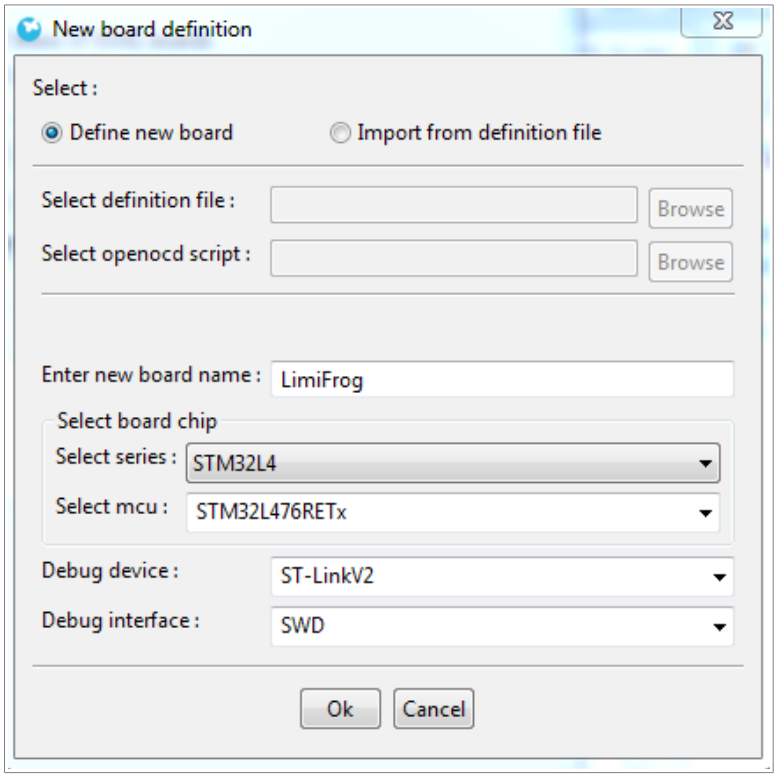
Select board chip / Select Series : STM32L4

Select board chip / Select mcu : STM32L476RETx

Debug device : ST-LinkV2

Debug interface : SWD

OK



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.

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

The path is :

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

where we place target file *LimiFrog_l476re.cfg*

The name is free, the key is to point to that file as « local script » in the launch configuration.

Its contents are :

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

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

transport select hla_swd

# increase working area to 96KB
set WORKAREASIZE 0x18000

#source [find target/stm32l4.cfg]
source [find target/LimiFrog_stm32l4.cfg]
```

As can be seen, this file includes a chip target file *LimiFrog_stm32l4.cfg*, its path is :

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

(similar to previous, 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.