LimiFrog

Known Limitations and Features To Be Aware Of

Rel. 0.1

March, 2106

---

1. Maximum core clock when using USB (72MHz rather than 80MHz)

2. DFU Mode (Code download over USB)

3. Microphone output

4. Temperature sensors

1) Maximum core clock when using the USB (72MHz rather than 80MHz)

I experimented some problems when trying to use at the same time the maximum system clock of 80MHz and the USB peripheral in device mode (possibly related to some STM32 hardware limitation but this has not been confirmed).
Because of that, **the software package provided sets the system clock to 72MHz** rather than 80MHz, which allows to solve the problem. This is done in source file
`LimiFrog-SW/libraries/src/LBF_Board_Inits/ LBF_SysClkCfg.c`.
A variant that sets the system clock to the maximum 80MHz is available in the same directory under the name `LBF_SysClkCfg.c.80MHz`. However, when using 80MHz as system clock, the STM32 clock generator has to be set in a different way to produce the 48MHZ USB clock and this is the situation where the issue has been observed (USB functionality in device mode was faulty on 50% of the parts, OK on the rest).

Some details :
The clock generator inside the STM32, described in Figure 13 of the STM32L4 User Manual, generates all internal clocks using 3 PLL (Phase Locked Loops).
- One of these clocks is the system clock that goes to the Cortex core, its maximum frequency is 80MHz.
- Another one is the clock that goes to the USB peripheral, its frequency must be 48MHz.

When setting the system clock to 80MHz, the division ratios available from the first PLL mean it is not possible to obtain 48MHz from that one. The second PLL (PLLSAI1) must be used. However, on 10 samples I tested, generating 48MHz with the second PLL (from an 8MHz external source) did not allow the USB to work reliably: some samples were OK, others were not – apparently, because some configuration registers could not be written properly.
The highest system clock that can be used to still be able to generate 48MHz from the first PLL is 72MHz. When generating the 48MHZ USB clock from this PLL, the issue disappears and the USB works reliably.
I have done this tests in USB Device Mass Storage Mode and have not tested if the problem also exists for other USB classes.

2) DFU Mode (Code download over USB)

DFU (Device Firmware Update) mode is a special mode where, upon reset, the STM32 tries to download code from one of its serial interface rather than normally executing the code placed in system (Flash) memory.

The normal behavior is as follows :
- if pin BOOT0 of the STM32 is pulled high, the normal boot sequence executes,
- if pin BOOT0 is pulled low, the STM32 jumps to a bootloader routine stored in ROM. This routine examines various serial interfaces (I2C, SPI, UART, USB...) in turn and if it detects some activity it tries to obtain code through this interface using a specific protocol.

On LimiFrog, the behavior is slightly different :
- the role of pin BOOT0 is played by a user-selectable GPIO available on the extension port
- this GPIO is selected in a specific routine part of the provided software package – by default this is STM32 pin PB8, available on extension port position 10, but you could change this by editing the code.
- **it is essential to keep this routine (or something equivalent) the first piece of code executed after start-up, to retain DFU mode capability.**

The routine is called: `void LBF_DFU_If_Needed(void)`
and is found in:
`LimiFrog-SW/libraries/LimiFrog-Libs/src/LBF_Board_Inits/LBF_DFU_If_Needed.c`

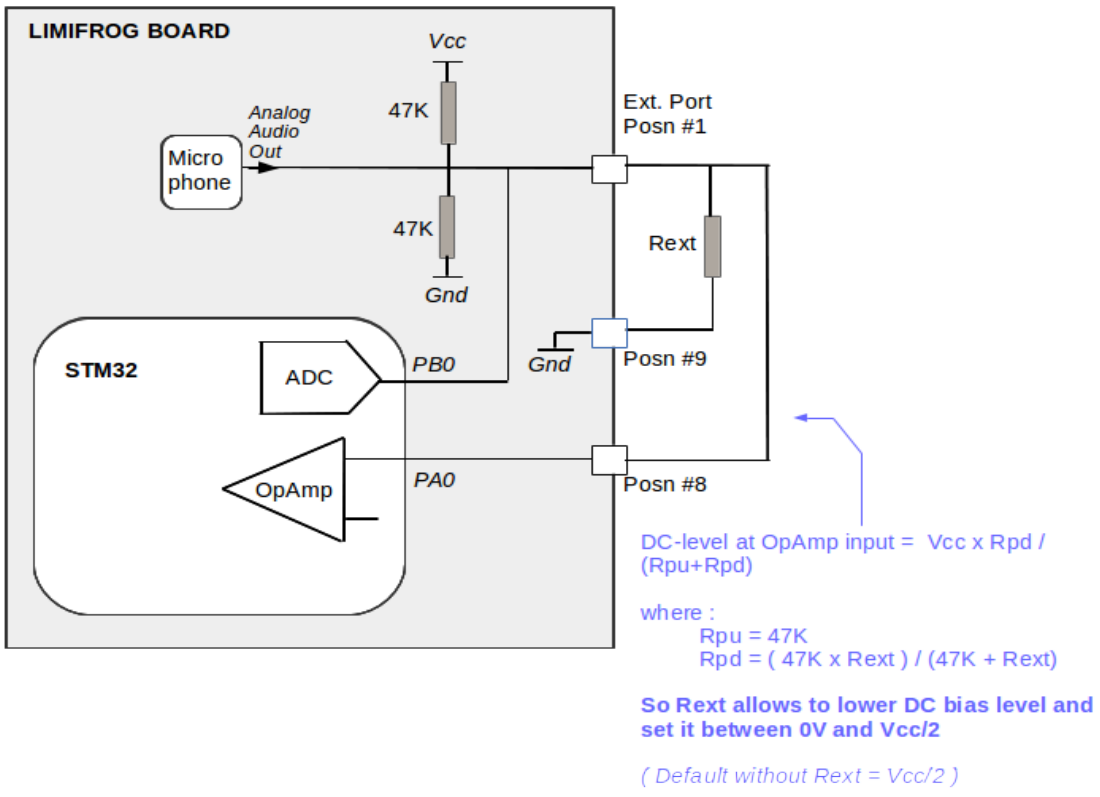
See the source code for more information on the root cause and work-around.

Other related documentation:
`LimiFrog-SW/documentation/LimiFrog/Dev_Environment/Flashing_LimiFrog_over_USB.pdf`

3) Microphone output

The MEMS microphone implemented on LimiFrog can capture noise events (such as clapping hands, tapping the case or whistling close to the box), but has not been optimized for quality audio.
It produces a fairly weak signal that may need amplification, depending on usage scenario.
The microphone analog output is DC-biased around VCC/2 (~ 1.47V) and is routed directly to an analog input of the STM32, as well as to position 0 of the extension port.
When no amplification is required, the on-board direct path can be used. The pin to which the audio signal is directly routed is an input to an internal Analog-to-Digital Converter.
When amplification is required, one solution is to use the Operational Amplifier of the STM32. The positive input of the OpAmp (pin PA0 of the STM32) is available on position 8 of the extension port. Therefore an external connection from position 1 to position 8 of the extension port does the job. However, because the input to the OpAmp is DC-coupled, one must take care of setting a suitable DC-level to avoid saturation at the output of the OpAmp.
The figure below outlines a possible scheme.

Another factor to keep in mind :
The frequency response of the system is not flat. For example a whistling is much better picked than normal voice.



4) Temperature Sensors

Several on-board sensors embed a temperature sensor as they need this information to calibrate themselves. Pay attention to the absolute accuracy (specified in the datasheets), though.

Also, keep in mind the measured temperature reflects the temperature at sensor level and is influenced by its environment. For example, when the board is 'sandwiched' between the battery (which may impede heat evacuation) and the OLED display (which produces some heat when active) and confined in a case, the measured temperature will rise after power-up. Some software calibration may help.
Overall power consumption and therefore heat dissipation of the module is low so this effect is limited, but sufficient to be noticeable. Strategies to limit the temperature rise include choosing an arrangement where the board is not confined, turning the OLED off when possible, etc.