ping-pong bootloader是可以支持A/B两块固件跳转执行的bootloader，同一时刻只有一块固件可以被执行（比如固件A）。当需要固件更新时，新的固件下载到未处于运行状态的存储区（比如固件B），完成后标记为下次可运行，重启后运行该固件。这种更新方式不会打断当前固件的执行，实现无感升级。并且，当下载了错误的固件导致系统无法运行，可通过手动重启，自动回退到之前可运行的固件版本。本文档描述了该bootloader的实现原理和对应代码库的使用方法。

1. 实现原理



图1. Ping-pong Bootloader存储区域划分

上图描述了在STM32F412平台采用的存储区域划分方式。其中：

1. Bootloader存储区：用于存储Ping-pong Bootlader
2. 固件状态存储区：用于存储固件A/固件B当前的下载状态、运行状态等信息
3. 其他存储区：目前用于保存加密的参数（非必须）
4. 固件A/固件B存储区：实际固件下载的区域
5. 未使用存储区：当前划分方式未使用到的区域（非必须）

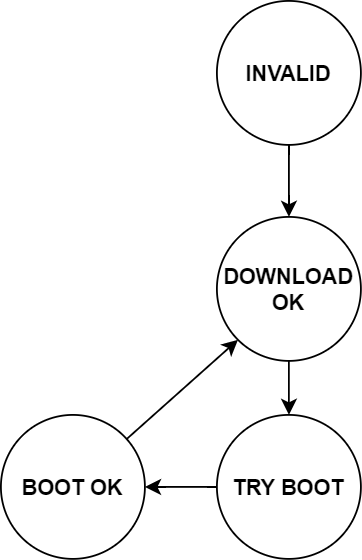


图2. 固件状态转换图

上图描述了一个固件在Ping-pong Bootloader和应用程序共同控制下的状态转换图。

1. INVALID状态：目前只有固件B初次上电默认处于这个状态，表示固件B存储区未下载新的固件
2. DOWNLOAD OK状态：目前固件A初次上电默认处于这个状态（出厂时将固件刷到固件A存储区），其他情况下，完成了指定固件下载，则对应固件状态更新成此状态
3. TRY BOOT状态：Bootloader第一次跳转执行某个新固件之前，将该固件状态更新成此状态，表示尝试启动该固件，但是否启动成功需要应用自己判断
4. BOOT OK状态：固件成功启动后，会在合适的时机更新固件状态为此状态，表示当前固件已成功启动

自动回退到上次可运行固件实现原理：当系统重上电或手动复位，Bootloader引导某个固件时，发现固件状态为TRY BOOT或INVALID，表示该固件无法正常启动（未被应用更新为BOOT OK或没有固件）。若此时另一个固件状态为BOOT OK，则会启动另一个固件。

1. 代码库使用方法
2. 下载bootloader\_utilitie代码库压缩文件，添加到工程中
3. 需要添加的文件：nboot.c/flash.c/mcu\_flash\_xx.c，设置对应头文件include目录
4. 根据不同的平台，需要修改的位置：
5. 不同平台对应不同的mcu flash底层操作函数mcu\_flash\_xx.c
6. flash\_config.h的FLASH/RAM大小
7. nboot\_config.h的IMAGE\_META\_DATA\_ADDR（固件状态存储区起始地址，也对应于bootloader大小），IMAGE\_SLOT\_A\_START（固件A存放地址），MAX\_IMAGE\_SIZE（最大固件大小）
8. 在用户的main.c文件中，对时钟、外设等进行初始化(init)，完成外设等的相应操作后，反初始化外设(deinit)，最后调用boot\_app()启动应用程序
9. nboot.h中的API使用方法
10. 下载固件时：
    1. 调用nboot\_get\_free\_slot() 获取空闲的固件存储区
    2. 向a中的存储区下载固件
    3. 下载完成后，调用nboot\_set\_images\_boot\_state(slot\_id, IMAGE\_DOWNLOAD\_OK)，slot\_id就是a中获取的id
    4. 调用nboot\_set\_slot\_run\_after\_reset(slot\_id)，设置下次运行的固件id。slot\_id就是a中获取的id
    5. 调用system\_reboot() 重启，也可以等稍后手动重启
11. 固件运行时（需要更新当前固件的状态）：
    1. 调用nboot\_get\_slot\_run\_after\_reset() 获取当前运行的固件slot\_id
    2. 调用nboot\_set\_images\_boot\_state(slot\_id, IMAGE\_BOOT\_OK) 更新固件启动状态
    3. 调用nboot\_set\_slot\_running(slot\_id) 更新固件的运行状态