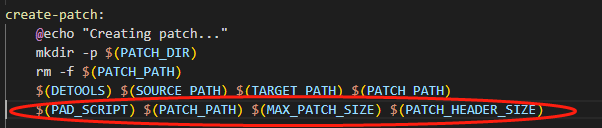
**本文档基于detools 开源算法，可用于测试对二进制文件的差分升级测试。**

工程路径中的Makefile文件已集成各种测试命令，具体思路如下；

首先用VSCode或者命令行编译出两个不同的binary 升级文件，其中一个为源文件（source.bin）,另一个为目的文件（target.bin）, 并基于这两个文件生成差分包（patch.bin）;然后将差分包烧入指定的flash区（PATCH\_STORAGE）。当MCU重启后会检查此块flash区中差分文件是否有效，主要是比较差分文件的头八个字节。新生成的差分包会在文件头部插入八个字节的标记（前四个字节为字符串“NEWP”,后四个字节为patch size）, 差分包apply后会改写前四个字节，这样MCU就不会重复执行apply操作了。MCU在bootloader中执行apply patch.bin的操作，将旧的image（source.bin）和patch.bin结合生成新的image并保存在flash slot1区域, 然后我们可以将slot1 区域的数据拷贝出来和target.bin 文件对比，若二者一致，则证明apply成功生成了新的image。

**MCU 端测试步骤：**

1. VSCode 编译先后生成两个不同的app\_update.bin文件， 并将其拷入工程目录下的binaries\signed\_images文件夹，将其中一个改为source.bin即源文件，另一个改为target.bin即目标文件
2. 烧录bootloader文件，可直接用VSCode烧录编译好的工程，因为已经使能了MCUBoot宏，所以会自动将MCUBoot一起烧录；
3. 在工程目录下打开git bash，输入指令 make flash-image烧入源文件，即source.bin;
4. 生成差分包patch.bin, 输入指令 make create-patch. 注意此时生成的patch,bin文件头部会插入八字节的标志位，这是由python脚本完成的，我们只需要在Makefile文件的create-patch指令后面加入如下指令即可。



1. 擦除flash 的 slot1区域，为新生成的image准备空间，输入指令 make erase-slot1
2. 烧写patch.bin到指定的flash区域， 执行指令 make flash-patch, 烧录完成后MCU会重启，在MCUboot中会完成apply操作，生成新的image存在slot1区域。如下所示

A screenshot of a computer

Description automatically generated

1. 输入指令 make dump-slot将slot1区域中的新生成的image拷贝出来并保存在binaries\flash\_dumps\slot1.bin文件中，可通过软件对比该文件和target.bin文件是否一致。或者输入命令 make flash-slot1将该文件直接烧录进primary slot,查看MCU的输出是否和target.bin一致。

**PC 端测试：**

1. VSCode 先后编译生成两个不同的app\_update.bin文件， 并将其拷入工程目录下的binaries\signed\_images文件夹，将其中一个改为source.bin即源文件，另一个改为target.bin即目标文件
2. 生成差分包patch.bin, 输入指令 make create-patch. 注意此时生成的patch,bin文件头部不需要插入八字结的标志位，否则执行apply操作的时候会报错。此时需要注释掉Makefile文件的create-patch指令最后面的一行脚本。

A picture containing graphical user interface

Description automatically generated

1. 执行apply操作， 输入指令 make apply-patch, 此时PC会将souce.bin和patch.bin结合生成新的image保存在binaries\flash\_dumps\target.bin文件中。
2. 烧入新生成的image文件， 输入命令 make flash-apply-target， 此时查看MCU的输出是否和target.bin一致。 也可以直接用软件对比新生成的image和target.bin是否一致。

注意：

初次测试时需要确保所需工具安装完成， 可执行 make tools指令自动安装所需工具， 主要工具有make， detools， pyocd, pynrfjprog 等。