单片机IAP协议

1. transport CRC 协议数据包

本文提到的所有数据包在发送、接收时，都遵循以下格式：

帧起始	ID	CMD_TYPE	CMD	SUB	DATA_LEN	DATA	CRC	帧结尾
0xAA, 0xAA	0x00	1 Byte	1 Byte	1 Byte	2 Byte, 低字节在前	长度由前一字段定义	2字节	0x55, 0x55

其中：

- CRC校验的是从“seq”字段开始到“data”字段的内容
- 打包时，除了增加0xAA和0x55的帧头尾和crc校验字段以外，如果从“seq”开始到“data”结束，还存在0xAA、0x55或0xA5数据，则在其前面增加一个0xA5作为转义字符
- 升级分为独立升级和同步升级模式，独立升级适合板卡仅有一个MCU的情况，同步升级适合板卡有双回路MCU的情况，二者不兼容。在同步升级模式下，默认的与升级上位机连接的MCU为主MCU，另一MCU位从MCU，主MCU会将升级指令同步给从MCU，主MCU默认ID为0，从MCU默认ID为1；

2. 存储器内固件结构

以SPC100为例，具有以下结构：

地址 0 ~ 0xC000 (48 kB)	0xC000 (48kB) ~ 0x10000 (64kB)	0x10000 (64 kB) 之后
bootloader	param 参数块	app

- param 参数块的结构如下：

```
typedef struct {
    char fw_version[64];
    char fw_name[64];
    uint8_t fw_md5[16];
    uint32_t fw_length;
} __attribute__((packed)) simple_firmware_info_t;

typedef struct {
    uint32_t magic;
    uint16_t upgrade_code;
    simple_firmware_info_t bootloader_info;
    simple_firmware_info_t application_info;
    simple_firmware_info_t upgrade_application_info;
    uint8_t uid_encrypt[12];
    uint8_t firmware_info_md5[16];
} __attribute__((packed)) merge_firmware_info_t;
```

3. 固件文件结构

每次会发布两个固件文件：`xxx-image-vx.x.x-0-gxxxxxxx.hex` 和 `xxx-app-vx.x.x-0-gxxxxxxx.bin`

其中，`hex`文件同时包含`bootloader`、`param`和`app`，直接刷入单片机即可。

`bin`文件包含`app`和`bintail`，是一个纯二进制文件。`bintail`中包含了此app的文件名、版本号等信息，位于bin文件的末尾，其结构如下:

```
typedef struct {
    uint32_t magic; // magic number must be 0xdeadbeef, to make sure that the
firmware_info partition is exist
    char fw_version[64]; // must make sure that the bytes after '\0' are all
0x00, instead of 0xff
    char fw_name[64];
    uint8_t fw_md5[16];
    uint32_t fw_length;
    uint8_t fw_info_md5[16];
} __attribute__((packed)) firmware_info_t;
```

`bintail`保证了固件的合法性：即使固件的文件名被更改，或其他错误固件的文件名被改为正确的文件名，也无法刷入到单片机中导致变砖。

4. 命令集定义

4.1. 命令类型枚举(CMD_TYPE)

```
enum {
    CMD_TYPE_BL     = 0x20,
};
```

4.2. 命令枚举(CMD)，仅列举`CMD_TYPE_BL`分类下的命令

```
// CMD_TYPE_BL option
enum {
    CMD_BL_NONE = 0X00,
    CMD_BL_STATUS = 0x20,
    CMD_BL_REBOOT = 0x21,
    CMD_BL_RUN_APP = 0x22,
    CMD_BL_DATA_TRANSFER = 0x23,
};
```

4.3. 子命令枚举(SUB_CMD)，列举`CMD_BL_DATA_TRANSFER`的子命令

```
// CMD_BL_DATA_TRANSFER option
enum {
    SUB_BL_DT_SOH = 0x01,  // start of data packet
    SUB_BL_DT_STX = 0x02,  // Data packet transfer
    SUB_BL_DT_EOT = 0x03,  // End of transmission

    SUB_BL_SOH_ACK = 0x10,  // packet verify OK
    SUB_BL_STX_ACK = 0x11,  // packet verify OK
    SUB_BL_EOT_ACK = 0x12,  // packet verify OK
};
```

列举CMD_BL_STATUS的子命令

```
// BL_STATUS option
enum {
    SUB_BL_STS_START = 0x00,
    SUB_BL_STS_APP_RUN = 0x01,
};
```

5. 升级流程与数据包定义

本节中提到的所有数据包，均是transport CRC**打包前**的数据包，实际发送和接收时都要按照transport CRC打包/解包处理。

5.1. 升级前准备

【上位机】打开`xxx-app-vx.x.x-0-gxxxxxxx.bin`文件，读取`bintail`。对固件（不包含`bintail`的部分）进行md5校验，并与`bintail`中记录的md5进行比对。只有二者md5相等的情况才能说明固件合法。

5.2. 开始升级

【上位机】向单片机发送reset命令：`0x00, CMD_TYPE_BL, CMD_BL_REBOOT, 0x00, 0x00, 0x00`。

frame[0]	命令类型	命令	子命令	data长度	data
0x00	CMD_TYPE_BL	CMD_BL_REBOOT	0	9	0x0d, 0x74, 0x6F, 0x5F, 0x62, 0x6F, 0x6F, 0x74, 0x0d

【单片机app】接收到reset命令，执行重启。

【单片机bootloader】bootloader启动时，向上位机发送bootloader启动消息：`0x00，CMD_TYPE_BL，BL_STATUS，CMD_BL_NONE，0x01，0x00，SUB_BL_STS_START`。

frame[0]	命令类型	命令	子命令	data长度	data
0x00	CMD_TYPE_BL	BL_STATUS	SUB_BL_STS_START	1	[升级模式](0-独立升级，1-同步升级)

【上位机】等待bootloader启动消息，若超时，则报错。

【上位机】若收到bootloader启动消息，则**立即**开始发送**首数据包**信息，首数据包内容：

frame[0]	命令类型	命令	子命令	data长度	data
0x00	CMD_TYPE_BL	CMD_BL_DATA_TRANSFER	SUB_BL_DT_SOH	4+len(len不定长,根据data内容而定)	见下表定义

“data”字段定义

固件大小	固件名称长度	固件名称	固件版本长度	固件版本
4字节	1字节	长度为固件名称长度，为不定长	1字节	长度为固件版本长度，为不定长

注：固件长度和名称都从`bintail`中读取，名称字符串不够64字节，剩余部分用0x00补齐。

【单片机bootloader】启动后，等待500ms。在500ms内接收到上位机发送的“首数据包”时，进入固件传输模式。否则，直接跳转到app。

【单片机bootloader】进入固件传输模式后：

1.判断接收到的名称字符串中是否包含“app”和`VALID_NAME`字段。`VALID_NAME`定义在bootloader代码中，用于判断此固件与bootloader是否匹配，防止下错固件。

2.判断固件大小是否符合芯片预留内存

3.判断是否为降版本升级

若判断固件不符合升级规则，需要上报：

【SOH故障码】

故障码	故障定义
0	固件合法
-1	名称非法
-2	固件过大
-3	降版本升级

frame[0]	命令类型	命令	子命令	data长度	data
0x00	CMD_TYPE_BL	CMD_BL_DATA_TRANSFER	SUB_BL_SOH_ACK	0x01, 0x00	见上表【SOH故障码】

5.3. 固件传输

【上位机】把固件（不含`bintail`的部分）按每256字节进行分解，然后对每一个packet进行编号（编号从0开始），然后依次打包发送，发送的格式为：

frame[0]	命令类型	命令	子命令	data长度	data
0x00	CMD_TYPE_BL	CMD_BL_DATA_TRANSFER	SUB_BL_DT_STX	4 + n（16位，低字节在前）	4字节packet编号（低字节在前）+ n字节固件

注：最后一包固件长度可能不足256字节，此时按照真实大小来传输，而不能凑出256字节去传输。”data长度“字段也必须是正确的长度。

【STX故障码】

故障码	故障定义
0	包序号正确
-1	包序号错误

【单片机bootloader】开始传输时，单片机内部也有一个packet编号变量`reqpackseq`，含义为”**请求的数据包的序号**“，其初始值为0。每次收到上位机发送的固件包时，判断上位机发送的固件包是否与单片机希望的固件包序号相等：若相等，则把此256字节固件写入到flash中，然后`reqpackseq`自增1，并向上位机返回ACK消息：

frame[0]	命令类型	命令	子命令	data长度	data
0x00	CMD_TYPE_BL	CMD_BL_DATA_TRANSFER	SUB_BL_STX_ACK	5	1字节【STX故障码】+4字节`reqpackseq`（低字节在前）

【上位机】每次发完一包数据，都等待单片机的ACK或NAK消息。有以下几种情况需要处理：

1）1000ms内未收到任何消息，重发当前数据包。重试5次都没有收到任何消息，报错超时，退出在线升级功能。

2）收到ACK消息，并且ACK消息中携带的`reqpackseq`恰好为自己将要发送的下一个包的序号。上位机正常发送下一个包。

3）收到NAK消息，携带的`reqpackseq`是自己刚刚已经发送的包的序号。上位机重发此包。

4）收到NAK消息，并且NAK消息中携带的`reqpackseq`恰好为自己将要发送的下一个包的序号。这可能是因为上位机没收到上次正确传输的ACK，上位机重发，导致目标收到重复的包导致的。这种情况下，上位机日志记录"目标收到重复的包"，然后正常发送下一个包即可。

5）收到ACK或NAK消息，并且`reqpackseq`既不是自己已经发送的包的序号，又不是即将发送的包的序号。说明传输出现了严重丢包，直接退出在线升级功能并报错。

5.4. 传输结束

【上位机】传输完最后一个数据包，并且接收到最后一个ACK之后。开始发送**尾数据包**：

frame[0]	命令类型	命令	子命令	data长度	data
0x00	CMD_TYPE_BL	CMD_BL_DATA_TRANSFER	SUB_BL_DT_EOT	16	16 Byte，md5值

注：md5值直接从`bintail`中读取。

【EOT故障码】

故障码	故障定义
0	固件升级成功
-1	实际接收到的固件大小不符合SOH阶段下发下来的预下载固件大小
-2	固件MD5校验不通过
-3	固件已升级到备份区，未能完成覆盖到运行区

若固件校验不通过，回复ACK：

frame[0]	命令类型	命令	子命令	data长度	data
0x00	CMD_TYPE_BL	CMD_BL_DATA_TRANSFER	SUB_BL_EOT_NAK	1	见上表【EOT故障码】

5.5. 程序启动

【单片机bootloader】单片机接收固件完成，并且md5也校验结束，需要再次覆写flash存储器中的param块，把`fw_valid`设为1，把`fw_size`设为接收到的固件大小。

然后开始执行app跳转。

【单片机bootloader】跳转程序是一个通用程序，无论是刚上电还是刚下载完，都需要执行此程序才能跳转到app。跳转程序中会执行一些检查，若这些检查不通过，会上报以下数据包：

frame[0]	命令类型	命令	子命令	data长度	data
0x00	CMD_TYPE_BL	CMD_BL_STATUS	SUB_BL_STS_APP_RUN	1	1 字节 **错误码**

错误码列举：

- 0：跳转成功
- -1 ： 固定校验码(magic)不通过
- -2 ： APP固件主栈指针（MSP）初始值不在SRAM范围内，说明固件文件在添加`bintail`之前就有错误
- -3：固件md5校验失败。flash中存储的固件与param块中存储的md5值不匹配
- -4：固件名称不对。param块中存储的固件名称不包含`VALID_NAME`或不包含“app”字段
- -5：固件信息MD5校验失败，param块中的固件信息存在错误
- -6：UID验证不通过

【上位机】等待单片机重启5s，然后一直检测上述数据包，读取错误码。根据不同的结果，显示相应文案。