1. 角色介绍

- `MCU0` 主控芯片
- `MCU1` 数据采集芯片

2. 资源规划

2.1. MCU0 Flash Code 区规划

	空间大小(KB)	十六进制(KB)	开始地址	结束地址
Bootloader	64	0x00010000	0x08000000	0x0800FFFF
Application	704	0x000B0000	0x08010000	0x080BFFFF

2.2. MCU1 Flash Code 区规划

	空间大小(KB)	十六进制(KB)	开始地址	结束地址
Bootloader	64	0x00010000	0x08000000	0x0800FFFF
Application	64	0x00010000	0x08010000	0x0801FFFF
Backup	64	0x00010000	0x08020000	0x0802FFFF
Reserved	64	0x00010000	0x08030000	0x0803FFFF

- Backup 区用于升级时，先将要升级的区域备份，然后升级，一旦升级失败，进行回滚

2.3. External Flash 规划

- `0x00000000`: 配置区
- `0x00100000`: MCU0_BOOT 信息区
- `0x00101000`: MCU0_BOOT 下载区
- `0x00133000`: MCU0_APP 信息区
- `0x00134000`: MCU0_APP 下载区
- `0x002C4000`: MCU1_BOOT 信息区
- `0x002C5000`: MCU1_BOOT 下载区
- `0x002F7000`: MCU1_APP 信息区
- `0x002F8000`: MCU1_APP 下载区

区段开始	信息容量(KB)	数据容量(KB)	区段总容量(KB)	区段结束	名称	说明	信息区开始位置	数据区起始位置	数据区结束位置
00000000			1024	00100000	配置区				
00100000	4	100	104	0011A000	MCU0_BOOT下载区	GD32F470 BOOT区 64KB	00100000	00101000	0011A000
0011A000	0	100	100	00133000	MCU0_BOOT备份区	GD32F470 BOOT区 64KB	0011A000	0011A000	00133000
00133000	4	800	804	001FC000	MCU0_APP下载区	GD32F470 应用区 704KB	00133000	00134000	001FC000
001FC000	0	800	800	002C4000	MCU0_APP备份区	GD32F470 应用区 704KB	001FC000	001FC000	002C4000
002C4000	4	100	104	002DE000	MCU1_BOOT下载区	GD32F303 BOOT区 64KB	002C4000	002C5000	002DE000
002DE000	0	100	100	002F7000	MCU1_BOOT备份区	GD32F303 BOOT区 64KB	002DE000	002DE000	002F7000
002F7000	4	256	260	00338000	MCU1_APP下载区	GD32F303 应用区 192KB	002F7000	002F8000	00338000
00338000	0	256	256	00378000	MCU1_APP备份区	GD32F303 应用区 192KB	00338000	00338000	00378000
		一共使用:	3552						
		剩余容量	12832						

3. 下载固件过程(`MCU0_BOOT`)

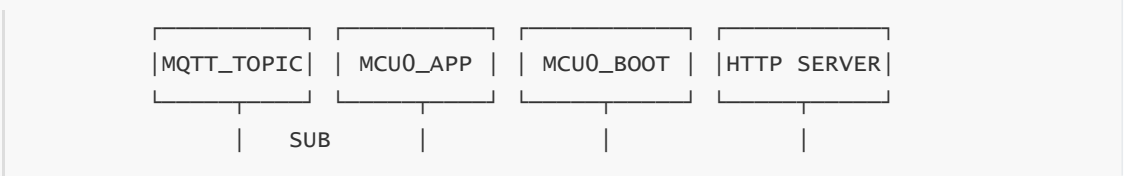
下载固件由 `MCU0_BOOT` 完成

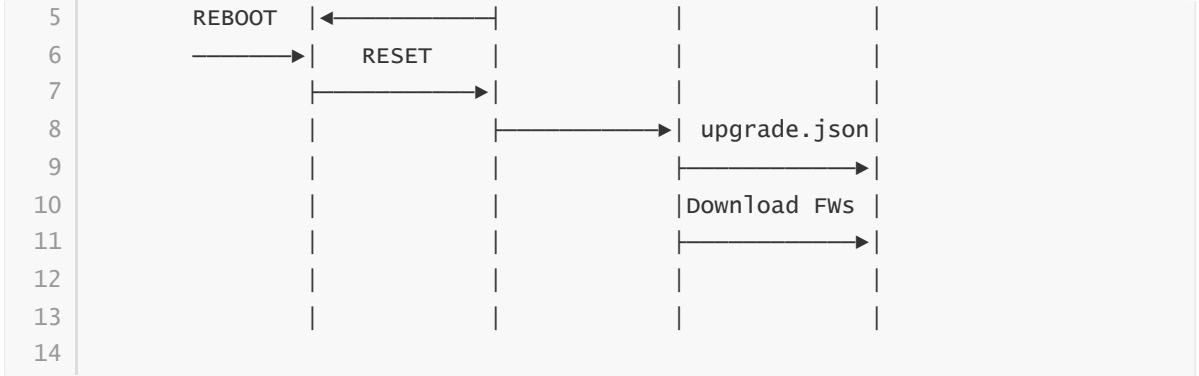
3.1. 升级触发

1. 通过向 `/Devices/[CPUID]/DOWNSTREAM` MQTT TOPIC 发出命令 `reboot`
2. `MCU0_APP` 收到该指令时，自动重启，在启动时，检查是否需要升级

3.2. 读取 `upgrade.json`

路径在: `http://host:port/Devices/[CPUID]/upgrade.json`





`upgrade.json` 格式

```
{
    "model":"DC_METER",
    "MCU0_APP":{
        "v":2024070102,
        "s":48584,
        "md5":"8087c4351d355358e388b31a563aa494",
 "url":"http://ubattery.cn:19093/Devices/5C6633343539360648343430/MCU0_APP.bin"
    },
    "MCU0_BOOT":{
        "v":<number>,
        "s":<number>,
        "md5":"<string>",
        "url":"<string>"
    },
    "MCU1_BOOT":{
        "v":<number>,
        "s":<number>,
        "md5":"<string>",
        "url":"<string>"
    },
    "MCU0_APP":{
        "v":<number>,
        "s":<number>,
        "md5":"<string>",
        "url":"<string>"
    },
}
```

3.3. 固件信息区

```
typedef struct iap_firmware_info_s{
    uint8_t     type;
    uint32_t    remote_version;
    uint32_t    download_version;
    uint32_t    installed_version;
    uint8_t     md5[16];
    uint8_t     state;
    uint32_t    size;
    int8_t      is_downloaded;
}iap_firmware_info_t;
```

- `remote_version`: 服务器上版本信息, 下载时设置
- `download_version`: 下载版本，下载完成时设置
- `installed_version`: 安装版本, 安装完成时设置
- `md5`: 下载完成时设置
- `size`: 下载完成时设置

3.4. 下载固件

下载固件由 `MCU0_BOOT` 完成

根据 `url` 指定的路径下载，根据类型存储到 external flash 指定位置

- `0x00000000`: 配置区
- `0x00100000`: MCU0_BOOT 信息区
- `0x00101000`: MCU0_BOOT 下载区
- `0x00133000`: MCU0_APP 信息区
- `0x00134000`: MCU0_APP 下载区
- `0x002C4000`: MCU1_BOOT 信息区
- `0x002C5000`: MCU1_BOOT 下载区
- `0x002F7000`: MCU1_APP 信息区
- `0x002F8000`: MCU1_APP 下载区

4. 安装过程

4.1. 程序不能更新自己的区域代码

- `MCU0_BOOT` 不能向 `MCU0` 的 `Bootloader` 区域写入数据，否则会造成程序无法运行，硬件变砖头!!!

4.2. 安装 MCU0_APP

在完成下载后，MCU0_BOOT 首先检查是否需要升级 MCU0_APP，如果需要升级，立即向 MCU0_APP 的代码区域写入程序

```
static int iap__upgrade_mcu0_app(void){
    printf("[IAP] Upgrade MCU0_APP...\n");
    iap_firmware_info_t fw_info;
    int err = 0;
    MD5_CTX md5_ctx;
    uint8_t md5[16];
    int nRetry = 3;

    /* 读取 firmware 固件信息 */
    iap__firmware_info_read(IAP_FW_TYPE_MCU0_APP, &fw_info);

    if(fw_info.installed_version== fw_info.download_version){
        printf("[IAP] MCU0_APP download version == installed version! SKIP!!!\n");
        return IAP_RET_OK;
    }

    /* 确认外部 FLASH 可用 */
    uint32_t flash_id = sFLASH_ReadID();
    if(!sFLASH_IsValidID(flash_id)){
        return IAP_RET_ERROR;
    }

    /* 确认程序要更新的地址和读取地址 */
    uint32_t mcu_flash_address = IAP_MCU0_APP_FLASH_ADDR;
    uint32_t ext_flash_address = IAP_FW_MCU0_APP_DOWNLOAD_AREA;

__iap__upgrade_mcu0_app_program:
    if(nRetry----0){
        return IAP_RET_ERROR;
    }


    /* 先擦除再写入 */
    size_t mcu_pages = PAGE(fw_info.size, MCU_FLASH_PAGE_SIZE);
    uint32_t used_sector_size = 0;

    /* 关闭中断 */
    __disable_irq();

    /* FLASH 写入规定： 写入前必须先擦除 */
    /* unlock the flash program erase controller */
    fmc_unlock();
    /* get the information of the start and end sectors */
    fmc_sector_info_struct start_sector_info = fmc_sector_info_get(mcu_flash_address);
    fmc_sector_info_struct end_sector_info = fmc_sector_info_get(mcu_flash_address + fw_info.size);
    /* erase sector */
```

```
    for(uint32_t i = start_sector_info.sector_name; i <= end_sector_info.sector_name; i++){
        uint32_t sector_num = sector_name_to_number(i);
        printf("[IAP] Erase MCU sector %d\n", i);
        /* clear pending flags */
        fmc_flag_clear(FMC_FLAG_END | FMC_FLAG_OPERR | FMC_FLAG_WPERR | FMC_FLAG_PGMERR | FMC_FLAG_PGSERR);

        if(FMC_READY != fmc_sector_erase(sector_num)){
            printf("[IAP] ERR erase sector %d failed!\n", i);
            err = IAP_RET_ERROR;
            goto __iap__upgrade_mcu0_app_exit;
        }
    }


    /* 准备校验 MD5 */
    MD5Init(&md5_ctx);

    uint32_t read_size = 0;
    uint32_t total_read = 0;
    uint32_t write_address = mcu_flash_address;

    /* 写入MCU内部代码区域 */
    while(1){
        read_size = fw_info.size - total_read;
        read_size = (read_size < MCU_FLASH_PAGE_SIZE)? read_size:MCU_FLASH_PAGE_SIZE;
        sFLASH_ReadBuffer(iap__download_buffer, ext_flash_address+total_read, read_size);

        MD5Update(&md5_ctx, iap__download_buffer, read_size);

        printf("[IAP] %d/%d bytes, Read from 0x%08x - %d bytes from W25Q128\r\n"
                , total_read
                , fw_info.size
                , ext_flash_address+total_read
                , read_size);

        uint32_t* word_addr = (uint32_t*)iap__download_buffer;
        int word_page = PAGE(read_size, 4);

        for(int x=0; x< word_page; x++){
            fmc_flag_clear(FMC_FLAG_END | FMC_FLAG_OPERR | FMC_FLAG_WPERR | FMC_FLAG_PGMERR | FMC_FLAG_PGSERR);
            if (FMC_READY != fmc_word_program(write_address, word_addr[x])) //write
            {
                printf("[IAP] Write MCU Flash Addr %08x Failed!\r\n", write_address);
                err = IAP_RET_ERROR;
                goto __iap__upgrade_mcu0_app_exit;
            }
            write_address+=4;
        }
```

```
        total_read += read_size;

        if(total_read==fw_info.size){
            break;
        }
    }

    /* 完成 MD5 校验码计算 */
    MD5Final(md5, &md5_ctx);

    /* 校验MD5是否正确 */
    if(memcmp(md5, fw_info.md5, 16)!=0){
        printf("[IAP] ERROR Wrong MD5:\n");
        char md5_str[33];
        SDK_HEX_ENCODE_BE(md5_str, sizeof(md5_str), fw_info.md5,
sizeof(fw_info.md5));
        printf("Remote MD5: %s\n", md5_str);
        SDK_HEX_ENCODE_BE(md5_str, sizeof(md5_str), md5, sizeof(md5));
        printf("Check MD5: %s\n", md5_str);
        err = IAP_RET_ERROR;
        goto __iap__upgrade_mcu0_app_program; /* 校验失败，重新写入，一共重试3
次，如果3次都失败，设备变砖 */
    }

    /*升级成功*/
    fw_info.installed_version = fw_info.download_version;
    iap__firmware_info_write(&fw_info);
    printf("Upgraded Firmware Info Saved!\n");
    err = IAP_RET_OK;

__iap__upgrade_mcu0_app_exit:
    fmc_lock(); /* 重新锁定 MCU 内部 flash */
    return err;
}
```

4.3. 安装 `MCU1_BOOT`

1. `MCU0_BOOT` 向 发出升级指令, 这里有两种情况, 一种是 `MCU1` 正运行在 `MCU1_APP` 状态，一种是 `MCU1` 运行在 `MCU1_BOOT` 状态；
    - 运行在`MCU1_BOOT`状态，可以直接升级
    - 运行在`MCU1_APP`状态，因为 `MCU1_APP` 运行了很多任务，启用了很多设备驱动，这种情况下升级，很容易失败；另外 `MCU1_BOOT` 是不能自己写入 `MCU1_BOOT`的程序区的，因此必须先启动到 `MCU1_BOOT`，然后再启动回 `MCU1_APP`，这时系统处于干净的状态，可以安全升级

```
┌──────────┐  ┌──────────┐  ┌──────────┐
|MCU0_BOOT |  |MCU1_APP  |  |MCU1_BOOT |
└──────┬───┘  └──────┬───┘  └──────┬───┘
       |UPGRADE      |             |
       ├─────────────►| REBOOT      |
       |             ├─────────────►|
```

```
      |             | JUMP        |
      |             |◀────────────┤
      |UPG_READY    |             |
      |◀────────────┤             |
      |             |             |
      |SEND_FW_DATA|             |
      ├────────────▶|             |
      |             |             |
      |RECV_OK      |             |
      |◀────────────┤Write To     |
      |             |MCU0_BOOT    |
      |RECV_ERR     ├────────────▶|
      |◀────────────┤             |
      |SEND_FW_DATA|             |
      ├────────────▶|             |
      |             |             |
```

4.4. 安装 MCU1_APP

1. MCU0_BOOT 向 MCU1_APP 发送 UPGRADE 指令
2. MCU1_APP 在接收到 UPGRADE 指令后，重启，进入 MCU1_BOOT
3. MCU1_BOOT 检查到需要升级，向 MCU0_BOOT 发送 UPG_READY 指令
4. MCU0_BOOT 向 MCU1_BOOT 发送固件数据，命令为 SEND_FW_DATA
5. MCU1_BOOT 在接收到数据后，返回 RECV_OK，表示接收正常
6. MCU1_BOOT 在接收到数据后，返回 RECV_ERR，表示接收异常，
7. MCU0_BOOT 需要重新传输接收失败的数据
8. 在最后一个包发送完成后，MCU1_BOOT 重启
9. MCU0_BOOT 在接收到最后一个包也被正确接收后，重启，进入 MCU0_APP

```
┌────────────┐ ┌────────────┐ ┌────────────┐
|MCU0_BOOT | |MCU1_APP  | |MCU1_BOOT |
└─────┬──────┘ └─────┬──────┘ └─────┬──────┘
      |              |              |
      |(1)UPGRADE    |              |
      ├─────────────▶ |(2)REBOOT     |
      |              ├─────────────▶|
      |              |(3)UPG_READY|
      |◀─────────────┼──────────────┤
      |              |              |
  (4)|SEND_FW_DATA|              |
      ├─────────────┼─────────────▶|
      |              |              |
      |          (5)|RECV_OK       |
      |◀─────────────┼──────────────┤
      |          (6)|RECV_ERR      |
      |◀─────────────┼──────────────┤
  (7)|SEND_FW_DATA|              |
```

```
19      ├──────────┼──────────►|
20      |          |           |
21      |          |           |
```

4.5. 安装 MCU0_BOOT

其它固件都完成安装后，最后处理 `MCU0_BOOT` 的安装，这时需要启动到 `MCU0_APP` 中，`MCU0_APP` 启动时，先检查是否需要升级

```
1   static int iap__upgrade_mcu0_boot(void){
2       printf("[IAP] Upgrade MCU0_BOOT...\n");
3       iap_firmware_info_t fw_info;
4       int err = 0;
5       MD5_CTX md5_ctx;
6       uint8_t md5[16];
7       int nRetry = 3;
8
9       /*读取 MCU0_BOO 信息*/
10      iap__firmware_info_read(IAP_FW_TYPE_MCU0_BOOT, &fw_info);
11
12      /* 如果已经安装，直接返回 */
13      if(fw_info.installed_version== fw_info.download_version){
14          printf("[IAP] MCU0_BOOT download version == installed version!
    SKIP!!!\n");
15          return IAP_RET_OK;
16      }
17
18      /* 确认外部 FLASH 可用 */
19      uint32_t flash_id = sFLASH_ReadID();
20      if(!sFLASH_IsValidID(flash_id)){
21          return IAP_RET_ERROR;
22      }
23
24      /* 确认程序的写入地址和读取地址 */
25      uint32_t mcu_flash_address = IAP_MCU0_BOOT_FLASH_ADDR;
26      uint32_t ext_flash_address = IAP_FW_MCU0_BOOT_DOWNLOAD_AREA;
27
28  __iap__upgrade_mcu0_boot_program:
29      if(nRetry---==0){
30          return IAP_RET_ERROR;
31      }
32
33
34      /* 先擦除再写入 */
35      size_t mcu_pages = PAGE(fw_info.size, MCU_FLASH_PAGE_SIZE);
36      uint32_t used_sector_size = 0;
37
38
39      __disable_irq();
40      /* unlock the flash program erase controller */
41      fmc_unlock();
```

```
    /* get the information of the start and end sectors */
    fmc_sector_info_struct start_sector_info = fmc_sector_info_get(mcu_flash_address);
    fmc_sector_info_struct end_sector_info = fmc_sector_info_get(mcu_flash_address + fw_info.size);
    /* erase sector */
    for(uint32_t i = start_sector_info.sector_name; i <= end_sector_info.sector_name; i++){
        uint32_t sector_num = sector_name_to_number(i);
        printf("[IAP] Erase MCU sector %d\n", i);
        /* clear pending flags */
        fmc_flag_clear(FMC_FLAG_END | FMC_FLAG_OPERR | FMC_FLAG_WPERR | FMC_FLAG_PGMERR | FMC_FLAG_PGSERR);

        if(FMC_READY != fmc_sector_erase(sector_num)){
            printf("[IAP] ERR erase sector %d failed!\n", i);
            err = IAP_RET_ERROR;
            goto __iap__upgrade_mcu0_boot_exit;
        }
    }


    MD5Init(&md5_ctx);

    uint32_t read_size = 0;
    uint32_t total_read = 0;
    uint32_t write_address = mcu_flash_address;

    /* 一边读取下载的程序，一边写入 */
    while(1){
        read_size = fw_info.size - total_read;
        read_size = (read_size < MCU_FLASH_PAGE_SIZE)? read_size:MCU_FLASH_PAGE_SIZE;
        sFLASH_ReadBuffer(iap__download_buffer, ext_flash_address+total_read, read_size);

        MD5Update(&md5_ctx, iap__download_buffer, read_size);

        printf("[IAP] %d/%d bytes, Read from 0x%08x - %d bytes from W25Q128\r\n"
                , total_read
                , fw_info.size
                , ext_flash_address+total_read
                , read_size);

        uint32_t* word_addr = (uint32_t*)iap__download_buffer;
        int word_page = PAGE(read_size, 4);

        for(int x=0; x< word_page; x++){
            fmc_flag_clear(FMC_FLAG_END | FMC_FLAG_OPERR | FMC_FLAG_WPERR | FMC_FLAG_PGMERR | FMC_FLAG_PGSERR);
            if (FMC_READY != fmc_word_program(write_address, word_addr[x])) //write
            {
                printf("[IAP] Write MCU Flash Addr %08x Failed!\r\n", write_address);
```

```
                err = IAP_RET_ERROR;
                goto __iap__upgrade_mcu0_boot_exit;
            }
            write_address+=4;
        }

        total_read += read_size;

        if(total_read==fw_info.size){
            break;
        }
    }

    /* 完成 MD5 计算 */
    MD5Final(md5, &md5_ctx);

    /* 校验 MD5 值 */
    if(memcmp(md5, fw_info.md5, 16)!=0){
        printf("[IAP] ERROR Wrong MD5:\n");
        char md5_str[33];
        SDK_HEX_ENCODE_BE(md5_str, sizeof(md5_str), fw_info.md5,
sizeof(fw_info.md5));
        printf("Remote MD5: %s\n", md5_str);
        SDK_HEX_ENCODE_BE(md5_str, sizeof(md5_str), md5, sizeof(md5));
        printf("Check MD5: %s\n", md5_str);
        err = IAP_RET_ERROR;
        /* 写入失败，再次重试，最多重试3次 */
        goto __iap__upgrade_mcu0_boot_program;
    }

    /*升级成功*/
    fw_info.installed_version = fw_info.download_version;
    iap__firmware_info_write(&fw_info);
    printf("Upgraded Firmware Info Saved!\n");
    err = IAP_RET_OK;

__iap__upgrade_mcu0_boot_exit:
    /*锁定MCU内部 flash，不允许随意改动*/
    fmc_lock();
    return err;
}
```