BK-MESH-OTA模块指南

Beken Corporation
Building 41, Capital of Tech Leaders, 1387 Zhangdong Road,
Zhangjiang High-Tech Park, Pudong New District, Shanghai, China
Tel： (86)21 51086811
Fax： (86)21 60871089

目录

版本	发布/更新日期	更新人员	重要变更内容
V1.0.0	2020/12/12	关文瑞	初始版本
V1.0.1	2021/1/29	关文瑞	用户使用版本

1. 文档概述

BEKEN-MESH工程，对于博通硬件平台和操作系统以及协议栈具有灵活的可移植性。本文基于BK3633芯片平台上Ali-OS的MESH工程，详细介绍OTA功能的使用和程序定制化修改。

2. 镜像文件结构分析

2.1. Flash 区域划分

MESH工程编译出的用于烧录的镜像文件，由三个基础工程拼接而成，分别为Boot、Controller-stack以及Upper-OS。其中Boot及Controller-stack由Beken开发人员进行维护并提供库及烧录文件，Upper-OS中集成了开源的BLE-stack、MESH协议以及操作系统，不定期向用户提供更新代码版本，可由客户根据使用需求自行修改。

文件board.c中向用户展示了当前工程中flash区域的划分，定义为专项结构体数组，如下图所示：

```
const hal_logic_partition_t hal_partitions_4M[] =
{
    [HAL_PARTITION_BOOTLOADER] =
    {
        .partition_owner            = HAL_FLASH_EMBEDDED,
        .partition_description      = "Bootloader",
        .partition_start_addr       = 0x000000,
        .partition_length           = PARTITION_STACK_CRC_ADDR - 0,
        .partition_options          = PAR_OPT_READ_EN,
    },
    [HAL_PARTITION_BT_FIRMWARE] =
    {
        .partition_owner            = HAL_FLASH_EMBEDDED,
        .partition_description      = "BLE FW",
        .partition_start_addr       = PARTITION_STACK_CRC_ADDR,
        .partition_length           = PARTITION_APP_CRC_ADDR - PARTITION_STACK_CRC_ADDR,
        .partition_options          = PAR_OPT_READ_EN,
    },
    [HAL_PARTITION_APPLICATION] =
    {
        .partition_owner            = HAL_FLASH_EMBEDDED,
        .partition_description      = "Application",
        .partition_start_addr       = PARTITION_APP_CRC_ADDR,
        .partition_length           = PARTITION_OTA_TEMP_CRC_ADDR - PARTITION_APP_CRC_ADDR,
        .partition_options          = PAR_OPT_READ_EN,
    },
    [HAL_PARTITION_OTA_TEMP] =
    {
```

结构体数组中可以浏览对应各个分块名称的起始地址以及长度等详细信息。其中HAL_PARTITION_BOOTLOADER对应了boot.bin的存储区域，HAL_PARTITION_BT_FIRMWARE则存放了controller.bin，由于这两个模块已经预先编译完成，所以起始地址和长度不允许修改。

HAL_PARTITION_APPLICATION是Upper-OS有效代码的存储区域，起始地址已经写入controller跳转的固定地址，故不可修改，其长度随着编译结果而改变。当编译后的Upper-OS镜像文件超出原划分大小时，修改该区域长度是必要的。

其后是OTA备份区以及功能预留SECTION。原则上如果没有修改过相应的功能，不可更改预留SECTION。OTA备份区域长度限制了OTA升级文件的大小。

2.2. Flash 区域修改

a) 区域划分的原则

Upper-OS区域的起始地址不可独立于Boot和Stack模块自行修改。

用户层代码在修改后，需要格外关注编译生成的文件大小是否已经超出当前软件内的分区上限。如若忽略边界，可能在程序运行的过程中造成新写入的备份文件覆盖掉了正常执行的代码区域等不可挽回的错误。如此，在大量修改客户程序后，必须注意新的镜像文件占据空间是否已经超出了OTA备份区的起始标线。

此外仍需重点注意的是， FLASH芯片擦除操作最小的单位sector为4KByte，即其擦除驱动程序以(传入地址&(~0xFFF))为起始地址，擦除长度是0x1000的整数倍。一般在进行flash区域划分的时候处于对资源的充分利用原则，模块之间尽量安排紧凑，但是不可以将可独立擦除的不同模块在同一个sector内接壤。故而大多是以4K的整数倍为基准划分。

b) 静态存储区

在Flash末端预先划分了一块起始地址0x7F000、长度4K的区域，将其命名为HAL_PARTITION_STATIC_PARA的静态存储区。

该区域的使用规则是，预先定义片段，每个片段存储各自的内容，互相不干涉，对某一片段的改写不允许影响其他片段内容。

```
platform_static_partition_s bk3633_static_partition[] =
{
    {STATIC_SECTION_OTA, 0xFC, 0},
    {STATIC_SECTION_MAC, 0x100, 0xFC},

};
```

有效片段会包含一个特定Header，读取片段时会通过校验Header来确定当前片段存储数据的有效性。当数据写入时，会先将其他片段信息提取，然后擦除整个Sector，再将修改数据和提取数据拼接后写入。

如上，第一个片段存储了OTA相关的信息。硬件首次通过串口烧录代码时可通过特殊工具写入结构体数据，默认为空，在OTA升级后会将新的信息录入。该片段不可替换。

3. OTA 功能模块

3.1. OTA 层级结构

为增强模块的可移植性，OTA进行分层设计，分为平台层、抽象层、协议层。平台层与芯片具体实现相关，在层级内部分配硬件资源以及调用驱动程序，不同的芯片平台可能有不同的实现代码；抽象层将平台层的各项功能，打包成OTA模块会使用到的通用函数接口，上层只需按照需求的参数形式调用对应函数；在协议层预先内置了BEKEN-OTA协议，搭配BEKEN手机APP实现了一个简单的OTA应用传输示例，用户可依照范例自定传输协议，或叠加更复杂的上层应用接口。

a) 抽象层

抽象层提供了全部OTA所用到的函数接口，存放于文件hal_ota.c，该文件定义了一个结构体hal_ota_module_s装配了平台提供的全部OTA相关功能。抽象层将其包裹为hal_xxx函数形式在用户层调用。

```
struct hal_ota_module_s {
    hal_module_base_t base;

    /* Link to HW */
    int (*init)(void *something);
    int (*deinit)(void *something);
    int (*ota_save)(uint8_t *in_buf , uint32_t in_buf_len);
    int (*ota_read)(volatile uint32_t *off_set, uint8_t *out_buf , uint32_t out_buf_len);
    int (*ota_set_boot)(void *something);
    int (*ota_rollback)(void *something);
    int (*ota_tag_check)(uint32_t ota_type, uint16_t ver, uint32_t size);
    int (*ota_tag_get)(void *something);
};
```

hal_ota_register_module：将平台功能模块注册到抽象层，后续调用抽象层函数则全部使用对应平台功能。

```
/**
 * Arch register a new module before HAL startup
 * @param  module  the pointer to the platform ota module
 */
void hal_ota_register_module(hal_ota_module_t *module);
```

hal_ota_init：在使用功能之前对注册的OTA模块进行初始化。
hal_ota_deinit：在OTA进程中断时清除注册痕迹并释放分配的资源。
hal_ota_save：存储OTA数据。
hal_ota_read：读取存入的OTA数据。
hal_ota_tag_get：获取当前OTA信息串。
hal_ota_tag_check：检查当前升级中文件的信息串，与已烧录的数据进行对比。
hal_ota_set_boot：升级完成执行boot操作。

b) 平台层

平台层内容是抽象层函数在具体芯片上的实现，涉及芯片驱动程序的调用以及Flash空间资源的分配。定义了hal_ota_module_s结构体内容供上层注册使用。

```
struct hal_ota_module_s bk3633_ota_module = {
    .init = bk3633_ota_init,
    .deinit = bk3633_ota_deinit,
    .ota_save = bk3633_ota_save,
    .ota_read = bk3633_ota_read,
    .ota_set_boot = bk3633_ota_set_boot,
    .ota_tag_check = bk3633_ota_tag_check,
    .ota_tag_get = bk3633_ota_tag_get,
};
```

bk3633_ota_init：执行OTA功能初始化，将FLASH内OTA备份区完全擦除并初始化环境变量，创建crc_table。此处定义OTA_PAGE_SIZE作为OTA一次写入Flash数据量的上限，可根据使用情况更改，当传入数据不足时会暂存在分配的变量空间内。
bk3633_ota_deinit：还原，释放环境变量空间。
bk3633_ota_save：传入OTA数据，数据会暂存在环境变量空间中直到达到写入上限，然后一次性写入FLASH内OTA备份区，偏移量自动累加，并迭代计算CRC校验结果。

bk3633_ota_read：读取当前备份区内某偏移量某长度的数据内容。

bk3633_ota_tag_check：在升级开始前对文件信息的检查。函数传入手机端文件的版本、大小以及升级模式等，与当前执行代码的对应信息做对比，如果合法则暂存在环境变量中。

bk3633_ota_tag_get：获取HAL_PARTITION_STATIC_PARA中存储的关于当前版本的信息。

bk3633_ota_set_boot：将当前变量区内暂存的剩余数据一次性写入，检查完整性之后将更新后的版本信息存入HAL_PARTITION_STATIC_PARA，执行boot操作。

c) 协议层

ota_service.c文件内注册OTA SERVICE并依照BEKEN-OTA-DEMO实现了一个简单的OTA应用。这里不对BLE服务的注册过程做详细介绍。

```
static struct bt_gatt_attr _ota_srv_attrs[] = {
    BT_GATT_PRIMARY_SERVICE(BEKEN_OTA_SERVICE_UUID),

    BT_GATT_CHARACTERISTIC(BEKEN_OTA_IMG_IDENTIFY_UUID, BT_GATT_CHRC_READ | BT_GATT_CHRC_WRITE | BT_GATT_CHRC_NOTIFY),
    BT_GATT_DESCRIPTOR(BEKEN_OTA_IMG_IDENTIFY_UUID, BT_GATT_PERM_WRITE | BT_GATT_PERM_READ, beken_ota_ffc1_read, beken_ota_ffc1_write, NULL),
    BT_GATT_CCC(beken_ota_ffc1_ccc_cfg, beken_ota_ffc1_ccc_cfg_changed),
    BT_GATT_CUD(NULL, BT_GATT_PERM_READ),

    BT_GATT_CHARACTERISTIC(BEKEN_OTA_IMG_BLOCK_UUID, BT_GATT_CHRC_READ | BT_GATT_CHRC_WRITE | BT_GATT_CHRC_NOTIFY),
    BT_GATT_DESCRIPTOR(BEKEN_OTA_IMG_BLOCK_UUID, BT_GATT_PERM_WRITE | BT_GATT_PERM_READ, beken_ota_ffc2_read, beken_ota_ffc2_write, NULL),
    BT_GATT_CCC(beken_ota_ffc2_ccc_cfg, beken_ota_ffc2_ccc_cfg_changed),
    BT_GATT_CUD(NULL, BT_GATT_PERM_READ),
};
```

BEKEN-OTA-DEMO注册了两个服务，UUID为FCC1的服务用于交互版本信息，FCC2用于传输文件数据。服务链接建立后先向FCC1发送空包来获取当前执行版本的信息，并显示在手机端；手机端选取升级文件后，在对应位置显示更新的版本信息，若不匹配则禁止升级。验证通过后同样通过FCC1发送新文件的版本信息，供板端校验和存储。

DEMO中原始版本信息数据以beken_image_hdr形式在执行编译时由脚本文件固定写在镜像文件的前16个字节，不计入正式数据长度，即当前的逻辑没有将其直接写入HAL_PARTITION_STATIC_PARA的对应OTA区域。只能在首次升级时主动读出该字段并写入HAL_PARTITION_STATIC_PARA。

数据由手机端通过FCC2单方面传入，无需板端回复。当前固定数据长度为16字节，前面附加两个字节的段号。开发者可根据手机能力对传输长度做相应修改。

BEKEN-OTA-DEMO的协议流程较为简单，客户可参考开发自己的工程。

3.2. BOOT 模块

OTA数据传输成功后，新文件覆写旧文件的操作在重启后的Boot模块中执行。该模块先检查HAL_PARTITION_STATIC_PARA中的版本相关信息，如果有效，则将源地址数据搬移到目标地址。

该模块中Controller-stack的跳转地址以及HAL_PARTITION_STATIC_PARA中OTA信息位置为固定值，不可由用户自行修改，用户代码若修改可能会造成不可还原的错误；其他分区，如Upper-OS和OTA备份区的起始地址和长度，Boot模块没有默认值，该值存储在升级文件传输成功后程序校验写入的HAL_PARTITION_STATIC_PARA中，从而应用层可以灵活更改烧写区域起始地址和长度，但上层需要预先自己验证新的存储区域划分没有重叠。