HONOR Connect IoT SDK集成开发指导

文档版本 V1.0.0

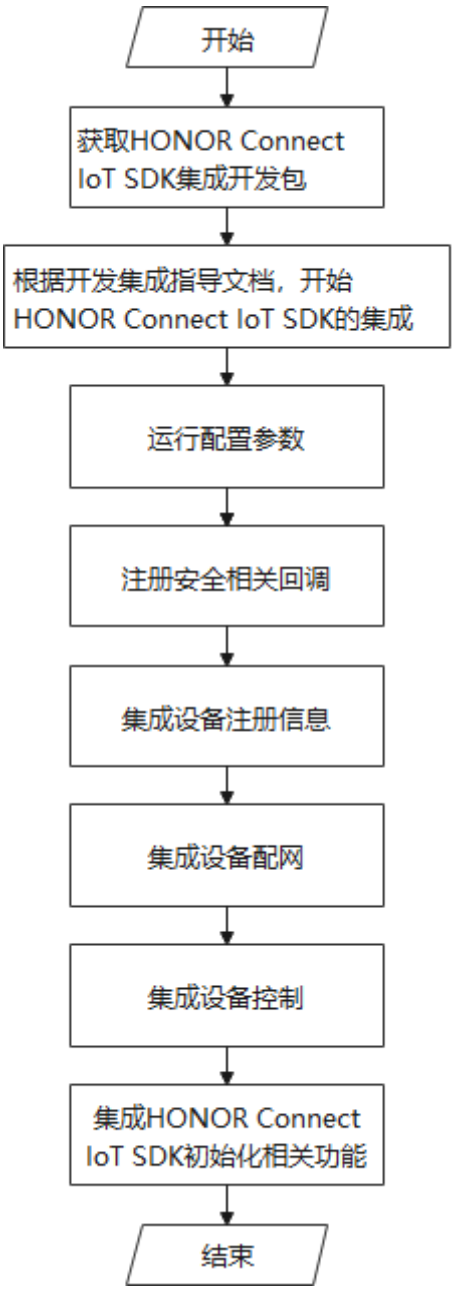
荣耀终端有限公司

目录

1. 概述

HONOR Connect IoT SDK是运行在IOT设备上的SDK，用于实现设备的发现、注册、配网、控制等功能，以及IOT设备间的互联互通。集成HONOR Connect IoT SDK 有助于帮助设备快速安全的接入荣耀MagicLink生态。

本文档用于指导开发者在智能设备中集成和调测HONOR Connect IoT SDK。HONOR Connect IoT SDK的开发集成整体流程如下：



备注：

本集成开发指导对应的HONOR Connect IoT SDK版本号为1.1.0.344，如果您的HONOR Connect IoT SDK开发包为其他版本，请联系荣耀技术支持获取对应版本的集成开发指导。

2. 支持的模组列表和分区表介绍

2.1 支持的模组型号

当前HONOR Connect IoT SDK集成支持的模组型号为：

厂家	WiFi模组型号	芯片平台
博联	BL3383-P	RTL8720CF
博联	BL3357-P	RTL8720CF
爱联	WF-R720F-RTA1	RTL8720CF
爱联	WF-R710-RTA1	RTL8710BX
博联	BL3332-P	RTL8710BN
乐鑫	ESP32-C3-MINI-1	ESP32-C3
小匠	XJ-WB60	BL602
全志	XR872AT	XR872
晶讯	JXC8720-18	RTL8720CM
亮牛	LN882H	LN882H

2.2芯片默认分区表

2.2.1RTL8720CF默认分区表

该分区表为RTL8720CF默认分区表，开发者可根据实际情况合理调整默认分区表，但要确保OTA分区足够，且必须给“HONOR Connect IoT SDK配置信息”分区预留48KB的连续空间，“HONOR Connect IoT SDK配置信息”分区起始地址须按照4KB对齐。

分区	起始地址	分区大小
Partition Table	0x000000	4KB
System Data	0x001000	4KB
Calibration data	0x002000	4KB
Reserved	0x003000	4KB
Boot Image	0x004000	32KB
Firm Ware1	0x00C000	928KB
分区间隔	0x0F4000	16KB
Firm Ware2	0x0F8000	928KB
Reserved	0x1E0000	60KB
HONOR Connect IoT SDK配置信息	0x1EF000	48KB
DCT_BEGIN_ADDR	/	/
UART_SETTING_SECTOR	0x1FB000	4KB
BT_FTL_BKUP_ADDR	0x1FC000	4KB
BT_FTL_PHY_ADDR1	0x1FD000	4KB
BT_FTL_PHY_ADDR0	0x1FE000	4KB
FAST_RECONNECT_DATA	0x1FF000	4KB

2.2.2RTL8720CM默认分区表

该分区表为RTL8720CM默认分区表，开发者可根据实际情况合理调整默认分区表，但要确保OTA分区足够，且必须给“HONOR Connect IoT SDK配置信息”分区预留至少48KB的连续空间，“HONOR Connect IoT SDK配置信息”分区起始地址须按照4KB对齐。

分区	起始地址	分区大小
Partition Table	0x000000	4KB
System Data	0x001000	4KB
Calibration data	0x002000	4KB
Reserved	0x003000	4KB
Boot Image	0x004000	32KB
Firm Ware1	0x00C000	1.5MB
分区间隔	0x18C000	128K
Firm Ware2	0x1AC000	1.5MB
Reserved	0x32C000	764KB
HONOR Connect IoT SDK配置信息	0x3EB000	64KB
UART_SETTING_SECTOR	0x3FB000	4KB
BT_FTL_BKUP_ADDR	0x3FC000	4KB
BT_FTL_PHY_ADDR1	0x3FD000	4KB
BT_FTL_PHY_ADDR0	0x3FE000	4KB
FAST_RECONNECT_DATA	0x3FF000	4KB

2.2.3RTL8710BX和RTL8710BN默认分区表

该分区表为RTL8710BX和RTL8710BN默认分区表，开发者可根据实际情况合理调整默认分区表，但要确保OTA分区足够，且必须给“HONOR Connect IoT SDK配置信息”分区预留48KB的连续空间，“HONOR Connect IoT SDK配置信息”分区起始地址须按照4KB对齐。

分区	起始地址	分区大小
BOOT	0x08000000	32KB
reserved	0x08008000	4KB
System Data	0x08009000	4KB
Calibration data	0x0800A000	4KB
firmWare1	0x0800B000	928KB
分区间隔	0x080F3000	16KB
fireWare2	0x080F7000	928KB
reserved	0x081DF000	68KB
HONOR Connect IoT SDK配置信息	0x081F0000	48KB
rdp_flash_addr	0x081FC000	4KB
AP_SETTING_SECTOR	0x081FD000	4KB
UART_SETTING_SECTOR	0x081FE000	4KB
FAST_RECONNECT_DATA	0x081FF000	4KB

需要特别注意的是RTL8710芯片在调用flash读写接口向flash中写入数据或者读取数据时，相应的地址都需要在上述地址范围的基础上减0x08000000，大致会应用在wifi重连信息的存储和OTA等业务中。

2.2.4ESP32-C3默认分区表

该分区表为ESP32-C3默认分区表，开发者可根据实际情况合理调整默认分区表，但要确保OTA分区足够，且必须给“HONOR Connect IoT SDK配置信息”分区预留48KB的连续空间，“HONOR Connect IoT SDK配置信息”分区起始地址须按照4KB对齐。

分区	起始地址	分区大小
reserved	0x000000	4KB
BootLoader	0x001000	28KB
PartitionTable	0x008000	4KB
nvs	0x009000	16KB
otadata	0x00D000	8KB
phy_init	0x00F000	4KB
ota_0	0x010000	1856KB
ota_1	0x1E0000	1856KB
reserved	0x3B0000	256KB
HONOR Connect IoT SDK配置信息	0x3F0000	48KB
reserved	0x3FC000	16KB

2.2.5BL602默认分区表

该分区表为默认分区表，开发者可根据实际情况合理调整默认分区表，但要确保OTA分区足够，且必须给“HONOR Connect IoT SDK配置信息”分区预留48KB的连续空间，“HONOR Connect IoT SDK配置信息”分区起始地址须按照4KB对齐。

分区	起始地址	分区大小
pt_table(address0)	0xE000	4KB
pt_table(address1)	0xF000	4KB
FW:address0(ota_0)	0x10000	930KB
FW:address1(ota_1)	0xF8800	930KB
mfg	0x1E1000	40KB
KEY	0x1EB000	4KB
DATA(HONOR Connect IoT SDK配置信息)	0x1EC000	48KB
factory	0x1F8000	28KB
rf_para	0x1FF000	4KB

2.2.6XR872默认分区表

该分区表为默认分区表，开发者可根据实际情况合理调整默认分区表，但要确保OTA分区足够，且必须给“HONOR Connect IoT SDK配置信息”分区预留48KB的连续空间，“HONOR Connect IoT SDK配置信息”分区起始地址须按照4KB对齐。

分区	起始地址	分区大小
IMG_1	0	1900KB
IMG_2	0x1DB000	1900KB
USERINFORMATION	0x3B6000	4KB
SSID	0x3B7000	4KB
other	0x3B8000	4KB
UTF8_TO_gbk	0x3B9000	176KB
FACTORY_RESET	0x3E5000	4KB
MAC	0x3E6000	4KB
ALI	0x3E7000	4KB
REBOOT	0x3E8000	4KB
BP	0x3E9000	4KB
FIRST_POWERON	0x3EA000	4KB
MAC_BACKUP	0x3EB000	4KB
BLUETOOTH_VERSIONS	0x3EC000	4KB
空闲(HONOR Connect IoT SDK配置信息)	0x3ED000	72KB
CHECK	0x3FF000	4KB

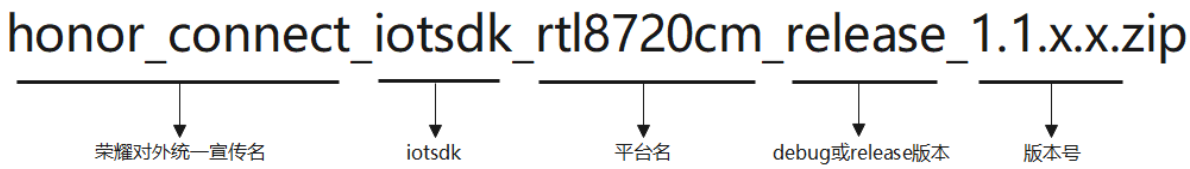
2.2.7LN882H默认分区表

该分区表为默认分区表，开发者可根据实际情况合理调整默认分区表，但要确保OTA分区足够，且必须给“HONOR Connect IoT SDK配置信息”分区预留48KB的连续空间，“HONOR Connect IoT SDK配置信息”分区起始地址须按照4KB对齐。

分区	起始地址	分区大小
BOOT	0	24KB
PART_TAB	0x00006000	4KB
APP	0x00007000	1200KB
OTA	0x00133000	680KB
CONF	0x001DD000	48KB
USER	0x001E9000	64KB
NVDS	0x001F9000	12Kb
KV	0x001FC000	16Kb

3. 开发包结构介绍

HONOR Connect IoT SDK是一套包含头文件、API和动态库在内的开发包，以RTL8720CM平台为例，根据操作系统、编译链获取版本包。版本包格式如下：



SDK包主要是通过操作系统及平台类型进行区分，在集成SDK时仅需要根据相应的操作系统和平台类型选择平台下的demo、include、lib、opensourcelib四个子目录中的文件即可。解压版本包，目录如下（不同平台提供的版本包内容存在差异，以实际版本包为准）：

目录	文件名	说明
├── demo	├── demo.c ├── demo.h ├── demo_profile.h ├── rtl8720cm_sdk.patch	demo main函数所在文件 demo main函数入口声明头文件 HONOR Connect IoT SDK profile头文件 对RTL8720CM原厂sdk的修改
├── include	├── magiclink.h ├── magiclink_netcfg.h ├── magiclink_ota.h ├── magiclink_log.h	MagicLink主接口 设备配网接口 OTA升级接口 日志接口
├── lib	static ├── libbi.a ├── libblenetcfg.a ├── libcoapconnect.a ├── libconnmng.a ├── libcontrol.a ├── libmagiclink.a ├── libnetcfg.a ├── libota.a ├── libp2p.a ├── libregister.a └── liblogcollect.a	bi：商业大数据上报业务库 blenetcfg：ble辅助配网组件库 coapconnect：局域网COAP通信组件 connmng：连接管理模块 control：设备控制库 magiclink：HONOR Connect IoT SDK主库 netcfg：网络参数配置库 ota：ota升级库 p2p：wifi p2p组件库 register：注册库 logcollect：日志收集库
├── opensourcelib	主要包含cjson、mqtt、coap三种开源软件编译生成的静态库	

当前RTL8720CF和RTL8720CM开发包均基于以下RTL8720 SDK版本集成：

名称	版本
SDK	00018588-sdk-ameba-v7.1d(19346)
Patch	7.1d_critical_patch_full_20230620_1d29c079_v(20)(115814)
Patch	7.1d_patch_protect_mac_and_ip_info_230728_1d29c079_v1(119853)

当前RTL8710系列芯片开发包基于以下RTL8710 SDK版本集成：

名称	版本
SDK	00014366-sdk-ameba-v4.0e(7694)
Patch	4.0e_critical_patch_r44443_full_patch_v(08)(61751)
Patch	4.0e_patch_r44443_protect_ip_and_mac_info

当前ESP32-C3系列芯片开发包基于以下ESP-IDF版本集成：

名称	版本
SDK	ESP-IDF v4.4.2

当前BL602系列芯片开发包基于以下BL602 SDK版本集成：

名称	版本
SDK	bouffalolab_release_bl_iot_sdk_1.6.40

当前XR872系列芯片开发包基于以下XR872 SDK版本集成：

名称	版本
SDK	xradio-skylark-sdk

当前LN882H系列芯片开发包基于以下LN882H SDK版本集成：

名称	版本
SDK	v2.1_rc1

说明：开发包中的.c和.h文件中的代码和中英文注释均采用UTF8编码，查看源码时请在相应的IDE调整编码格式为UTF8。

4. 功能集成

本章节介绍了集成HONOR Connect IoT SDK的步骤，依次按照HONOR Connect IoT SDK的基本参数和接口的初始化和注册、基础功能的适配和实现等顺序进行介绍。指导开发者调试设备配网、注册、控制、上报等功能。完成了5章节中的编译和构建过程介绍之后，整体demo即可运行全部功能。本小节将介绍HONOR Connect IoT SDK的基本业务功能集成，集成开发阶段需要注册必要的信息辅助基本业务功能正常运行。

4.1 HONOR Connect IoT SDK初始化

4.1.1 初始化接口

调用API接口MagicLinkInit()来初始化HONOR Connect IoT SDK资源。具体接口定义请参考SDK API接口文档，调用方法参考：

```
if (MagicLinkInit(g_profile, &g_ctrlFunc) != 0) {
    printf("MagicLinkInit fail\r\n");
    vTaskDelete(NULL); //或视情况使用return返回
}
```

说明：

1）HONOR Connect IoT SDK初始化的接口函数声明位于magiclink.h中；

2）上述参考代码中的g_profile指向由设备模型文件（profile）转换成的字符串，具体实现参考HONOR Connect IoT SDK包中的/include/demo_profile.h文件；

3）上述参考代码中的g_ctrlFunc是设备控制回调结构体，其实现参考4.5设备控制回调；

4）在初始化前建议可以选择是否调用4.1.2章节和4.1.3章节中的HONOR Connect IoT SDK属性设置和获取，同时也可根据业务需要选用相应PIN码注册方式，通过4.3章节中的方式向HONOR Connect IoT SDK注册pin码;

5）在初始化HONOR Connect IoT SDK前，必须先调用MbedTlsConfigInit()，完成mbedtls的部分接口适配，其实现参考版本包/demo/demo.c中以下内容：

```
static void* MyCalloc(size_t count, size_t size)
{
    void *ptr = pvPortMalloc(count * size);
    if (ptr) {
        memset(ptr, 0, count * size);
    }

    return ptr;
}

#define MyFree vPortFree

static void MbedTlsConfigInit()
{
    init_rom_ssl_ram_map(MyCalloc, MyFree, NULL, 0);
    init_rom_ssl_hw_crypto_aes_ecb(NULL, NULL, NULL);
    init_rom_ssl_hw_crypto_aes_cbc(NULL, NULL, NULL);
    init_rom_ssl_hw_crypto_des_cbc(NULL, NULL, NULL);
    init_rom_ssl_hw_crypto_3des_cbc(NULL, NULL, NULL);
```

```
}
```

6）在初始化HONOR Connect IoT SDK前，必须先设置HONOR Connect IoT SDK配置信息存储区在Flash中的起始地址，该地址参考2.2芯片默认分区表，地址须按照4KB对齐，代码参考：

```
#define CONFIG_FLASH_BASE_ADDR (0x1EF000) /* 8720片上Flash大小2MB，预留最后48KB+20KB存储SDK数据和厂商数据，其中48KB存储SDK数据 */
```

```
unsigned int cfgBaseAddr = CONFIG_FLASH_BASE_ADDR;
if (MagicLinkSetAttr(MAGICLINK_ATTR_CONFIG_FLASH_BASE_ADDR, &cfgBaseAddr, sizeof(cfgBaseAddr)) != 0) {
    printf("set cfg addr fail\r\n");
    vTaskDelete(NULL); //或视情况使用return返回
}
```

4.1.2 设置HONOR Connect IoT SDK属性

在进行HONOR Connect IoT SDK初始化之前首先应该进行设备属性的设置，设备属性的设置将为下面设备初始化和其他相关功能提供支撑，相关的属性均可选，具体接口定义请参考SDK API接口文档。

- 示例：通过设置MAGICLINK_ATTR_NETCFG_TIMEOUT属性设置配网超时时间。

```
unsigned int timeout = 10;
if (MagicLinkSetAttr(MAGICLINK_ATTR_NETCFG_TIMEOUT, &timeout, sizeof(timeout)) != 0) {
    vTaskDelete(NULL); //或视情况使用return返回
}
```

注意：关于MAGICLINK_ATTR_AUTO_REPORT_MODE参数的使用说明如下：

以如下的light服务为例:

```
{
  "index": 4,
  "id": "light",
  "propertyList": [
    {
      "index": 0,
      "id": "switch",
      "description": "电源开关状态；变化就上报",
    },
    {
      "index": 1,
      "id": "brightness",
      "description": "亮度；变化就上报",
    },
    {
      "index": 2,
      "id": "hue",
      "description": "色相；变化就上报",
    },
    {
      "index": 3,
```

```
      "id": "cct",
      "description": "色温；变化就上报",
    },
    {
      "index": 4,
      "id": "lightMode",
      "valueList": [
        {
          "description": "自定义模式"
        },
        {
          "description": "阅读模式"
        },
        {
          "description": "绘画模式"
        },
        {
          "description": "休闲模式"
        },
        {
          "description": "电子屏模式"
        }
      ]
    }
  ]
}
```

MAGICLINK_ATTR_AUTO_REPORT_MODE设置为0（默认值）时，当用户通过APP修改了light服务下的某个属性，HONOR Connect IoT SDK会将light服务下的所有属性状态上报给云端，例如：用户修改灯光模式为阅读模式时，HONOR Connect IoT SDK会将light服务下{switch，brightness， hue， cct，lightMode}的状态上报给云端。

MAGICLINK_ATTR_AUTO_REPORT_MODE设置为1时，当用户通过APP修改了light服务下的某个属性，HONOR Connect IoT SDK只会上报被修改属性的状态给云端，例如：用户修改灯光模式为阅读模式时，HONOR Connect IoT SDK只会将light服务下lightMode的状态上报给云端。如果修改灯光模式时，light服务下的其他属性会随模式联动修改，需要开发者在修改模式的回调接口中将其他联动修改的属性上报给云端。例如：用户修改灯光模式为阅读模式时，联动修改了{brightness，cct}的状态，则需要开发者在修改模式的回调接口中将{brightness，cct}的状态上报给云端。

4.1.3 获取HONOR Connect IoT SDK属性

HONOR Connect IoT SDK中获取设备属性的接口为MagicLinkGetAttr()，具体接口定义请参考SDK API接口文档。

```
int MagicLinkGetAttr(enum MagicLinkAttr attr, void *buff, unsigned int buffLen,
unsigned int *outLen);
```

4.2 配网

4.2.1 启动配网

调用API接口MagicLinkStartNetCfg()来启动配网，具体接口定义请参考SDK API接口文档，调用方法参考demo中对MagicLinkStartNetCfg()的调用：

```
if (MagicLinkStartNetCfg() != 0) {
    vTaskDelete(NULL); //或视情况使用return返回
}
```

说明：

1）设备未注册时，调用此接口，可实现设备配网，并将设备注册到设备云；

2）设备已注册时，调用此接口，仅实现重新配网功能(重新配网功能需要设备有物理触发机制，例如:按键、组合按键)；

3）在配网失败、设备从云端删除、恢复出厂后这几种场景下，开发者可根据实际情况选择是否调用启动配网接口重新进入设备配网；

4）如设备可以自行连接互联网，可以无需配网。

4.2.2 停止配网

当设备处于配网状态，调用此接口停止配网，具体接口定义请参考SDK API接口文档。

```
int MagicLinkStopNetCfg(void);
```

4.3 注册PIN码获取接口

PIN码（Personal Identification Number）是设备的识别码，HONOR Connect IoT SDK使用PIN码主要用于设备和荣耀智慧空间APP初次连接时，防止误认证，集成方应当在HONOR Connect IoT SDK初始化前即初始化pin码。PIN码是一个长度限定为6字节的字符串，可以使用纯数字字符串，也可以是数字+字母。

HONOR Connect IoT SDK根据设备模型文件中pinMode字段，决定是否使用此接口：如果使用输入PIN码、扫码方式获取PIN码，调用此接口；使用默认PIN码不调用此接口。PIN码的类型在设备模型文件中进行定义，产品根据需要选择PIN码类型，定义之后不可修改。

调用API接口MagicLinkRegGetPin()注册PIN码获取接口，具体接口定义请参考SDK API接口文档。

- 示例：通过MagicLinkRegGetPin()注册PIN码获取接口。

```
if (MagicLinkRegGetPin(&GetPin) != 0) {
    printf("reg pin fail\r\n");
    vTaskDelete(NULL); //或视情况使用return返回
}
```

获取PIN码的回调接口getPin()实现参考：

```
static int GetPin(char *pinBuf, unsigned int pinBufLen)
{
    char *pin = GetDevPin(); //GetDevPin()需开发者实现，用于获取厂家产线预置的PIN
    if (pin == NULL) {
        printf("get pin fail\r\n");
        return 0;
    }
```

```
    (void)memset(pinBuf, 0, pinBufLen);
    (void)memcpy(pinBuf, pin, strlen(pin));

    printf("GEN PIN(%s)\r\n", pinBuf);
    return strlen(pinBuf);
}
```

4.4 设备合法性认证适配

4.4.1 注册硬件根秘钥

在进行HONOR Connect IoT SDK初始化之前首先应该进行硬件根秘钥的注册，硬件根秘钥是HONOR Connect IoT SDK用于产生密钥的根秘钥，开发者根据产品硬件能力，返回硬件根秘钥。硬件根秘钥长度要求为不少于32字节。如需获取硬件根秘钥，建议开发者咨询产品对应芯片厂商。

调用API接口MagicLinkRegHardwareRootKey()注册硬件根秘钥PIN，具体接口定义请参考SDK API接口文档。

- 示例：通过MagicLinkRegHardwareRootKey()注册硬件根密钥。

```
if (MagicLinkRegHardwareRootKey(MyGetHardwareRootKey) != 0) {
    printf("reg hw root key fail\r\n");
    vTaskDelete(NULL); //或视情况使用return返回
}
```

获取硬件根密钥的回调接口MyGetHardwareRootKey()接口实现参考：

```
int MyGetHardwareRootKey(unsigned char *rootKeyBuf, unsigned int rootKeyBufLen)
{
    unsigned char *rootKey = NULL;
    unsigned int rootKeyLen = 0;

    /* GetHwRootKey()需开发者实现，用于获取硬件根秘钥 */
    if (GetHwRootKey(rootKey, &rootKeyLen) != 0) {
        printf("get root key fail\r\n");
        return -1;
    }

    (void)memcpy(rootKeyBuf, rootKey, rootKeyLen);

    return rootKeyLen;
}
```

4.4.2 产品固定秘钥

产品秘钥为荣耀设备云颁发，需要通过产线预置到设备固件中，一款产品一个产品秘钥，产品秘钥主要用于设备登录云认证等方面。

调用API接口MagicLinkRegGetPrdKey()注册产品固定密钥，具体接口定义请参考SDK API接口文档。

- 示例：通过MagicLinkRegGetPrdKey()注册产品密钥。

```
if (MagicLinkRegGetPrdKey(MyGetPrdKey) != 0) {
    vTaskDelete(NULL); //或视情况使用return返回
}
```

获取产品密钥回调函数MyGetPrdKey()实现参考：

```
int MyGetPrdKey(char *key, unsigned int keyBufLen, char *secret, unsigned int
secretBufLen)
{
    unsigned char *prdKey = NULL;
    unsigned int prdKeyLen = 0;
    unsigned char *prdSecret = NULL;
    unsigned int prdSecretLen = 0;

    /* GetPrdKey()需开发者实现，用于获取prdkey */
    if (GetPrdKey(prdKey, &prdKeyLen) != 0) {
        printf("get prdkey fail\r\n");
        return -1;
    }

    /* GetPrdSecret()需开发者实现，用于获取prdsecret */
    if (GetPrdSecret(prdSecret, &prdSecretLen) != 0) {
        printf("get 用于获取prdsecret fail\r\n");
        return -1;
    }

    (void)memcpy(key, prdKey, prdKeyLen);
    (void)memcpy(secret, prdSecret, prdSecretLen);

    return 0;
}
```

4.4.3 软license

License为荣耀设备云颁发，需要通过产线预置到设备中，一台机器一个license，license将主要用于和云端进行认证。

调用API接口MagicLinkRegGetLicense()注册产品密钥，具体接口定义请参考SDK API接口文档。

- 示例：通过MagicLinkRegGetLicense()注册产品密钥。

```
if (MagicLinkRegGetLicense(MyGetLicense) != 0) {
    vTaskDelete(NULL); //或视情况使用return返回
}
```

获取license的回调函数MyGetLicense()实现参考：

```
int MyGetLicense(char *key, unsigned int keyBufLen, char *secret, unsigned int
secretBufLen)
{
    unsigned char *licenseKey = NULL;
    unsigned int licenseKeyLen = 0;
    unsigned char *licenseSecret = NULL;
    unsigned int licenseSecretLen = 0;
```

```
    /* GeLicenseKey()需开发者实现，用于获取license key */
    if (GeLicenseKey(licenseKey, &licenseKeyLen) != 0) {
        printf("get license key fail\r\n");
        return -1;
    }

    /* GetLicenseSecret()需开发者实现，用于获取license secret */
    if (GetLicenseSecret(licenseSecret, &licenseSecretLen) != 0) {
        printf("get license secret fail\r\n");
        return -1;
    }

    (void)memcpy(key, licenseKey, licenseKeyLen);
    (void)memcpy(secret, licenseSecret, licenseSecretLen);

    return 0;
}
```

4.5 设备控制回调

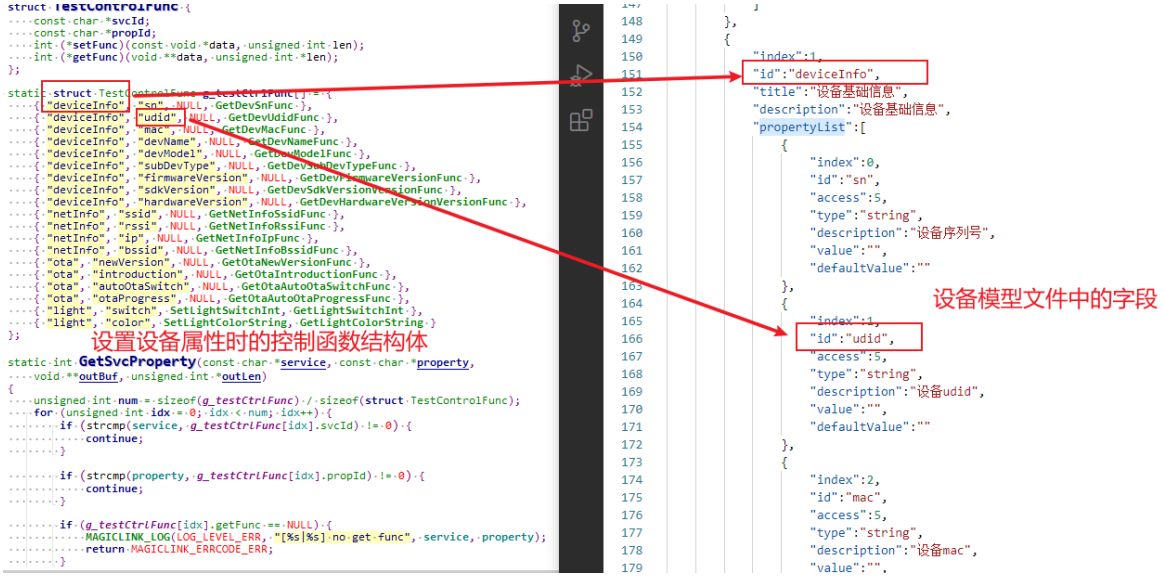
HONOR Connect IoT SDK设备初始化时候的设备控制回调主要作用是将查询设备状态信息、设置设备状态信息，执行控制动作的接口注册到HONOR Connect IoT SDK。该结构体的定义如下：

```
/* 设备控制回调 */
struct MagicLinkCtrlFunc {
    /* 查询设备属性，同步接口
     * 参数out,在接口内部使用malloc分配内存空间，outLen表示分配出的内存空间大小，
     * HONOR Connect IoT SDK调用接口后会自行调用free释放空间
     */
    int (*getServiceProperty)(const char *service, const char *property, void
**out, unsigned int *outLen);

    /* 设置设备属性，返回-100表示正在执行中，执行完成后，通过调用report上报状态信息 */
    int (*setServiceProperty)(const char *service, const char *property, const
void *in, unsigned int inLen);

    /* 执行动作，MagicLink等待动作执行结果 */
    int (*execAction)(const char *service, const char *action, unsigned int
dataNum, const struct MagicLinkDataVal *dataArray);
};
```

集成开发过程中可以参考demo中的注册逻辑对设备控制回调进行注册，可以通过注册回调接口来进行属性的获取或者设置以及相应的执行动作。

设备控制回调的结构体初始化注册时，设备获取设备属性和配置设备属性的数据结构可以参考demo文件。

在getServiceProperty()函数中根据相应的svcId和propId来执行对应的属性配置和获取。svcId对应为设备模型文件中json表格的serviceList对象中的Service的ID，而proId则对应每一个serviceList下面的propertyList中的propertyID。例如，如下图所示为示例代码和设备模型的对应关系示例：



以下是开发者必须实现的控制回调列表：

```
/* deviceInfo netInfo ota 为开发者必须要实现的基础能力 */
static struct TestControlFunc g_testCtrlFunc[] = {
    /* deviceInfo 设备基础信息相关回调 */
    { "deviceInfo", "sn", NULL, GetDevSnFunc },
    { "deviceInfo", "udid", NULL, GetDevUdidFunc },
    { "deviceInfo", "mac", NULL, GetDevMacFunc },
    { "deviceInfo", "devName", SetDevNameFunc, GetDevNameFunc },
    { "deviceInfo", "devModel", NULL, GetDevModelFunc },
    { "deviceInfo", "subDevType", NULL, GetDevSubDevTypeFunc },
    /* 模组固件升级时需要实现 */
    { "deviceInfo", "firmwareVersion", NULL, GetDevFirmwareVersionFunc },
    { "deviceInfo", "hardwareVersion", NULL, GetDevHardwareVersionVersionFunc },
    /* 设备中包含MCU时需要实现 */
    { "deviceInfo", "MCUVersion", NULL, GetMCUVersionFunc },

    /* netInfo 设备网络信息相关回调 */
    { "netInfo", "ssid", NULL, GetNetInfoSsidFunc },
    { "netInfo", "rssi", NULL, GetNetInfoRssiFunc },
    { "netInfo", "ip", NULL, GetNetInfoIpFunc },
    { "netInfo", "bssid", NULL, GetNetInfoBssidFunc },
};
```

- 示例：可以根据deviceInfo和devName来获取或设置设备的devName，相应的示例代码如下所示：

```
/* 设备名称的获取需从持久化保存单元获取， 全局变量g_devName仅作为示例 */
/* 设备重置（RESET）后，设备名的存储单元内容需恢复为默认设备名称 */
static char g_devName[64] = "test_my_DevName";
static int GetDevNameFunc(void **data, unsigned int *len)
{
    unsigned int tmpLen = strlen(g_devName) + 1;
    *len = tmpLen - 1;

    *data = malloc(tmpLen);
    if (*data == NULL) {
        printf("malloc err\r\n");
        return -1;
    }
```

```
    (void)memset(*data, 0, tmpLen);

    (void)strcpy(*data, g_devName);

    return 0;
}

/* 设备名称的更改开发者需要落盘，持久化保存， 全局变量g_devName仅作为示例 */
/* 设备重置（RESET）后，设备名的存储单元内容需恢复为默认设备名称 */
static int SetDevNameFunc(const void *data, unsigned int len)
{
    (void)memset(g_devName, 0, sizeof(g_devName));
    (void)memcpy(g_devName, data, len);

    printf("new devname[%s]\r\n", g_devName);
    return 0;
}
```

最终HONOR Connect IoT SDK中使用的devName是test_my_DevName。

示例：获取产品子型号，相应的示例代码如下所示：

```
/* 获取子产品类型型号，输出data要求为单个字符*/
static int GetDevSubDevTypeFunc(void **data, unsigned int *len)
{
    char *tmp = 获取产线写入的子型号值;
    unsigned int tmpLen = strlen(tmp) + 1;
    *len = tmpLen - 1;

    *data = malloc(tmpLen);
    if (*data == NULL) {
        printf("malloc err\r\n");
        return -1;
    }
    (void)memset(*data, 0, tmpLen);

    (void)strcpy(*data, tmp);

    return 0;
}
```

子型号的定义是：各个子型号之间固件是同一个，只是产品外观颜色有差异。产品子型号的定义值可以是："0" ~ "35"。 当产品有多个子型号时，开发者需要在产线生产时正确预制子型号的值，并通过实现获取产品子型号的接口，将子型号值传递给HONOR Connect IoT SDK。若没有子型号，取默认值"0"。

除了将子型号值传递给HONOR Connect IoT SDK之外，开发者需要为每个子型号分别配置不同的外型，颜色和展示图标。 智慧空间APP会根据该子型号的值，进行子型号区分，展现不同子型号的图标。

有的设备属性接口需要根据实际产品决定，例如台灯类设备profile中有电源开关属性，对应获取和设置的代码实现参考：

```
static int SetLightSwitchInt(const void *data, unsigned int len)
{
```

```
    g_light.on = *(int *)data;
    printf("set light switch succ\r\n");
    return 0;
}

static int GetLightSwitchInt(void **data, unsigned int *len)
{
    *len = sizeof(g_light.on);

    *data = malloc(*len);
    if (*data == NULL) {
        printf("malloc err \r\n");
        return -1;
    }

    (void)memset(*data, 0, *len);
    (void)memcpy(*data, &g_light.on, *len);

    printf("get light switch succ \r\n");
    return 0;
}
```

设备控制函数中的execAction函数是用于根据设备模型文件来向SDK提供执行动作的回调，可以参考如下结构体来定义注册设备执行动作的结构体：

```
struct TestActionFunc {
    const char *svcId;
    const char *actionId;
    int (*action)(const char *inData, char **outData);
};
```

这里的svcId对应于设备模型文件中包含actionList的某一个Sevice ID，actionId对应设备模型文件中的某一个actionList对象，代码参考：

```
static int ExectionLightAction(const char *inData, char **outData)
{
    if (inData == NULL || outData == NULL) {
        printf("invalid param\r\n");
        return -1;
    }

    cJSON *outJson = cJSON_CreateObject();
    if (outJson == NULL) {
        printf(crt out obj\r\n");
        return -1;
    }

    if (cJSON_AddStringToObject(outJson, "actionOutData", "test action out
data") == NULL) {
        printf("add data fail\r\n");
        cJSON_Delete(outJson);
        return -1;
    }
```

```
    *outData = cJSON_Print(outJson);
    cJSON_Delete(outJson);
    return 0;
}

static struct TestActionFunc g_testActionFunc[] = {
    { "light", "lightAction", ExectionLightAction }
};
```

对于事件上报属性，参考代码如下：

```
struct EventData eventData = { 0, strlen("test event msg"), (unsigned char
*)"test event msg" };

if (MagicLinkReportServiceEvent("event", "testEventMsg", 1, &eventData) != 0) {
    printf("rpt event msg fail\r\n");
    return;
}
printf("rpt event msg succ\r\n");
```

4.6 状态上报

4.6.1 服务状态上报

服务状态上报是将设备模型文件中对应的服务的属性状态传递给云端的过程。

通过MagicLinkReportServiceStatus()接口可以一次将服务下的所有属性状态上报给云端，具体接口定义请参考SDK API接口文档。

```
/* 上报HONOR Connect IoT SDK服务状态 */
int MagicLinkReportServiceStatus(const char *service);
```

通过MagicLinkReportPropertyStatus()接口可以将服务下的某个或者某几个属性状态上报给云端，具体接口定义请参考SDK API接口文档。

```
/* 上报HONOR Connect IoT SDK服务属性状态 */
int MagicLinkReportPropertyStatus(const char *service, unsigned int propNum,
const unsigned int *propIndexArray);
```

4.6.2 上报事件

开发者可以在相应的事件产生之后通过MagicLinkReportServiceEvent()接口将相应的事件数据上报给云端，具体接口定义请参考SDK API接口文档。

```
/* 上报HONOR Connect IoT SDK事件状态 */
int MagicLinkReportServiceEvent(const char *service, const char *eventId,
unsigned int dataNum, const struct MagicLinkDataVal *dataArray);

示例：
int testInt = 20;
```

```
char testString[50]="testString";
struct MagicLinkDataVal eventData[2] = {
    { 0, 0, NULL },
    { 0, 0, NULL }
};

eventData[0].dataIdx = 0;
eventData[0].dataLen = strlen(testString);
eventData[0].data = (unsigned char *)testString;

eventData[1].dataIdx = 1;
eventData[1].dataLen = sizeof(testInt);
eventData[1].data = (unsigned char *)&testInt;
MagicLinkReportServiceEvent("event", "audioAlarm", 2, eventData);
```

4.7 HONOR Connect IoT SDK去初始化

设备去初始化主要用于设备在集成HONOR Connect IoT SDK并初始化之后需要主动停止HONOR Connect IoT SDK的情况下需要释放系统的资源，具体接口定义请参考SDK API接口文档。

```
int MagicLinkDeInit(void);
```

4.8 HONOR Connect IoT SDK恢复出厂

当设备需要进行恢复出厂时，HONOR Connect IoT SDK也要进行恢复出厂的操作，清空设备注册结果、注册信息数据和wifi连接信息，使设备可以作为一个全新的设备进行使用，具体接口定义请参考SDK API接口文档。

```
int MagicLinkReset(void);
```

4.9 HONOR Connect IoT SDK状态通知

在HONOR Connect IoT SDK集成的过程中，开发者需要为设备程序主动获取或者被动获取MagicLink的状态，以便根据相应状态进行不同的操作，具体接口定义请参考SDK API接口文档。

OTA版本包下载完成以后，HONOR Connect SDK会通过设备状态接收回调接口给开发者通知STATUS_OTA_DOWNLOAD_COMPLETE状态，并在回调接口返回之后执行重启动作。开发者需要在收到STATUS_OTA_DOWNLOAD_COMPLETE状态通知时同步执行完成配置保存等收尾工作。

- 示例：参考以下方法注册HONOR Connect IoT SDK状态回调：

```
if (MagicLinkRegRecvStatus(MyRecvStatus) != 0) {
    vTaskDelete(NULL); //或视情况使用return返回
}
```

其中，MyRecvStatus为接收设备状态的回调，可以参考如下示例代码进行设备状态接收回调函数的编写：

```
void MyRecvStatus(enum MagicLinkSDKStatus status)
{
    printf("current status is %d", status); /* 当前SDK的状态 */
    switch (status) {
        case STATUS_NETCFG_ENTER:
            /* 当进入到配网设置时执行相关逻辑 */
```

```
            break;
        case STATUS_REGISTER_RECV_INFO:
            /* 当进行到设备注册接收信息执行相关逻辑 */
            break;
        default:
            return;
    }
    return;
}
```

4.10 OTA 升级开发

OTA升级时，固件版本号对比算法是基于数字大小进行比较，版本号可由字母，数字或者.组成；如固件版本号带字母，则字母必须位于版本号最开头，例如v1000或者ver1.1.0.0；固件版本头、尾、中间均不能有空格；在产品的整个生命周期固件版本号格式需要保持一致，且新分配固件版本号向上递增，为了更好的在荣耀智慧空间显示，建议模组固件和MCU固件的版本号为点分格式（例如：1.0.0），且长度不超过31个字符。

4.10.1 模组固件升级

HONOR Connect IoT SDK已经实现模组固件升级的相关功能，开发者无需开发模组固件升级功能。如无需实现MCU的固件升级功能，则参考4.11章节制作固件升级包即可。

4.10.2 MCU固件升级

开发者如需实现MCU的固件升级，需要注册MCU升级接口，并在MCU升级成功和升级过程中发生异常时通知HONOR Connect IoT SDK。

1）注册MCU升级接口

开发者实现开始MCU升级、发送MCU版本包、通知MCU版本包发送完成、通知MCU升级过程异常的接口，并通过MagicLinkRegMcuOtaFunc注册到HONOR Connect IoT SDK，HONOR Connect IoT SDK在MCU升级过程中调用对应接口。具体接口定义请参考SDK API接口文档，注册方法参考：

```
const struct MagicLinkMcuOtaFunc mcuOtaFunc = {
    .mcuOtaStart = McuOtaStart,
    .mcuOtaSendPkg = McuOtaSendPkg,
    .mcuOtaEnd = McuOtaEnd,
    .notifyMcuOtaException = NotifyMcuOtaException,
};

if (MagicLinkRegMcuOtaFunc(&mcuOtaFunc) != 0) {
    return -1;
}
```

说明：注册MCU升级接口的函数声明位于magiclink_ota.h中；注册MCU升级接口需要在执行MCU升级动作之前完成。

2）MCU通知HONOR Connect IoT SDK升级状态

MCU升级成功和升级过程中发生异常时，需要调用MagicLinkSetMcuOtaStatus接口通知HONOR Connect IoT SDK。HONOR Connect IoT SDK收到异常通知后停止升级过程。具体接口定义请参考SDK API接口文档，调用方法参考：

```
if (MagicLinkSetMcuOtaStatus(0, "MCU upgrade success") != 0) {
    return -1;
}
```

4.11 OTA升级版本包制作

1）确认新版本号，使用新版本号编译生成新版本升级文件，升级文件不包含boot和分区表部分。

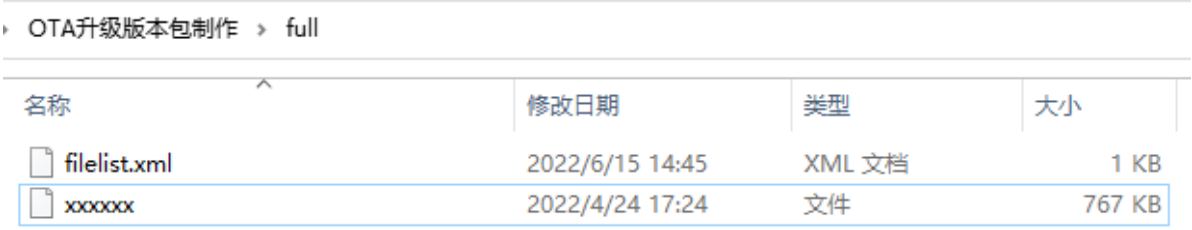
2）新建新版本描述文件filelist.xml，文件中写入以下内容，保持xml格式不变。

```
<?xml version="1.0" encoding="utf-8"?>
<root>
<file>
<spath>update_ota.bin</spath>
<sha256>8EB9A1620BAA3257357B4E38F32A29D2A6980A42AD89E984765C035C08F55D42</sha256
>
<size>3616840</size>
</file>
</root>
```

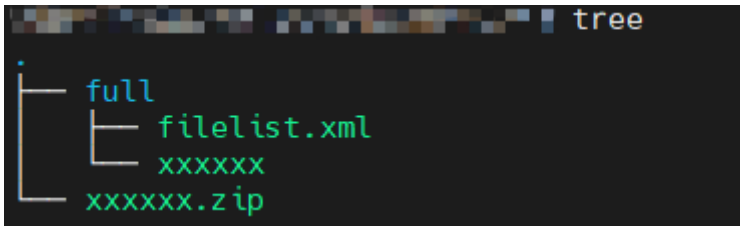
3）修改新版本描述文件filelist.xml内容，仅需要修改第四行到第六行的内容：

- spath字段填入待升级的新固件文件名称，必须包含文件后缀。
- sha256字段填入待升级的新固件文件的SHA256校验值。
- size字段写入待升级的新固件文件大小，单位为字节。

4）新建一个文件夹命名为full，把待升级的新固件文件和新版本描述文件filelist.xml放入full文件夹中。



5）以zip格式打包压缩full文件夹，得到版本包文件。



各模组版本包制作除了遵循上述格式要求之外的差异如下表所示：

序号	模组名称
1	RTL8710系列模组

4.12 循环升级测试

循环升级测试仅限于测试阶段，用于测试OTA升级的稳定性。商用版本的OTA升级不可配置循环升级。

以v1.0.0.3和v1.0.0.4版本之间的循环升级为例，配置步骤如下：

1）按版本包制作规则，制作固件版本号为v1.0.0.3和v1.0.0.4的两个OTA版本包；

2）将v1.0.0.3版本的固件烧录到设备中；

3）部署v1.0.0.3版本包到荣耀OTA服务器时，勾选支持循环升级，并配置循环升级路径为v1.0.0.3到v1.0.0.4；

4）部署v1.0.0.4版本包到荣耀OTA服务器时，勾选支持循环升级，并配置循环升级路径为v1.0.0.4到v1.0.0.3；

5）进行循环升级测试，此时可在v1.0.0.3和v1.0.0.4两个版本间反复循环升级。

5.编译和链接

5.1 RTL8720CF平台的编译和链接

编译建议环境：64bit x86-64 Linux操作系统。

在不增加自有代码的前提下，可以使用RTL8720CF SDK的示例工程目录编译运行第一个demo（以下操作请在同一路径下进行，步骤中版本包名称以实际版本为准）：

1、解压RTL8720CF原厂SDK

```
unzip -n -qO UTF-8 'OO018588-sdk-ameba-v7.1d(19346)'
```

2、解压RTL8720CF原厂SDK Patch

```
unzip -n -qO UTF-8 '7.1d_critical_patch_full_20230620_1d29c079_v(20)(115814)'
unzip -n -qO UTF-8
'7.1d_patch_protect_mac_and_ip_info_230728_1d29c079_v1(119853)'
```

3、安装RTL8720CF原厂SDK Patch，该patch直接复制并覆盖RTL8720CF SDK相同路径即可

```
cp -r '7.1d_critical_patch_full_20230620_1d29c079_v(20)'/* sdk-ameba-v7.1d
cp -r '7.1d_patch_protect_mac_and_ip_info_230728_1d29c079_v1'/* sdk-ameba-v7.1d
```

4、同一路径下，解压HONOR Connect IoT SDK产品开发包

```
unzip honor_connect_iotsdk_rtl8720cf_release_1.1.x.x.zip
```

5、安装产品开发包中的rtl8720cf_sdk.patch，此patch是HONOR Connect IoT SDK需要对原厂SDK做的一些修改，安装命令示例：

```
patch -d ./ -p0 < demo/rtl8720cf_v20_sdk.patch
```

6、开发者根据实际情况修改产品开发包中`demo/demo_profile.h`和`demo/demo.c`，然后拷贝demo相关文件到RTL8720CF SDK工程目录下

```
cp -r demo/demo.h demo/demo_profile.h sdk-ameba-
v7.1d/project/realtek_amebaz2_v0_example/inc/
cp -r demo/demo.c sdk-ameba-v7.1d/project/realtek_amebaz2_v0_example/src/
```

7、进入RTL8720CF SDK编译脚本所在路径，执行编译指令

```
cd sdk-ameba-v7.1d/project/realtek_amebaz2_v0_example/GCC-RELEASE
make clean && make all
```

8、将编译出的版本文件`sdk-ameba-v7.1d/project/realtek_amebaz2_v0_example/GCC-RELEASE/application_is/Debug/bin/flash_is.bin`烧录到设备并运行。

以上为整体编译链接的介绍，集成开发时可以自行修改demo和相关编译脚本，可以增加自己需要的文件或者引入自己工程的路径参与整体编译。

5.2 RTL8710平台的编译和链接

编译建议环境：64bit x86-64 Linux操作系统。

在不增加自有代码的前提下，可以使用RTL8710 SDK的示例工程目录编译运行第一个demo（以下操作请在同一路径下进行，步骤中版本包名称以实际版本为准）：

1、解压RTL8710原厂SDK

```
unzip -n -qO UTF-8 'OOO14366-sdk-ameba-v4.0e(7694).zip'
```

2、解压RTL8710原厂SDK Patch

```
unzip -n -qO UTF-8 '4.0e_critical_patch_r44443_full_patch_v(08)(61751).zip'
unzip -n -qO UTF-8 '4.0e_patch_r44443_protect_ip_and_mac_info.zip'
```

3、安装RTL8710原厂SDK Patch，该patch直接复制并覆盖RTL8710 SDK相同路径即可

```
cp -rf ./'4.0e_critical_patch_r44443_full_patch_v(08)'/* ./sdk-ameba-v4.0e/
cp -rf ./'4.0e_patch_r44443_protect_ip_and_mac_info'/* ./sdk-ameba-v4.0e/
```

4、同一路径下，解压HONOR Connect IoT SDK产品开发包

```
unzip  honor_connect_iotsdk_rtl8710_release_1.1.x.x.zip
```

5、安装产品开发包中的rtl8710_sdk.patch，此patch是HONOR Connect IoT SDK需要对原厂SDK做的一些修改，安装命令示例：

```
patch -d ./ -p0 < demo/rtl8710_sdk.patch
```

6、开发者根据实际情况修改产品开发包中`demo/demo_profile.h`和`demo/demo.c`，然后拷贝demo相关文件到RTL8710 SDK工程目录下

```
cp -rf ./demo/demo.h ./demo/demo_profile.h ./sdk-ameba-
v4.0e/project/realtek_amebaz_va0_example/inc/
cp -rf ./demo/demo.c ./sdk-ameba-v4.0e/project/realtek_amebaz_va0_example/src/
```

7、进入RTL8710 SDK编译脚本所在路径，执行编译指令

```
cd ./sdk-ameba-v4.0e/project/realtek_amebaz_va0_example/GCC-RELEASE/
make clean && make all
```

8、将编译出的版本文件`sdk-ameba-v4.0e/project/realtek_amebaz_va0_example/GCC-RELEASE/application/Debug/bin/image2_all_ota1.bin`烧录到设备并运行。

9、特别说明，针对8中，RTL8710SDK默认编译OTA1分区分固件，如需编译OTA2分区固件请参考附录7.1 RTL8710模组 OTA适配及注意事项。

以上为整体编译链接的介绍，集成开发时可以自行修改demo和相关编译脚本，可以增加自己需要的文件或者引入自己工程的路径参与整体编译。

5.3 ESP32-C3平台的编译和链接

5.3.1 Linux环境

编译建议环境：64bit x86-64 Linux操作系统。

在不增加自有代码的前提下，可以使用HONOR Connect IoT SDK产品开发包中的示例工程编译运行第一个demo（以下操作请在同一路径下进行，步骤中版本包名称以实际版本为准）：

1、安装ESP-IDF v4.4.2版本，并确认环境变量IDF_PATH已经正确指向ESP-IDF所在的完整安装路径，且未对ESP-IDF v4.4.2做任何修改；

2、解压HONOR Connect IoT SDK产品开发包：

```
unzip honor_connect_iotsdk_esp32c3_release_1.1.x.x.zip
```

3、安装产品开发包中的esp32idf.patch，此patch是HONOR Connect IoT SDK需要对原厂ESP-IDF做的一些修改，安装命令示例：

```
patch -d ${IDF_PATH}/../ -p0 < demo/esp32idf.patch
```

4、开发者根据实际情况修改产品开发包中`demo/main/demo_profile.h`和`demo/main/demo.c`；

5、执行如下指令进行编译：

```
cd demo/
mkdir build && cd build
export IDF_TARGET=esp32c3
cmake -DPYTHON_DEPS_CHECKED=1 -DESP_PLATFORM=1 -DCCACHE_ENABLE=1 ../
make clean && make -j4
```

6、在官网下载Flash工具，将编译出的版本文件`demo/build/bootloader/bootloader.bin`，`demo/build/partition_table/partition-table.bin`，`demo/build/demo.bin`烧录到设备并运行：

https://www.espressif.com.cn/zh-hans/support/download/other-tools

7、特别说明，如需修改工程的sdkconfig配置，请确保如下配置项保持不变，否则会引起HONOR Connect IoT SDK功能异常：

```
CONFIG_FREERTOS_UNICORE=y
CONFIG_FREERTOS_HZ=100
CONFIG_FREERTOS_MAX_TASK_NAME_LEN=16
CONFIG_LWIP_TCP_OVERSIZE_MSS=y
# CONFIG_LWIP_IP4_REASSEMBLY is not set
# CONFIG_LWIP_IPV6 is not set
```

5.3.2 Windows环境

编译建议环境：64位Windows操作系统。

在不增加自有代码的前提下，可以使用HONOR Connect IoT SDK产品开发包中的示例工程编译运行第一个demo（以下操作步骤中版本包名称以实际版本为准）：

1、参考ESP-IDF官网指导，在Windwos平台安装ESP-IDF v4.4.2版本：
https://docs.espressif.com/projects/esp-idf/zh_CN/stable/esp32/get-started/windows-setup.html

2、安装Git For Windows：
https://gitforwindows.org/

3、打开Git Bash，跳转到HONOR Connect IoT SDK产品开发包所在目录，解压开发包：

```
unzip honor_connect_iotsdk_esp32c3_release_1.1.x.x.zip
```

4、Git Bash中，仍在HONOR Connect IoT SDK产品开发包所在目录，安装esp32idf.patch。此patch是HONOR Connect IoT SDK需要对原厂ESP-IDF做的一些修改，安装命令示例如下（其中`/c/Espressif/frameworks/`是Windows上esp-idf-v4.4.2默认路径，使用时请根据实际情况替换）：

```
patch -d /c/Espressif/frameworks/ -p0 < demo/esp32idf.patch
```

建议将`Espressif/frameworks/esp-idf-v4.4.2`原始文件备份，每次打patch前请确认esp-idf-v4.4.2未被修改；

5、开发者根据实际情况修改产品开发包中`demo/main/demo_profile.h`和`demo/main/demo.c`；

6、打开ESP-IDF 4.4 PowerShell，跳转到开发包中demo文件所在目录，执行编译指令：

```
cd demo/
idf.py build
```

在对代码进行修改后，需要再次编译前，建议先执行如下指令清除build目录内容，再进行编译：

```
idf.py fullclean
```

7、编译结束后，ESP-IDF 4.4 PowerShell中仍在demo目录下，执行烧录指令（请将PORT替换为ESP32-C3开发板的串口名称COMX）：

```
idf.py -p PORT flash
```

也可在官网下载Flash工具，将编译出的版本文件`demo/build/bootloader/bootloader.bin`，`demo/build/partition_table/partition-table.bin`，`demo/build/demo.bin`烧录到设备并运行：
https://www.espressif.com.cn/zh-hans/support/download/other-tools

8、特别说明，如需修改工程的sdkconfig配置，请确保如下配置项保持不变，否则会引起HONOR Connect IoT SDK功能异常：

```
CONFIG_FREERTOS_UNICORE=y
CONFIG_FREERTOS_HZ=100
CONFIG_FREERTOS_MAX_TASK_NAME_LEN=16
CONFIG_LWIP_TCP_OVERSIZE_MSS=y
# CONFIG_LWIP_IP4_REASSEMBLY is not set
# CONFIG_LWIP_IPV6 is not set
```

以上为整体编译链接的介绍，集成开发时可以自行修改demo和相关编译脚本，可以增加自己需要的文件或者引入自己工程的路径参与整体编译。

5.4 BL602平台的编译和链接

5.4.1 Linux环境

编译建议环境：64bit x86-64 Linux操作系统。

在不增加自有代码的前提下，可以使用HONOR Connect IoT SDK产品开发包中的示例工程编译运行第一个demo（以下操作请在同一路径下进行，步骤中版本包名称以实际版本为准）：

1、解压原厂sdk，并确认未对bl_iot_sdk做任何修改；

```
unzip bl_iot_sdk-master.zip
```

2、解压HONOR Connect IoT SDK产品开发包：

```
unzip honor_connect_iotsdk_bl602_release_1.1.x.x.zip
```

3、同一路径下，安装产品开发包中的bl602idf.patch，此patch是HONOR Connect IoT SDK需要对原厂bl_iot_sdk做的一些修改，安装命令示例：

```
patch -d ./ -p0 < demo/bl602idf.patch
chmod 766 -R bl_iot_sdk-master
```

4、开发者根据实际情况修改产品开发包中`demo/demo_profile.h`和`demo/demo.c`,然后拷贝到bl_iot_sdk工程目录下；

```
cp -r demo/demo.c demo/demo_profile.h bl_iot_sdk-
master/bl_iot_sdk/customer_app/bl602_demo_event/bl602_demo_event/
```

5、执行如下指令进行编译：

```
cd bl_iot_sdk-master/bl_iot_sdk/customer_app/bl602_demo_event
rm -rf build_out/
./genromap
```

6、在官网下载Flash工具，将编译出的版本文件`bl_iot_sdk-master/bl_iot_sdk/customer_app/bl602_demo_event/build_out/bl602_demo_event.bin`以及Flash工具中的blsp_boot2_release.bin烧录到设备并运行。

以上为整体编译链接的介绍，集成开发时可以自行修改demo和相关编译脚本，可以增加自己需要的文件或者引入自己工程的路径参与整体编译。

5.5 XR872平台的编译和链接

编译建议环境：64bit x86-64 Linux操作系统。

在不增加自有代码的前提下，可以使用HONOR Connect IoT SDK产品开发包中的示例工程编译运行第一个demo（以下操作请在同一路径下进行，步骤中版本包名称以实际版本为准）：

1、解压XR872原厂SDK

```
unzip xradio-skylark-sdk.zip
```

2、同一路径下，解压XR872使用的交叉编译链

```
tar -jxvf gcc-arm-none-eabi-4_9-2015q2-20150609-linux.tar.bz2
chmod 766 -R gcc-arm-none-eabi-4_9-2015q2
```

3、同一路径下，解压HONOR Connect IoT SDK产品开发包

```
unzip  honor_connect_iotsdk_xr872_debug_1.1.x.x.zip
```

4、安装产品开发包中的xr872_sdk.patch，此patch是HONOR Connect IoT SDK需要对原厂SDK做的一些修改，安装命令示例：

```
patch -d ./ -p0 < demo/xr872_sdk.patch
chmod 766 -R xradio-skylark-sdk
```

5、开发者根据实际情况修改产品开发包中`demo/demo_profile.h`和`demo/demo.c`，然后拷贝demo相关文件到XR872 SDK工程目录下

```
cp -rf ./demo/demo.h ./demo/demo_profile.h ./demo/demo.c ./xradio-skylark-
sdk/project/demo/hello_demo
```

6、进入XR872 SDK编译脚本所在路径，执行编译指令

```
cd ./xradio-skylark-sdk/project/demo/hello_demo/gcc
make build_clean && make build
```

7、将编译出的版本文件 xradio-skylark-sdk/project/demo/hello_demo/image/xr872/xr_system.img烧录到设备并运行。

以上为整体编译链接的介绍，集成开发时可以自行修改demo和相关编译脚本，可以增加自己需要的文件或者引入自己工程的路径参与整体编译。

5.6 RTL8720CM平台的编译和链接

编译建议环境：64bit x86-64 Linux操作系统。

在不增加自有代码的前提下，可以使用RTL8720CM SDK的示例工程目录编译运行第一个demo（以下操作请在同一路径下进行，步骤中版本包名称以实际版本为准）：

1、解压RTL8720CM原厂SDK

```
unzip -n -qO UTF-8 'OOO18588-sdk-ameba-v7.1d(19346)'
```

2、解压RTL8720CM原厂SDK Patch

```
unzip -n -qO UTF-8 '7.1d_critical_patch_full_20230620_1d29c079_v(20)(115814)'
unzip -n -qO UTF-8
'7.1d_patch_protect_mac_and_ip_info_230728_1d29c079_v1(119853)'
```

3、安装RTL8720CM原厂SDK Patch，该patch直接复制并覆盖RTL8720CM SDK相同路径即可

```
cp -r '7.1d_critical_patch_full_20230620_1d29c079_v(20)'/* sdk-ameba-v7.1d
cp -r '7.1d_patch_protect_mac_and_ip_info_230728_1d29c079_v1'/* sdk-ameba-v7.1d
```

4、同一路径下，解压HONOR Connect IoT SDK产品开发包

```
unzip honor_connect_iotsdk_rtl8720cm_release_1.1.x.x.zip
```

5、安装产品开发包中的rtl8720cm_sdk.patch，此patch是HONOR Connect IoT SDK需要对原厂SDK做的一些修改，安装命令示例：

```
patch -d ./ -p0 < demo/rtl8720cm_v20_sdk
```

6、开发者根据实际情况修改产品开发包中`demo/demo_profile.h`和`demo/demo.c`，然后拷贝demo相关文件到RTL8720CM SDK工程目录下

```
cp -r demo/demo.h demo/demo_profile.h sdk-ameba-
v7.1d/project/realtek_amebaz2_v0_example/inc/
cp -r demo/demo.c sdk-ameba-v7.1d/project/realtek_amebaz2_v0_example/src/
```

7、进入RTL8720CM SDK编译脚本所在路径，执行编译指令

```
cd sdk-ameba-v7.1d/project/realtek_amebaz2_v0_example/GCC-RELEASE
make clean && make all
```

8、将编译出的版本文件`sdk-ameba-v7.1d/project/realtek_amebaz2_v0_example/GCC-RELEASE/application_is/Debug/bin/flash_is.bin`烧录到设备并运行。

以上为整体编译链接的介绍，集成开发时可以自行修改demo和相关编译脚本，可以增加自己需要的文件或者引入自己工程的路径参与整体编译。

5.7 LN882H平台的编译和链接

5.4.1 Linux环境

编译建议环境：64bit x86-64 Linux操作系统。

在不增加自有代码的前提下，可以使用HONOR Connect IoT SDK产品开发包中的示例工程编译运行第一个demo（以下操作请在同一路径下进行，步骤中版本包名称以实际版本为准）：

1、解压ln882h使用的交叉编译链,导出环境变量；

```
tar -xjf gcc-arm-none-eabi-10-2020-q4-major-x86_64-linux.tar.bz2
export CROSS_TOOLCHAIN_ROOT=./gcc-arm-none-eabi-10-2020-q4-major
source ~/.bashrc
```

2、同一路径下，解压原厂sdk，并确认未对ln882h-custom-imou做任何修改；

```
unzip ln882h-custom-imou.zip
```

2、解压HONOR Connect IoT SDK产品开发包：

```
unzip honor_connect_iotsdk_ln882h_release_1.1.x.x.zip
```

3、同一路径下，安装产品开发包中的ln882h_sdk.patch，此patch是HONOR Connect IoT SDK需要对原厂ln882h-custom-imou做的一些修改，安装命令示例：

```
patch -d ./ -p0 < demo/ln882h_sdk.patch
chmod 766 -R ln882h-custom-imou
```

4、开发者根据实际情况修改产品开发包中`demo/demo_profile.h`和`demo/demo.c`,然后拷贝到ln882h-custom-imou工程目录下；

```
cp -r demo/demo.c demo/demo.h demo/demo_profile.h ln882h-custom-
imou/project/combo_mcu_basic_example/app
```

5、执行如下指令进行编译：

```
cd ln882h-custom-imou
python3  start_build.py rebuild
```

6、在官网下载Flash工具，将编译出的版本文件`combo_mcu_basic_example/build-combo_mcu_basic_example-release/bin/flashimage.bin`烧录到设备并运行。

以上为整体编译链接的介绍，集成开发时可以自行修改demo和相关编译脚本，可以增加自己需要的文件或者引入自己工程的路径参与整体编译。

5.8 HONOR Connect IoT SDK组件功能介绍

SDK组件化能力，厂商可以根据设备实际功能需求选择对应的功能组件进行集成，解压版本包后，必选集成组件如下：

库名	文件名	说明
HONOR Connect IoT SDK主库	libmagiclink.a	SDK主库，必须集成
局域网COAP通信组件	libcoapconnect.a	用于COAP近场通信，必须集成
设备控制组件	libcontrol.a	SDK提供的设备控制功能，必须集成
配网组件	libnetcfg.a	SDK提供的WIFI配网功能，必须集成
注册组件	libregister.a	SDK提供的注册上云功能，必须集成
开源组件	主要包含cjson、mbedTLS、mqtt、coap	必须集成

解压版本包后，可选集成组件如下：

库名	文件名	说明
ota升级组件	libota.a	需要使用SDK提供的荣耀OTA升级功能的设备选择集成
蓝牙辅助配网组件	libblenetcfg.a	需要使用蓝牙进行配网的设备选择集成

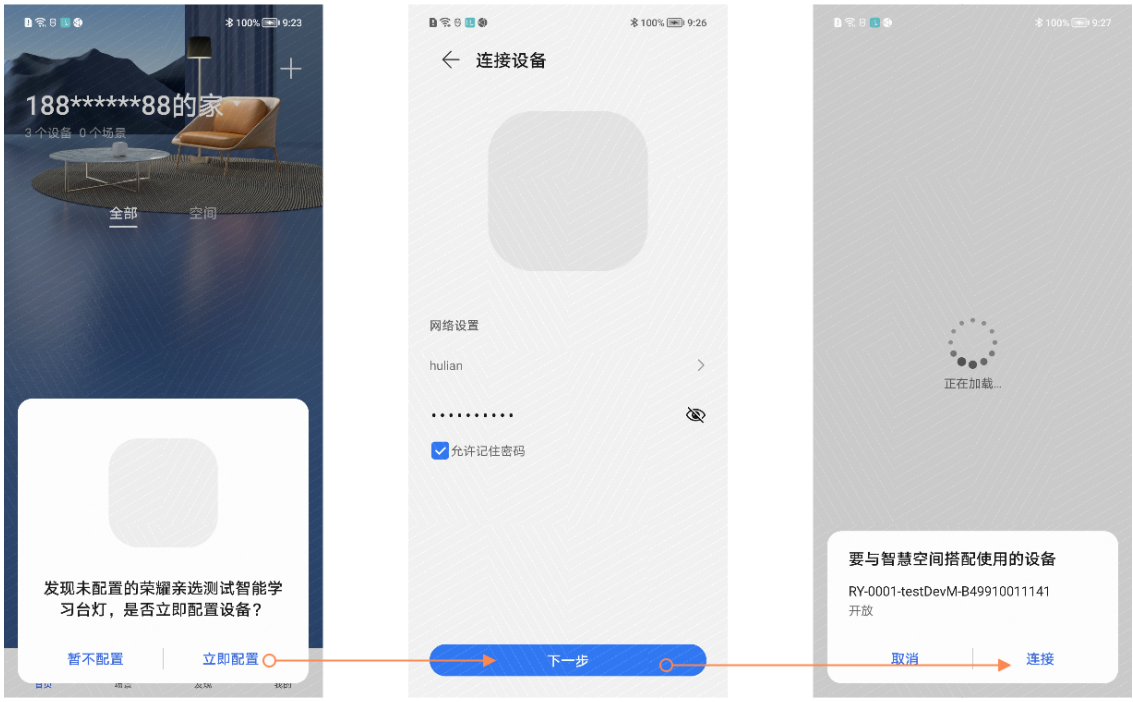
6. 功能验证

6.1 概述

根据产品定义的功能，验证业务功能，使用配套的荣耀智慧空间APP实现设备对应的添加、注册、控制功能，与设备共同验证。

6.2 APP调试环境设置

1. 运行荣耀智慧空间APP，同意相关权限。
2. 按照如下步骤操作：




连接设备
请扫描设备标签上带有PIN码的二维码
手动输入


连接设备
操作指导
在设备上找到图示标签
输入设备标签上的6位PIN码或扫描二维码
输入设备标签上的6位PIN码


连接设备
操作指导
在设备上找到图示标签
输入设备标签上的6位PIN码或扫描二维码
123456
确定


连接设备
3%
设备联网中
请确保手机靠近设备，且无线网络通畅


连接设备
连接成功


设备设置
家庭
188******88的家
设备名称
RY-A001-testDevM-B5011001114
1
所属房间
添加房间
客厅
卧室
次卧
主卧
书房
客房
餐厅
厨房
浴室
卫生间
玄关
走廊
天台
阳台
默认房间
完成


欧普智能台灯
已关闭

7. 附录

7.1 RTL8710模组 OTA适配及注意事项

7.1.1 RTL8710模组OTA升级的注意事项

（1）RTL8710模组出厂时必须烧录system.bin文件，system.bin需要使用ImageTool.exe工具进行手动生成。生成步骤如下图所示：

其中选择的分区必须为当前需要启动分区，OTA2分区启动地址与下文中OTA编译相关中的地址保持一致。

（2）RTL8710的升级固件有两个：image2_all_ota1.bin和image2_all_ota2.bin，分别对应OTA1和OTA2分区，在进行OTA升级业务前需要提前准备好这两个固件，固件的编译生成方法将在下文中阐述，这两个固件的名称不可更改。

7.1.2 RTL8710模组OTA功能的开发

1. 确认分区地址，即OTA1、OTA2的地址划分，根据所确定的分区地址，需要分别修改如下文件：

（1）rlx8711B-symbol-v02-img2_xip1.ld和rlx8711B-symbol-v02-img2_xip2.ld中的内容均按照下图所示修改，其中XIP1和XIP2分别对应OTA1和OTA2两个分区：

```
ENTRY(Reset_Handler)

INCLUDE "export-rom_symbol_v01.txt"

MEMORY
{
        ROM (rx) :                  ORIGIN = 0x00000000, LENGTH = 0x80000          /* ROM: 512k */
        ROMBSS_RAM (rw) :           ORIGIN = 0x10000000, LENGTH = 0x2000 /* ROM BSS RAM: 8K */
        BOOTLOADER_RAM (rwx) :  ORIGIN = 0x10002000, LENGTH = 0x3000 /* BOOT Loader RAM: 12K */
        BD_RAM (rwx) :          ORIGIN = 0x10005000, LENGTH = 0x39000          /* MAIN RAM: 228 */
        MSP_RAM (wx) :          ORIGIN = 0x1003E000, LENGTH = 0x1000 /* MSP RAM: 4k */
        RDP_RAM (wx) :          ORIGIN = 0x1003F000, LENGTH = 0xFF0  /* RDP RAM: 4k-0x10 */

        XIPBOOT (rx) :              ORIGIN = 0x08000000+0x20, LENGTH = 0x04000-0x20        /* XIPBOOT: 32k, 32 Bytes resvd for header*/
        XIPSYS (r) :                ORIGIN = 0x08009000, LENGTH = 0x1000 /* XIPSYS: 4K system data in flash */
        XIPCAL (r) :                ORIGIN = 0x0800A000, LENGTH = 0x1000 /* XIPCAL: 4K calibration data in flash */
        XIP1 (rx) :                 ORIGIN = 0x0800B000+0x20, LENGTH = 0xE8000-0x20       /* XIP1: 928KB, 32 Bytes resvd for header */
        XIP2 (rx) :                 ORIGIN = 0x080F7000+0x20, LENGTH = 0xE8000-0x20       /* XIP2: 928KB, 32 Bytes resvd for header */
}
```


（2）application.mk文件，需要在文件头部对ota_idx赋值，当赋值为1时会生成OTA1分区固件，当赋值为2时会生成OTA2分区固件，如下图所示：

```
CROSS_COMPILE = $(ARM_GCC_TOOLCHAIN)/arm-none-eabi-

#ota zone chose ,you can chose ota1 or ota2 by chose only one to uncommented in next 2 lines
# ota_idx = 2
ota_idx = 1
```


同时必须删除该文件如下位置所示的一行代码：

```
ifeq ("${ota_idx}", "1")
        cat $(BIN_DIR)/xip_image2.p.bin > $(BIN_DIR)/$(IMAGE2_OTA1)
        chmod 777 $(BIN_DIR)/$(IMAGE2_OTA1)
        cat $(BIN_DIR)/ram_2.p.bin >> $(BIN_DIR)/$(IMAGE2_OTA1)
        $(CHKSUM) $(BIN_DIR)/$(IMAGE2_OTA1) || true
        #rm $(BIN_DIR)/xip_image2.p.bin $(BIN_DIR)/ram_2.p.bin
else ifeq ("${ota_idx}", "2")
        cat $(BIN_DIR)/xip_image2.p.bin > $(BIN_DIR)/$(IMAGE2_OTA2)
        chmod 777 $(BIN_DIR)/$(IMAGE2_OTA2)
        cat $(BIN_DIR)/ram_2.p.bin >> $(BIN_DIR)/$(IMAGE2_OTA2)
        $(CHKSUM) $(BIN_DIR)/$(IMAGE2_OTA2) || true
        $(OTA) $(BIN_DIR)/$(IMAGE2_OTA1) 0x800B000 $(BIN_DIR)/$(IMAGE2_OTA2) 0x080800000 0x20170111 $(BIN_DIR)/$(OTA_ALL)
else
        @echo ==========================================================
        @echo ota_idx must be "1" or "2"
        @echo ==========================================================
```


（3）修改rtl8710b_ota.h文件中的分区地址，按照如下图所示的方式进行修改即可，需要注意的是，所有地址均按照上述事先需要确定的分区地址修改：

```
/** @defgroup OTA_Exported_Constants OTA Exported Constants
  * @{
  */
/** @defgroup OTA_system_parameter_definitions
  * @{
  */
#define OFFSET_DATA    FLASH_SYSTEM_DATA_ADDR      /*system data offset address*/
#define BACKUP_SECTOR (FLASH_SYSTEM_DATA_ADDR - 0x1000) /*back up system data offset address*/
#define OTA1_ADDR    0x0800B000             /*the OTA1 start address*/
#define OTA2_ADDR    0x080F7000             /*the OTA2 start address*/
#define BUF_SIZE       512                 /*the size of the buffer used for receiving firmware data from server*/
#define RDP_FLASH_ADDR  0x081FC000             /*the flash addresss for RDP image*/

#define OTA_IMAG       0               /*identify the OTA image*/
#define RDP_IMAG       1               /*identify the RDP image*/
```


（4）使用配置属性的方法设置主分区和备份分区的地址，地址需要与上述所述的确认的OTA1的地址和OTA2的地址相对应：

```
unsigned int masterPkgAddr = 0x0800B000;
if (MagicLinkSetAttr(MAGICLINK_ATTR_OTA_FLASH_MASTER_PKG_ADDR, &masterPkgAddr,
sizeof(masterPkgAddr)) != 0) {
    printf("set ota1 addr fail");
}

unsigned int slavePkgAddr = 0x080F7000;
if (MagicLinkSetAttr(MAGICLINK_ATTR_OTA_FLASH_SLAVE_PKG_ADDR, &slavePkgAddr,
sizeof(slavePkgAddr)) != 0) {
    printf("set ota2 addr fail");
}
```

2. 根据上述的修改在application.mk中通过指定ota_idx的值分别编译生成OTA1和OTA2所对应的固件包：image2_all_ota1.bin和image2_all_ota2.bin。
3. 将image2_all_ota1.bin和image2_all_ota2.bin固件连同4.11章节中所述的filelist.xml文件一同上传至OTA服务器进行升级包的部署。且因为升级包中包含两个固件，需要修改filelist.xml文件格式如下：

```
<?xml version="1.0" encoding="utf-8"?>
<root>
<files>
<file>
<spath>image2_all_ota1.bin</spath>
<sha256>5ED6E6865A97C84C743809E68AF4492BD29E250E14C6BA5670528D9DD8DEB648</sh
a256>
<size>811796</size>
</file>
<file>
<spath>image2_all_ota2.bin</spath>
<sha256>1CA35B4D2DF8F4DC1EDEB62C24B438A6B6B57A60F543EAFC8E053C6A2BF27AB3</sh
a256>
<size>811796</size>
</file>
</files>
</root>
```

   - spath字段不修改，保持为image2_all_ota1.bin和image2_all_ota2.bin。
   - sha256字段对应填入image2_all_ota1.bin和image2_all_ota2.bin的SHA256校验值。
   - size字段对应填入image2_all_ota1.bin和image2_all_ota2.bin的文件大小，单位为字节。

7.2 ESP32-C3模组OTA适配注意事项

1. 在ESP32-C3平台上，以下固件和OTA分区相关的HONOR Connect IoT SDK运行参数不生效，固件和OTA分区的起始地址和存储空间大小以ESP32-C3分区表为准：

```
MAGICLINK_ATTR_OTA_FLASH_MASTER_PKG_ADDR, /* 整数，固件主分区flash起始地址，必须为4K对齐 */

MAGICLINK_ATTR_OTA_FLASH_SLAVE_PKG_ADDR, /* 整数，固件备份分区flash起始地址，必须为4K对齐 */

MAGICLINK_ATTR_OTA_FLASH_MASTER_PKG_SIZE, /* 整数，OTA固件主分区flash存储空间大小，单位为字节，必须为4K的整数倍，建议该值设置要比实际固件大，考虑升级可扩展性 */

MAGICLINK_ATTR_OTA_FLASH_SLAVE_PKG_SIZE, /* 整数，OTA固件备份分区flash存储空间大小，单位为字节，必须为4K的整数倍，建议该值设置要比实际固件大，考虑升级可扩展性 */
```

2. ESP32-C3只有软重启接口，重启时WiFi, BT, UART0, SPI1，legacy timers会被复位，其余外设不会被复位，对以下HONOR Connect IoT SDK运行参数的修改不会生效：

```
MAGICLINK_ATTR_OTA_REBOOT_METHOD = 200, /* 整数，0表示软重启，1表示硬重启 */
```

7.3 BL602模组OTA适配注意事项

1. 在BL602平台，ota升级包FW_OTA.bin.ota需要使用BLDevCube工具生成，生成步骤如下图所示：