OPL1000

ULTRA-LOW POWER 2.4GHZ WI-FI + BLUETOOTH SMART SOC

Sensor Device Reference via AWS cloud Application Guide

OPULINKS

OPL1000-Reference-Sensor-Device-AWS-cloud-guide-R01 | Version V01

REVISION HISTORY

Date	Version	Contents Updated
2019-12-24	0.1	● Initial Release

LIST OF TABLES

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF TABLES

1. 介绍

1.1. 文档应用范围

本文档介绍如何基于 OPL1000 以及亚马逊云 MQTT 协议开发上下行的完整应用，主要针对 AWS blewifi example 的介绍。 内容包括固件设计，云端设备配置以及操作过程。

1.2. 缩略语

Abbr.	Explanation
AP	Wireless Access Point 无线访问接入点
APP	APPlication 应用程序
APS	Application Sub-system 应用子系统，在本文中亦指 M3 MCU
AWS	Amazon Web Services 亚马逊云服务
Blewifi	BLE config WIFI 蓝牙配网应用
DevKit	Development Kit 开发工具板
MQTT	Message Queuing Telemetry Transport 消息队列遥测传输协议
OTA	Over-the-Air Technology 空间下载技术
TCP	Transmission Control Protocol 传输控制协议

1.3. 参考文献

[1] OPL1000 数据手册 OPL1000-DS-NonNDA.pdf

[2] Download 工具使用指南 OPL1000-patch-download-tool-user-guide.pdf

访问链接：https://github.com/Opulinks-Tech/OPL1000A2-SDK/tree/master/Doc/OPL1000A2-patch-download-tool-user-guide.pdf

[3] SDK 开发使用指南 OPL1000-SDK-Development-guide.pdf

访问连接：https://github.com/Opulinks-Tech/OPL1000A2-SDK/blob/master/Doc/OPL1000-SDK-Development-guide.pdf

[4] AWS key app tool 使用指南 OPL1000-aws-key-app-tool-guide.pdf

2. 项目构成和工作原理

2.1. 项目构成

AWS 参考用例项目由以下几部分构成：

- 项目 code
- 说明文档
- AWS 证书写入工具
- AWS 证书写入工具介绍

2.2. 工作原理

AWS 参考设计主要部件：物联网模块 OPL1000，移动设备（APP），云端（亚马逊云）。

Figure 1:工作原理图

3. 运行 AWS 应用

运行 OPL1000 AWS 应用需要以下步骤：

a) 更新工程配置文件，修改头文件中的宏定义参数（参考 3.1 章节）。
b) 使用编译工具完成项目工程编译，生成 M3 bin 文件（二进制固件文件）。
c) 通过 download tool 打包 M3 bin 文件，生成完整固件 opl1000.bin，并下载到 opl1000 模块。
d) 打开 OPL1000 蓝牙配网 app，进行蓝牙扫描动作，扫描 opl1000 蓝牙设备，配置连接能够访问 Internet 的 AP。
e) 用 OpulinksAWSBidirection APP（该 apk 在工程的 tools 目录下）配网连接 AWS 云，在 apk 上设置灯的状态和查看门磁的状态。

3.1. 生成 OPL1000 设备固件

编译 prj_src 目录下的 opl1000_app_m3 项目工程文件可以生成 OPL1000 M3 固件。在编译之前用户可以根据需要自行修改参数及头文件。

使用 Keil C 手动更新参数配置需要分两步完成：

1 第一步使用 Keil C 开发工具打开头文件 （blewifi_configuration.h），是否进入省电模式，设备名称等参数；

2 第二步使用 download 工具完成固件 Pack，下载操作，请参考 “Download 工具使用指南 OPL1000-patch-download-tool-user-guide.pdf" 了解操作方法。

3 使用提供的 AWS key 工具写入证书，事物名称，节点等信息。**注意**：关于 AWS key 的申请，app 的创建，和 key 的下载，请参考 “AWS key app tool 使用指南 OPL1000-aws-key-app-tool-guide.pdf" 了解操作方法。

3.2. OPL1000 APP 完成蓝牙配网

首先确认需要连接设备的 MAC 地址， APP 会自动扫描附近的 OPL1000 蓝牙信息，点击连接，然后进入配置 wifi 界面，此时会扫描附近可用的 WIFI AP，然后点击连接需要连接的 AP，细节参考 “OPL1000-Demo-BLE-setup-network-and-BLE-OTA-guide.pdf” 。

Figure 2:OPL1000 APP 开启蓝牙扫描界面

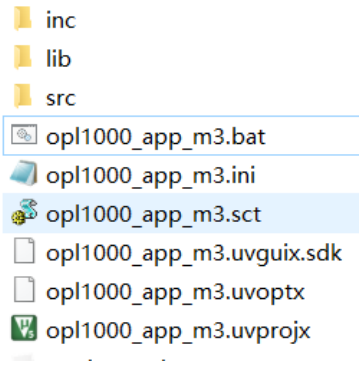
4. AWS 应用设计

本章介绍设备端固件工作原理，以及如何进行功能扩展。

4.1. 项目工程构成

如 Figure 3 所示，AWS 项目包含蓝牙配网， MQTT 处理和库文件等目录。

Figure 3:工程文件构成



各文件夹及文件构成如表。具体内容如 Table 1 所述。

Table 1: AWS 项目文件夹和内容

文件夹和文件	内容说明
inc	包含工程编译所需的头文件
lib	包含工程编译所需要的 lib 库
src	存放蓝牙配网，数据收发相关.c 和.h 头文件，以及 main 文件
opl1000_app_m3.bat opl1000_app_m3.ini opl1000_app_m3.sct opl1000_app_m3.uvoptx opl1000_app_m3.uvprojx	编译工程文件。

4.2. 参数配置 blewifi_configuration.h 使用说明

blewifi_configuration.h 文件集中了需要配置的参数，用户可以根据实际应用更新参数配置。

blewifi_configuration.h 文件定义了可配置参数的默认值。

Table 2 主要参数配置宏定义功能详细介绍

宏定义	说明
MW_FIM_VER11_PROJECT	Group11 的 FIM 版本信息，取值范围为 0x00-0xFF. Notes: 当该文件中的宏定义值有更新时，请务必更新一下这个值（只有跟原来的值不一样就好）。
MW_FIM_VER13_PROJECT	Group13 的 FIM 版本信息，取值范围为 0x00-0xFF. Notes: 当该文件中的宏定义值有更新时，请务必更新一下这个值（只有跟原来的值不一样就好）。
BLEWIFI_COM_POWER_SAVE_EN	是否 Enable smart sleep. 1: Enable. 0: Disable
BLEWIFI_COM_RF_POWER_SETTINGS	用于设置 RF 模式。具体取值请参考该文件的注释。

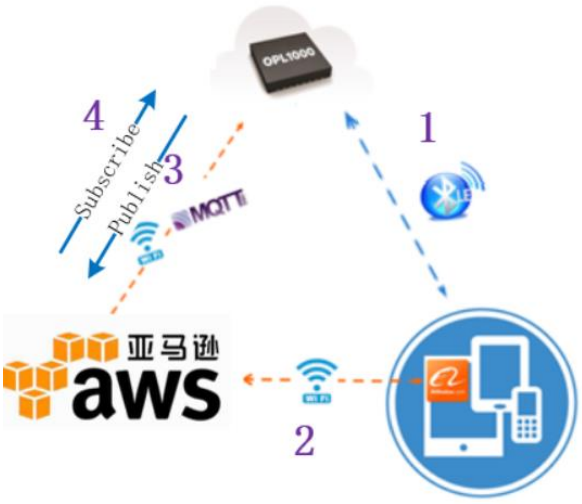
4.3. 执行流程和模块说明

本章节介绍 OPL1000 固件处理流程。

4.3.1. 执行流程

主程序执行流程如 Figure 4 所示。在完成设备初始化操作后，设备将自动尝试连接 AWS 云。如果连接成功，设备会 publish 一段字符串，并且等待服务端的订阅消息。

Figure 4:固件执行流程图



4.3.2. 主要 Task Handler

本项目内部启动了两个任务处理器

1. BLE Handler

BLE Handler 功能是等待手机端蓝牙与 OPL1000 的连接，此时 OPL1000 会持续发送 BLE 广播，直到蓝牙建立连接

2. WIFI Handler

WIFI Handler 是 OPL1000 与 AP 建立连接后，连线及断线检查，断线后重连功能

4.3.3. 云连接和数据传输

OPL1000 与 AWS 云通过 TCP 协议连接，数据传输则采用的是 MQTT(v3.1)传输协议。

MQTT 协议工作原理如 Figure 5 所示。

Figure 5:MQTT 实现方式图

MQTT 协议中有三种身份：发布者（Publish）、代理（Broker）（服务器）、订阅者（Subscribe）。其中，消息的发布者和订阅者都是客户端，消息代理是服务器即 AWS 云，消息发布者可以同时是订阅者。

MQTT 传输的消息分为：主题（Topic）和负载（payload）两部分

Topic，可以理解为消息的类型，订阅者订阅（Subscribe）后，就会收到该主题的消息内容（payload）

MQTT 会构建底层网络传输：它将建立客户端到服务器的连接，提供两者之间的一个有序的、无损的、基于字节流的双向传输，当应用数据通过 MQTT 网络发送时，MQTT 会把与之相关的服务质量（QoS）和主题名（Topic）相关连。

5. AWS 智能传感器应用开发

为方便用户在本例程的基础上做二次开发，本章将介绍一些基于 AWS 智能传感器应用开发过程中需要改动的代码和常见问题的处理方法。

5.1. MQTT 收发消息相关的更新

用户做二次开发时，通常需要为设备规划好收发消息的主题和数据包的格式。在本例程中，下行数据(Rx)的主题在 subscribe_publish_sample.c 文件中定义和订阅如下：

Figure 6：主题的定义

```
#define HOST_ADDRESS_SIZE 255
#define SUBSCRIBE_DOOR_TOPIC  "OPL/AWS/%02x%02x%02x/DOOR"
#define SUBSCRIBE_LIGHT_TOPIC "OPL/AWS/%02x%02x%02x/LIGHT"

char SubscribeTopic_Door[MAX_SIZE_OF_TOPIC_LENGTH];
char SubscribeTopic_Light[MAX_SIZE_OF_TOPIC_LENGTH];
```

...

```
IOT_INFO("Subscribing Door topic : %s", SubscribeTopic_Door);
rc = aws_iot_mqtt_subscribe(&client, SubscribeTopic_Door, strlen(SubscribeTopic_Door), QOS1, iot_subscribe_callback_handler_Door, NULL);
if(SUCCESS != rc) {
    IOT_ERROR("Error subscribing Door: %d ", rc);
    return rc;
}

IOT_INFO("Subscribing Light topic : %s", SubscribeTopic_Light);
rc = aws_iot_mqtt_subscribe(&client, SubscribeTopic_Light, strlen(SubscribeTopic_Light), QOS1, iot_subscribe_callback_handler_Light, NULL);
if(SUCCESS != rc) {
    IOT_ERROR("Error subscribing Light : %d ", rc);
    return rc;
}
```

而上行数据(Tx)的格式和发布在 sensor_https.c 文件中定义和实现如下：

Figure 7：上行数据的格式和发布

```
#define CALC_AUTH_KEY_FORMAT "{\"apikey\":\"%s\",\"deviceid\":\"%s\",\"d_seq\":\"%u\"}"
#define POST_DATA_DEVICE_ID_FORMAT "{\"deviceid\":\"%s\",\"d_seq\":\"%u\",\"params\":{"
#define POST_DATA_SWITCH_FORMAT "\"switch\":\"%s\","
#define POST_DATA_BATTERY_FORMAT "\"battery\":%s,"
#define POST_DATA_FWVERSION_FORMAT "\"fwVersion\":\"%s\","
#define POST_DATA_TYPE_FORMAT "\"type\":%d,"
#define POST_DATA_CHIP_FORMAT "\"chipID\":\"%02x%02x%02x%02x%02x%02x%02x\","
#define POST_DATA_MACADDRESS_FORMAT "\"mac\":\"%02x%02x%02x%02x%02x%02x\","
#define POST_DATA_RSSI_FORMAT "\"rssi\":%d}}"
#define OTA_DATA_URL "%s?deviceid=%s&ts=%u&sign=%s"
```

```
/* Combine https Post Content */
memset(content_buf, 0, BUFFER_SIZE);
snprintf  content_buf, BUFFER_SIZE, POST_DATA_DEVICE_ID_FORMAT
                                    POST_DATA_SWITCH_FORMAT
                                    POST_DATA_BATTERY_FORMAT
                                    POST_DATA_FWVERSION_FORMAT
                                    POST_DATA_TYPE_FORMAT
                                    POST_DATA_CHIP_FORMAT
                                    POST_DATA_MACADDRESS_FORMAT
                                    POST_DATA_RSSI_FORMAT
                                  , g_tAWSDeviceInfo.client_id
                                  , PostContentData->TimeStamp
                                  , PostContentData->DoorStatus ? "off": "on"
                                  , PostContentData->Battery
                                  , PostContentData->FwVersion
                                  , PostContentData->ContentType
                                  , ubaWiFiMacAddr[0], ubaWiFiMacAddr[1], ubaWiFiMacAddr[2], ubaWiF
                                  , PostContentData->ubaMacAddr[0], PostContentData->ubaMacAddr[1],
                                  , PostContentData->rssi
                                  );

  paramsQOS1.qos = QOS1;                                          len = strlen(content_buf);
  paramsQOS1.payload = (void *) g_ubSendBuf;                      memcpy(g_ubSendBuf, content_buf, strlen(content_buf));
  paramsQOS1.isRetained = 0;
  paramsQOS1.payloadLen = len;

  do
  {
      //PostResult = Sensor_Https_Post(g_ubSendBuf, len);
      if( false == BleWifi_Ctrl_EventStatusGet(BLEWIFI_CTRL_EVENT_BIT_IOT_INIT))
      {
          printf("IOT not init \n");
          break;
      }

      PostResult = aws_iot_mqtt_publish(&client, SubscribeTopic_Door, strlen(SubscribeTopic_Door), &paramsQOS1);
      Count++;
```

5.2. 外设控制

外设控制大致可分为四个方面：外设状态的收集，外设状态的上报，外设状态的接收，和外设状态的设置。

5.2.1. 外设状态的收集

外设状态的收集指的是 OPL1000 通过 GPIO 口采集到各类传感器数据的过程。这个数据可以是灯的开或关的状态，物体的重量，和门磁的闭合状态等。这个过程会涉及 GPIO pin 的定义，API 接口函数的定义和调用位置等问题。

在本例程中，门磁闭合状态的收集用到的是 GPIO_IDX_05 pin(在 blewifi_configuration.h 文件中定义)，由 Hal_Vic_GpioInput()函数读取该引脚的电平，然后设置门磁的状态并把数据 push 到 ring buffer.

在去抖动后，这个过程由 BleWifi_Ctrl_TaskEvtHandler_DoorDebounceTimeOut()函数实现，它定义在 blewifi_ctrl.c 文件中如下。

Figure 8：外设状态的获取

```
static void BleWifi_Ctrl_TaskEvtHandler_DoorDebounceTimeOut(uint32_t evt_type, void *data, int len)
{
    BLEWIFI_INFO("BLEWIFI: MSG BLEWIFI_CTRL_MSG_DOOR_DEBOUNCETIMEOUT \r\n");
    unsigned int u32PinLevel = 0;

    // Get the status of GPIO (Low / High)
    u32PinLevel = Hal_Vic_GpioInput(MAGNETIC_IO_PORT);
    printf("MAG_IO_PORT pin level = %s\r\n", u32PinLevel ? "GPIO_LEVEL_HIGH:OPEN" : "GPIO_LEVEL_LOW:CLOSE");

    // When detect falling edge, then modify to raising edge.

    // Voltage Low   / DoorStatusSet - True  / Door Status - CLose - switch on  - type = 2
    // Voltage Hight / DoorStatusSet - False / Door Status - Open  - switch off - type = 3
    if(GPIO_LEVEL_LOW == u32PinLevel)
    {
        /* Disable - Invert gpio interrupt signal */
        Hal_Vic_GpioIntInv(MAGNETIC_IO_PORT, 0);
        // Enable Interrupt
        Hal_Vic_GpioIntEn(MAGNETIC_IO_PORT, 1);

        if (BleWifi_Ctrl_EventStatusGet(BLEWIFI_CTRL_EVENT_BIT_DOOR) == false)
        {
            /* Send to IOT task to post data */
            BleWifi_Ctrl_EventStatusSet(BLEWIFI_CTRL_EVENT_BIT_DOOR, true); // true is Door Close
            Sensor_Data_Push(BleWifi_Ctrl_EventStatusGet(BLEWIFI_CTRL_EVENT_BIT_DOOR), BleWifi_Ctrl_EventStatu
            Iot_Data_TxTask_MsgSend(IOT_DATA_TX_MSG_DATA_POST, NULL, 0);
        }
    }
```

5.2.2. 外设状态的上报

外设状态的上报指的是 OPL1000 在收集到外设相关的数据后，把它封装为指定主题的 MQTT 数据包，并发布(publish)到 AWS 云端的过程。

在本例程中，门磁闭合状态的上报由在 sensor_https.c 文件中的 Sensor_Https_Post_On_Line()函数实现(用户可根据需要自行修改这里的代码，达到定制化的目的),它在 build 好数据后调用 AWS SDK 库的 aws_iot_mqtt_publish ()函数实现数据上报。

Figure 9： 外设状态的上报

```
int Sensor_Https_Post_On_Line(void)
{
    int len = 0;
    int PostResult = 0;
    int Count = 0;
    HttpPostData_t PostContentData;
    uint8_t ubaSHA256CalcStrBuf[SCRT_SHA_256_OUTPUT_LEN];
    IoT_Publish_Message_Params paramsQOS1;
```

...

```
paramsQOS1.qos = QOS1;
paramsQOS1.payload = (void *) g_ubSendBuf;
paramsQOS1.isRetained = 0;
paramsQOS1.payloadLen = len;

do
{
    //PostResult = Sensor_Https_Post(g_ubSendBuf, len);
    if( false == BleWifi_Ctrl_EventStatusGet(BLEWIFI_CTRL_EVENT_BIT_IOT_INIT))
    {
        printf("IOT not init \n");
        break;
    }

    PostResult = aws_iot_mqtt_publish(&client, SubscribeTopic_Door, strlen(SubscribeTopic_Door), &paramsQOS1);
    Count++;
```

5.2.3. 外设状态的接收

外设状态的接收指的是 OPL1000 在订阅(subscribe)特定主题后，收到从 AWS 云端下发的该主题的消息，并提取出跟外设状态设置或控制相关的数据的过程。在本例程中，通过注册回调函数实现消息的接收和解析的过程。消息的接收和解析由在 subscribe_publish_sample.c 文件中的 iot_subscribe_callback_handler_Light()函数实现.

Figure 10： 外设状态的接收和解析

```
void iot_subscribe_callback_handler_Light(AWS_IoT_Client *pClient, char *topicName, uint16_t topicNameLen,
                                          IoT_Publish_Message_Params *params, void *pData) {
    IOT_UNUSED(pData);
    IOT_UNUSED(pClient);
    IOT_INFO("Subscribe Light callback");

    if(strncmp((const char *) params->payload, "on", strlen("on")) == 0)
    {
        IOT_INFO("++ %.*s\t%.*s", topicNameLen, topicName, (int) params->payloadLen, (char *) params->payload);
        Hal_Vic_GpioOutput(LED_IO_PORT, GPIO_LEVEL_HIGH);
    }
    else if(strncmp((const char *) params->payload, "off", strlen("off")) == 0)
    {
        IOT_INFO("-- %.*s\t%.*s", topicNameLen, topicName, (int) params->payloadLen, (char *) params->payload);
        Hal_Vic_GpioOutput(LED_IO_PORT, GPIO_LEVEL_LOW);
    }
```

而 iot_subscribe_callback_handler_Light()则是在订阅主题时注册的。

Figure 11：注册订阅主题的回调函数

```
IOT_INFO("Subscribing Door topic : %s", SubscribeTopic_Door);
rc = aws_iot_mqtt_subscribe(&client, SubscribeTopic_Door, strlen(SubscribeTopic_Door), QOS1, iot_subscribe_callback_handler_Door, NULL);
if(SUCCESS != rc) {
    IOT_ERROR("Error subscribing Door: %d ", rc);
    return rc;
}

IOT_INFO("Subscribing Light topic : %s", SubscribeTopic_Light);
rc = aws_iot_mqtt_subscribe(&client, SubscribeTopic_Light, strlen(SubscribeTopic_Light), QOS1, iot_subscribe_callback_handler_Light, NULL);
if(SUCCESS != rc) {
    IOT_ERROR("Error subscribing Light : %d ", rc);
    return rc;
}
```

5.2.4. 外设状态的设置

外设状态的设置指的是 OPL1000 在提取出跟外设状态设置或控制相关的数据后，通过 GPIO 口设置各类设备的状态(如，灯的开/关等)的过程。在本例程中，灯的开关动作通过直接调用 HAL 层提供的接口函数 Hal_Vic_GpioOutput()实现。

5.3. AWS 收发消息 buffer 的大小调整

本例程使用的 AWS SDK 库的版本是 3.0.1.它所在的目录是 src\aws-iot-device-sdk-embedded-C-3.0.1。在 src\aws-iot-device-sdk-embedded-C-3.0.1\include\opl_aws_iot_config.h 文件中有定义了 AWS_IOT_MQTT_RX_BUF_LEN 和 AWS_IOT_MQTT_TX_BUF_LEN 两个宏，分别用于指定发送和接收消息的 buffer 的最大长度。它们的默认值分别为 2048 和 512.用户可根据需要自行修改它们。

Figure 12：AWS 收发消息用的 buffer 大小

```
// MQTT PubSub
#define AWS_IOT_MQTT_RX_BUF_LEN 2048
#define AWS_IOT_MQTT_TX_BUF_LEN 512 ///< Any time a
#define AWS_IOT_MQTT_NUM_SUBSCRIBE_HANDLERS 5 ///<
#define MAX_SIZE_OF_TOPIC_LENGTH 128
```

5.4. Heap size 的调整

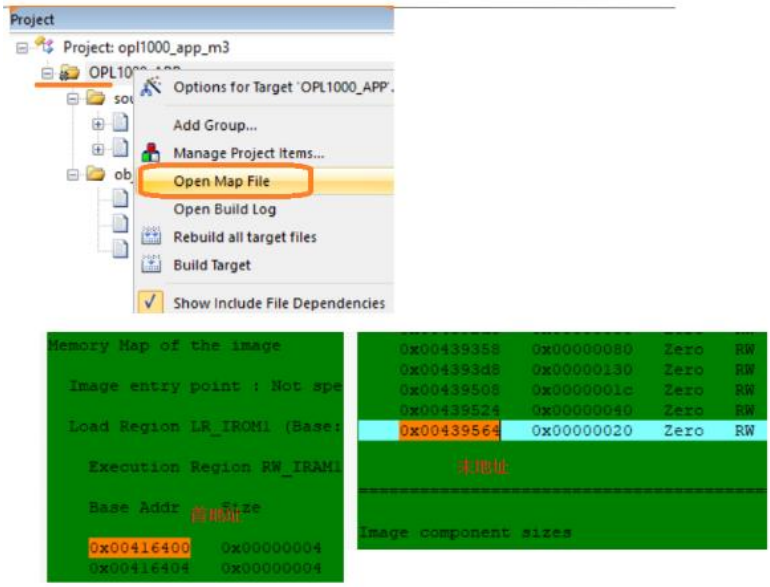
随着功能的不断加入，应用程序可能会碰到由于代码的不断增加，导致空间不够而引起的宕机问题.用户可以参考本节的内容，对 scatter file, heap size 做适当调整，尝试着解决内存不够问题。

（1）在调整 scatter file 之前，需要在 map 文件获取首末地址，方法如下：

- 用 keil 打开相应的工程
- 右键点击工程,选择'Open Map File'选项
- 在打开的.map 文件中获取 image 在 Memory Map 中的首地址和末地址。

在该例子中，首地址是： 0x00416400 ，末地址是： 0x00439564 。如下图所示。对末地址以 4k 或其它值(1k,512B 等)为单位向上取整，为 0x0043a000.

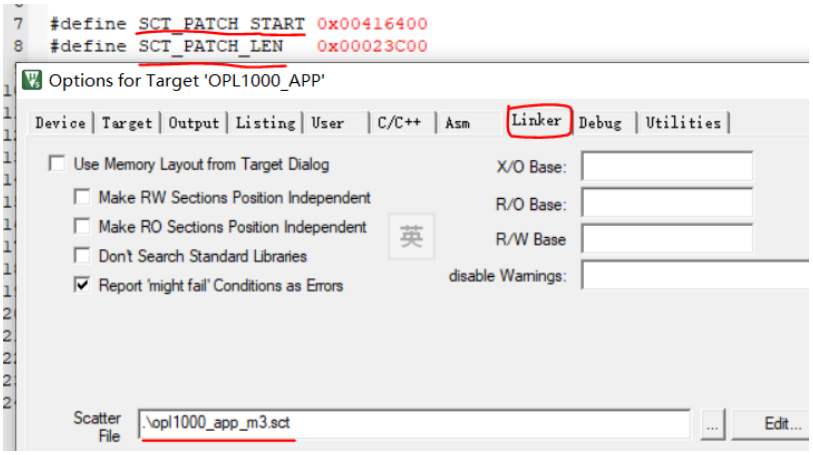
Figure 13：获取首末地址



（2）在 Memory Map 中获取首，末地址后，就可根据它们对 scatter file 做相应调整，方法如下：

- 右键点击工程,选择'Options'选项,选择'Linker'，再点击'Edit'scatter file,如下图所示
- 保持 SCT_PATCH_START 的值不变，而 SCT_PATCH_LEN 则可以改为末地址 – 首地址的值，如在该例中，大小 = 0x0043a000 - 0x00416400 = 0x00023C00 。

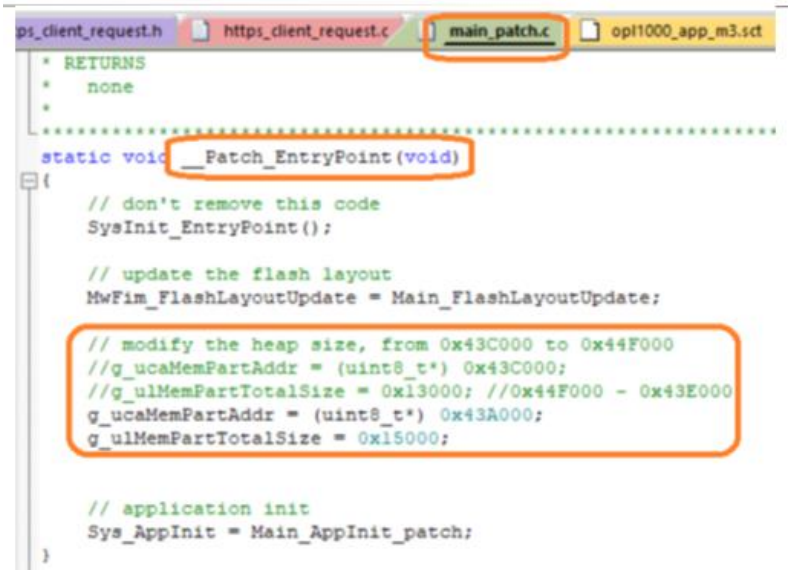
Figure 14：修改 SCT_PATCH_LEN 值



（3）在修改 scatter file 文件后，就可根据它设置 Heap 的起始地址，并计算出它的大小，方法如下：

- Heap 的设置通常都放在工程的 main_patch.c 文件的__Patch_EntryPoint() 中进行；
- Heap 的起始地址可以设置为末地址 ，在该例中也就是的 0x0043a000 。大小 = 0x44F000 - 末地址 ，对于下图中的例子，就是，大小 = 0x44F000 - 0x0043a000 = 0x15000

Figure 15： 调整 heap size 值



5.5. 低功耗相关的注意事项

在 blewifi_configuration.h 文件中定义了三个跟低功耗有关的宏：BLEWIFI_COM_POWER_SAVE_EN，BLEWIFI_WIFI_DTIM_INTERVAL，和 BLEWIFI_COM_RF_POWER_SETTINGS 宏。

其中，BLEWIFI_COM_POWER_SAVE_EN 控制是否使能 smart sleep 模式，它的默认值为 1，即使能的。

Figure 16： smart sleep 相关的宏定义

blewifi_configuration.h

```
17  FIM version
18  */
19  #define MW_FIM_VER11_PROJECT            0x03    // 0x00 ~ 0xFF
20
21  /*
22  Smart sleep
23  */
24  #define BLEWIFI_COM_POWER_SAVE_EN       (1)     // 1: enable    0: disable
```

用户可根据需要自行调用以下接口函数(在 ps_public.h 文件中定义)：

```
void ps_smart_sleep(int enable);
```

BLEWIFI_WIFI_DTIM_INTERVAL 用于设置 DTIM 的默认值，该值越大，功耗越低，响应时间也相应加长。用户可根据需要自行调用以下接口函数(在 blewifi_wifi_api.h 文件中定义)：

int BleWifi_Wifi_SetDTIM(uint32_t value)；

BLEWIFI_COM_RF_POWER_SETTINGS 用于设置 BLE 和 WIFI 模块的功耗模式的默认值，默认值为 0x00，用户可根据需要自行调用以下接口函数(在 blewifi_common.h 文件中定义)：

void BleWifi_RFPowerSetting(uint8_t level)；

CONTACT

sales@Opulinks.com