

# OPL1000

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

## Sensor Device Reference via Ali Cloud Application Guide



OPULINKS

---

<http://www.opulinks.com/>

Copyright © 2019, Opulinks. All Rights Reserved.

---

OPL1000-reference-Sensor-Device-Ali-cloud-guide | Version V02

Date	Version	Contents Updated
2019-11-08	0.1	• Initial Release
2019-11-26	0.2	Support duplexing MQTT data communication

## TABLE OF CONTENTS

1. 介绍	3
1.1. 文档应用范围	3
1.2. 缩略语	3
1.3. 参考文献	3
2. 项目构成和工作原理	5
2.1. 项目构成	5
2.2. 工作原理	6
3. 运行阿里智能传感器参考设计	7
3.1. 生成 OPL1000 设备固件	7
3.2. 云智能 APP 完成蓝牙配网	8
3.3. 检查 OPL 设备工作状态	9
3.4. APP 功能界面	11
3.5. 更新五元组	11
4. 阿里智能传感器 应用设计	12
4.1. 项目工程构成	12
4.2. 参数配置 blewifi_configuration.h 使用说明	13
4.3. 执行流程和模块说明	13
4.3.1. 执行流程	13
4.3.2. 主要 Task Handler	14
4.3.3. 云连接和数据传输	14
5. 阿里智能传感器应用开发	16
5.1. MQTT 收发消息相关的更新	16
5.2. 外设控制	17
5.2.1. 外设状态的收集	17
5.2.2. 外设状态的上报	19
5.2.3. 外设状态的接收	20
5.2.4. 外设状态的设置	21
5.3. 阿里收发消息 buffer 的大小调整	21
5.4. Heap size 的调整	22
5.5. 低功耗相关的注意事项	25

## LIST OF FIGURES

Figure 1:云智能 APP 下载链接.....	5
Figure 2:项目文件.....	5
Figure 3:工作原理图 .....	6
Figure 4:云智能 APP 查找 OPL1000 设备列表 .....	8
Figure 5:网络连接界面.....	9
Figure 6:阿里云连接串口 log 状态信息.....	9
Figure 7:云端查看设备状态.....	10
Figure 8:阿里智能传感器设备初始化、添加和功能界面 .....	10
Figure 9: APP 功能界面.....	11
Figure 10:工程文件构成 .....	12
Figure 11:固件执行流程图.....	14
Figure 12:MQTT 实现方式图.....	15
Figure 13:下行数据的解析.....	16
Figure 14: 上行数据的上报 .....	17
Figure 15: 门磁状态的收集 .....	18
Figure 16: 门磁状态的上报 .....	19
Figure 17: LED 灯状态的接收.....	20
Figure 18: Ali linkkit 回调函数的注册.....	20
Figure 19: LED 灯状态的设置.....	21
Figure 20: Ali 收发 buffer size 的设置 .....	22
Figure 21: 从 map 文件中获取首末地址 .....	23
Figure 22: 更新 SCT file 中的 size .....	23
Figure 23: 更新 heap size 相关的信息 .....	24
Figure 24: smart sleep 相关的宏定义 .....	25

# 1. 介绍

## 1.1. 文档应用范围

本文档介绍基于 OPL1000 A2 芯片的阿里传感器智能控制参考设计。本设计中连接到阿里云，使用 MQTT 协议完成双向数据传输。具体在设备上使用门磁作为传感器，上传 MQTT 数据到云端；使用 APP 控制设备的 LED 灯，即用下行 MQTT 数据控制灯的开关。本文档内容包括固件设计，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
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] Ali Cloud 物联网开发官方文档

访问链接：<https://living.aliyun.com/doc#index.html>

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



访问链接: <https://github.com/Opulinks-Tech/OPL1000A2-SDK/blob/master/Doc/OPL1000-SDK-Development-guide.pdf>

[5] Ali 5 元组和应用程序产生指导文档 OPL1000\_ali\_key&app\_create\_guide.pdf

访问链接: [https://github.com/Opulinks-Tech/OPL1000A2-Sensor-Device-Reference-Code-Ali-Cloud-with-MQTT/tree/master/doc/OPL1000\\_ali\\_key&app\\_create\\_guide.pdf](https://github.com/Opulinks-Tech/OPL1000A2-Sensor-Device-Reference-Code-Ali-Cloud-with-MQTT/tree/master/doc/OPL1000_ali_key&app_create_guide.pdf)

## 2. 项目构成和工作原理

### 2.1. 项目构成

阿里智能传感器参考设计需要下载阿里云智能 APP。云智能 APP 是阿里云物联网云智能手机应用程序，用于 OPL1000 蓝牙配网以及设备的数据显示及操作（APP 软件可以扫描下面二维码下载，或者手机应用市场直接搜索‘云智能’）；

Figure 1:云智能 APP 下载链接

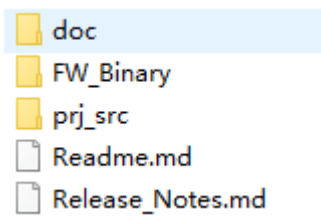
云智能APP 分为用户版和开发版，对于用户版需要产品发布才可以使用，否则无法扫描到蓝牙设备，对于开发版本则无此限制，云智能下载二维码如下：



注：使用的公版APP版本，若使用用户版本，则需要产品发布后才可以使⽤，否则无法扫描到对应设备的蓝牙，⽽开发版本则无此限制

参考设计项目文件如下图所示

Figure 2:项目文件



包含内容说明如下

目录和文件	说明
doc	存放 应用指南文档，即本文档
FW_Binary	存放需要 Pack 的 m0 文件，Pack 脚本文件；以及 opl1000_ali_sensor.bin 参考设计固件 bin 文件。该文件可以直接下载到硬件设备中使用。
prj_src	包含本参考设计的库文件，头文件以及应用层源代码
Readme.md	说明本参考设计功能和内容

Release_Notes.md	描述本版本发布更新内容和注意事项
------------------	------------------

2.2. 工作原理

阿里智能传感器参考设计主要部件：物联网模块 OPL1000，移动设备（APP），云端（阿里云），物联网设备（包括门磁和 led 灯）。

Figure 3:工作原理图





### 3. 运行阿里智能传感器参考设计

运行 OPL1000 阿里智能传感器智能应用(简称 阿里智能传感器项目)包含以下步骤：

1. 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）。
2. 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）。
3. 通过 download tool 打包 M3 bin 和 M0 bin 文件，生成完整固件 opl1000.bin，并下载到 opl1000 模块。
4. 打开阿里云智能 app，进行蓝牙扫描动作，扫描 opl1000 蓝牙设备，配置连接能够访问互联网的 AP。
5. OPL1000 的固件连接云端。通过阿里云智能 app 控制灯的开关,门磁的状态也能上报到云端，并在云智能 app 上更新。

#### 3.1. 生成 OPL1000 设备固件

编译阿里智能传感器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四步完成：

- 1 第一步使用 Keil C 开发工具打开头文件 blewifi\_configuration.h，修改 IOT 设备五元组，是否进入省电模式，设备名称等参数；阿里五元码由下面的 5 个宏定义参数确定。

```
#define ALI_PRODUCT_ID      (00000000)
#define ALI_PRODUCT_KEY     "xxxxxxxxxxxx"
#define ALI_PRODUCT_SECRET  "xxxxxxxxxxxx"
#define ALI_DEVICE_NAME     "xxxxxxxxxxxx"
#define ALI_DEVICE_SECRET   "xxxxxxxxxxxx"
```

注意 blewifi\_configuration.h 头文件中定义的宏定义参数修改数值后（例如上述的 5 元码）

需要对 FIM 版本参数 MW\_FIM\_VERxx\_PROJECT 加 1，保证修改有效。

```
#define MW_FIM_VER11_PROJECT  0x04 // 0x00 ~ 0xFF
#define MW_FIM_VER12_PROJECT  0x02 // 0x00 ~ 0xFF
```

- 2 第二步编译工程，生成 M3 Bin 文件。对本项目生成的 bin 文件为：opl1000\_app\_m3.bin

- 3 第三步使用 download 工具将 M3 bin 文件 opl1000\_app\_m3.bin 和 M0 bin 文件 opl1000\_m0.bin 封装在一起生成 opl1000.bin 文件，下载到设备正。具体操作请参考 [“Download 工具使用指南 OPL1000-patch-download-tool-user-guide.pdf”](#) 了解操作方法。
- 4 第四步通过 3.5 节介绍的更新五元组的方法，确认阿里五元组已更新。

## 3.2. 云智能 APP 完成蓝牙配网

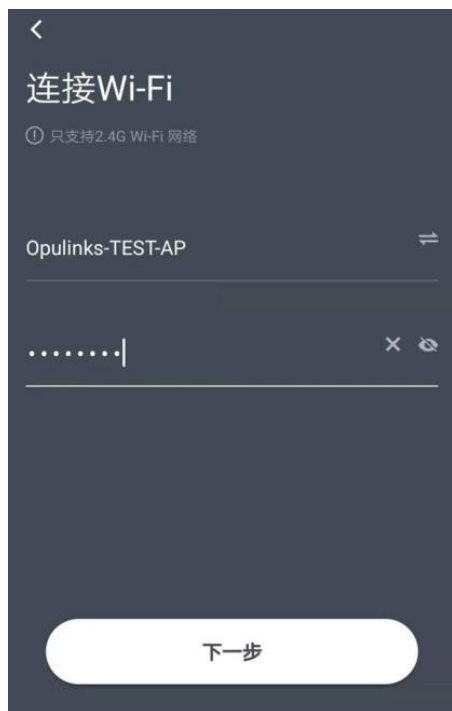
首先确认需要连接设备的 MAC 地址，以及设备名称，设备名称信息可以参考 3.1 工程编译设备名称(由宏 ALI\_DEVICE\_NAME 定义)，在 APP 点击+添加设备，APP 会自动扫描附近的 OPL1000 设备蓝牙信息。然后点击+配置需要连接的 AP

Figure 4:云智能 APP 查找 OPL1000 设备列表



点击需要配网的 OPL1000 设备右侧的 “+” 符号。进入 WIFI 配网界面，选择 AP 并输入密码，如下图所示。

Figure 5:网络连接界面



### 3.3. 检查 OPL 设备工作状态

OPL1000 设备是否连接到阿里云有三种方式来检查。

- 1 通过 OPL1000 设备的 UART 串口打印 log 信息来确认。出现下图信息表明阿里云连接成功。

Figure 6:阿里云连接串口 log 状态信息

```
user_initialized.157: Device Initialized, Devid: 0
user_connected_event_handler.38: Cloud Connected
```

- 2 在 UART 串口确认设备连网成功后，通过阿里云官网物联网接入模块中的“设备查看”检查设备在线状态。找到自己连接的设备对应名称，当设备信息出现在线状态时，则说明设备连接阿里云成功。

Figure 7:云端查看设备状态

forsvttest001	● 在线	2019-11-08 14:34:02	<a href="#">查看</a> <a href="#">调试</a> <a href="#">激活凭证</a>
---------------	------	---------------------	--

4 手机 APP 配网界面完成后进入初始化设备，添加设备和功能显示界面，说明配网和添加设备成功

Figure 8:阿里智能传感器设备初始化、添加和功能界面



### 3.4. APP 功能界面

下图是 APP 功能界面，APP 主要由四部分构成：门磁状态，LED 开关，接收信号强度和场景。

Ali APP 设计过程参见《[Ali 5 元组和应用程序产生指导文档](#)》介绍。

Figure 9: APP 功能界面



### 3.5. 更新五元组

设备五元组的更新请参考《OPL1000\_ali\_key&app\_create\_guide.pdf》的 P12.

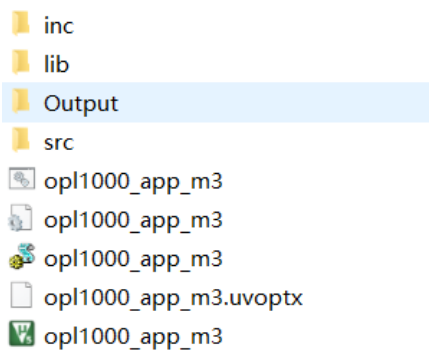
## 4. 阿里智能传感器 应用设计

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

如下图所示，阿里智能传感器项目包含蓝牙配网，数据收发相关的源文件，头文件和库文件等目录。

Figure 10: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: 阿里智能传感器项目文件夹和内容

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
Output	主要存放编译时产生的相关文件其中包括编译成功后的 opl1000_app_m3.bin 文件
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. 注意：当该文件中的宏定义值有更新时，请务必更新一下这个值（建议在原有值上加 1）。
MW_FIM_VER12_PROJECT	Group12 的 FIM 版本信息，取值范围为 0x00-0xFF. 注意：当该文件中的宏定义值有更新时，请务必更新一下这个值（建议在原有值上加 1）。
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。
ALI_PRODUCT_ID · ALI_PRODUCT_KEY · ALI_PRODUCT_SECRET · ALI_DEVICE_NAME · ALI_DEVICE_SECRET	Ali 云设备的五元组

4.3. 执行流程和模块说明

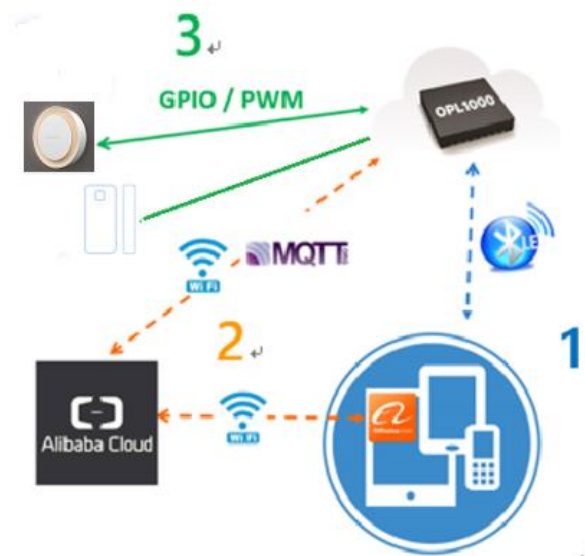
本章节介绍 OPL1000 固件处理流程。

4.3.1. 执行流程

在完成设备初始化操作后，设备将自动尝试连接阿里云。如果连接成功，用户就可以在手机端通过云智能 APP 控制智能灯的状态；门磁的状态也能够及时地更新在云智能 APP 上。



Figure 11: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 与阿里云通过 TCP 协议连接，数据传输则采用的是 MQTT(v3.1)传输协议。

MQTT 协议工作原理如下所示。



Figure 12: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阿里云，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. 阿里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阿里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，下行数据在 ali\_linkkitsdk\_impl.c 文件的 user\_property\_set\_event\_handler() 函数内，通过解析收到的数据包中的 'LedLightSwitch' 属性获取 Led 灯的开关状态(Rx)如下：

Figure 13:下行数据的解析

```
/* start : Add you code to handle request data */  
  
/* Try To Find LedLightSwitch Property */  
item_LedLightSwitch = cJSON_GetObjectItem(request_root, "LedLightSwitch");  
if (item_LedLightSwitch != NULL) {  
    EXAMPLE_TRACE("LedLightSwitch Enable : %d\r\n", item_LedLightSwitch->valueint);  
  
    T_BleWifi_Ctrl_DevStatus tStatus = {0};  
  
    tStatus.u8DevOn = item_LedLightSwitch->valueint;  
    tStatus.u8LedOn = tStatus.u8DevOn;  
  
    if(BleWifi_Ctrl_DevStatusSet(&tStatus))  
    {  
        EXAMPLE_TRACE("BleWifi_Ctrl_DevStatusSet fail\r\n");  
    }  
}  
  
/* end : Add you code to handle request data */
```

解析JSON格式來取值

自定義功能輸入的標誌符

根據取值來開關LED燈

而上行数据则在 ali\_linkkitsdk\_impl.c 文件的 user\_post\_property() 函数内，通过定义‘ContactState’属性上报门磁的状态(Tx):

Figure 14: 上行数据的上报



```

if(ptProp->u8Type == DEV_IND_TYPE_DOOR_STATUS)
{
    uint8_t u8FirstItemDone = 0;
    u32Offset += snprintf(ps8Buf + u32Offset, u32BufSize - u32Offset, "{");
    if(ptProp->tDoorStatus.u8EnableContact)
    {
        u8FirstItemDone = 1;
        u32Offset += snprintf(ps8Buf + u32Offset, u32BufSize - u32Offset, "\"ContactState\":%u", ptProp->tDoorStatus.u8ContactState);
    }
    if(ptProp->tDoorStatus.u8EnableRssi)
    {
        if(u8FirstItemDone)
        {
            u32Offset += snprintf(ps8Buf + u32Offset, u32BufSize - u32Offset, ",");
        }
        else
        {
            u8FirstItemDone = 1;
        }
        u32Offset += snprintf(ps8Buf + u32Offset, u32BufSize - u32Offset, "\"RSSI\":%d", ptProp->tDoorStatus.s8Rssi);
    }
    u32Offset += snprintf(ps8Buf + u32Offset, u32BufSize - u32Offset, "}");
}

```

## 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 15: 门磁状态的收集

```
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);
    BLEWIFI_INFO("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

            #if 1
            if (true == BleWifi_Ctrl_EventStatusGet(BLEWIFI_CTRL_EVENT_BIT_IOT_INIT))
            {
                BLEWIFI_WARN("[%s %d] Close\r\n", __func__, __LINE__);
                door_set(0, 1, 0);
            }
            #else

```

### 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)到阿里云端的过程。

在本例程中，门磁闭合状态的上报由在 ali\_linkkitsdk\_impl.c 文件中的 user\_post\_property() 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 好数据后调用阿里 SDK 库的 IOT\_Linkkit\_Report() 函数实现数据上报。

Figure 16: 门磁状态的上报

```
void user_post_property(IoT_Property_t *ptProp)
{
    int res = 0;
    user_example_ctx_t *user_example_ctx = user_example_get_ctx();
    char *ps8Buf = NULL;
    uint32_t u32BufSize = PER_STATUS_PROPERTY_LEN;
    uint32_t u32Offset = 0;

    #ifdef BLEWIFI_ALI_DEV_SCHED
    if(ptProp->u8Type == DEV_IND_TYPE_SCHED)
    {
        u32BufSize = PER_SCHED_PROPERTY_LEN;
    }
    #endif

    ...

    res = IOT_Linkkit_Report(user_example_ctx->master_devid, ITM_MSG_POST_PROPERTY,
        (unsigned char *)ps8Buf, u32Offset);

    EXAMPLE_TRACE("Post Property Message ID: %d, Len[%d]", res, u32Offset);

    user_post_add_needReply();
}
```

### 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，收到从阿里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 ali\_linkkitsdk\_impl.c 文件中的 user\_property\_set\_event\_handler() 函数实现。

Figure 17: LED 灯状态的接收

```
static int user_property_set_event_handler(const int devid, const char *request, const int request_len)
{
#ifdef BLEWIFI_ALI_DEV_SCHED
    //IoT_Property_t IoT_Property;
    cJSON *request_root = NULL, *item_LedLightSwitch = NULL;
    user_example_ctx_t *user_example_ctx = user_example_get_ctx();
    printf("Property Set Received, Devid: %d, Request: %s\r\n", devid, request);

    /* Parse Request */
    request_root = cJSON_Parse(request);
    if (request_root == NULL || !cJSON_IsObject(request_root)) {
        EXAMPLE_TRACE("JSON Parse Error");
        return -1;
    }
    /* start : Add you code to handle request data */

    /* Try To Find LedLightSwitch Property */
    item_LedLightSwitch = cJSON_GetObjectItem(request_root, "LedLightSwitch");
    if (item_LedLightSwitch != NULL) {
        EXAMPLE_TRACE("LedLightSwitch Enable : %d\r\n", item_LedLightSwitch->valueint);

        T_BleWifi_Ctrl_DevStatus tStatus = {0};

        tStatus.u8DevOn = item_LedLightSwitch->valueint;
        tStatus.u8LedOn = tStatus.u8DevOn;

        if(BleWifi_Ctrl_DevStatusSet(&tStatus))
        {
            EXAMPLE_TRACE("BleWifi_Ctrl_DevStatusSet fail\r\n");
        }
    }

    /* end : Add you code to handle request data */
    cJSON_Delete(request_root);

```

而 user\_property\_set\_event\_handler () 则是在 ali\_linkkit\_init() 初始化过程中注册。

Figure 18: Ali linkkit 回调函数的注册

```
IOT_RegisterCallback(ITE_PROPERTY_SET, user_property_set_event_handler);
IOT_RegisterCallback(ITE_PROPERTY_GET, user_property_get_event_handler);
```

#### 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，灯的开关动作由在 blewifi\_ctrl.c 文件中的 led\_set() 函数实现如下：

Figure 19: LED 灯状态的设置

```
static void led_set(uint8_t u8On, uint8_t u8Ind)
{
    if(u8On != g_tBleWifiDevStatus.u8LedOn)
    {
        E_GpioLevel_t tLevel = GPIO_LEVEL_LOW;
        //E_GpioLevel_t tLevel = GPIO_LEVEL_HIGH;

        g_tBleWifiDevStatus.u8LedOn = u8On;

        if(!g_u8AppLedBlinkEnable)
        {
            BLEWIFI_WARN("[%s %d] set LED[%u]\n", __func__, __LINE__, g_tBleWifiDevStatus.u8LedOn);
        }

        if(g_tBleWifiDevStatus.u8LedOn)
        {
            // active high
            tLevel = GPIO_LEVEL_HIGH;
            /*
            // active low
            tLevel = GPIO_LEVEL_LOW;
            */
        }

        Hal_Vic_GpioOutput(PER_LED_IO_PORT, tLevel);
    }
    else
    {
        BLEWIFI_INFO("[%s %d] LED already[%u]\n", __func__, __LINE__, g_tBleWifiDevStatus.u8LedOn);
    }

    return;
}
```

### 5.3. 阿里收发消息 buffer 的大小调整

本例程使用的阿里 SDK 库的版本是 3.0.1。它所在的目录是 APS\_PATCH\middleware\third\_party\aliyun\_3.0.1。在 src\mqtt\impl\iotx\_mqtt\_config.h 文件中有定义了 IOTX\_MC\_TX\_MAX\_LEN 和 IOTX\_MC\_RX\_MAX\_LEN 两个宏，分别用于指定发送和接收消息的 buffer 的最大长度。它们的默认值为 512。

Figure 20: Ali 收发 buffer size 的设置

```
77  #ifndef IOTX_MC_TX_MAX_LEN
78      #define IOTX_MC_TX_MAX_LEN          (512)
79  #endif
80
81  #ifndef IOTX_MC_RX_MAX_LEN
82      #define IOTX_MC_RX_MAX_LEN          (512)
83  #endif
```

用户在做二次开发，为设备规划好收发消息的主题和数据包的格式时，可根据实际情况调整这两个 buffer 的大小。每次修改过 buffer 大小后，需要重新编译阿里库，以使改动生效。

## 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。

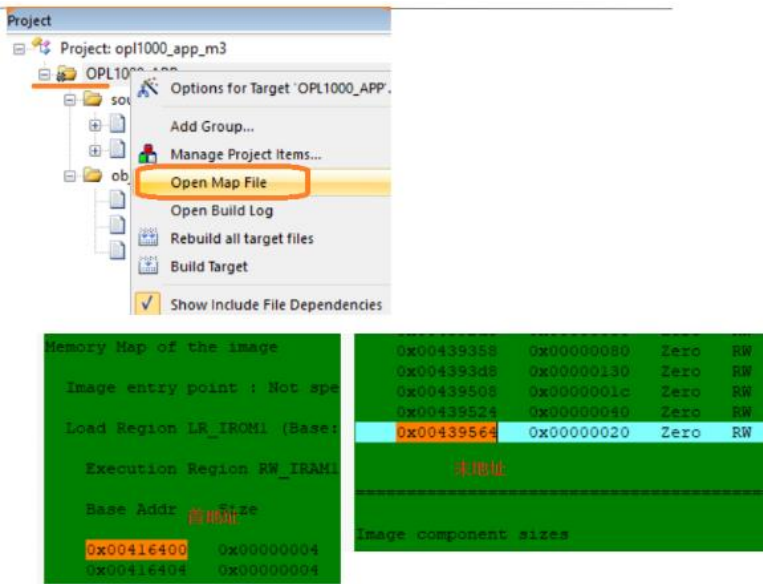
(1) 在调整 scatter file 之前，需要在 map 文件获取首末地址，方法如下：

- 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。

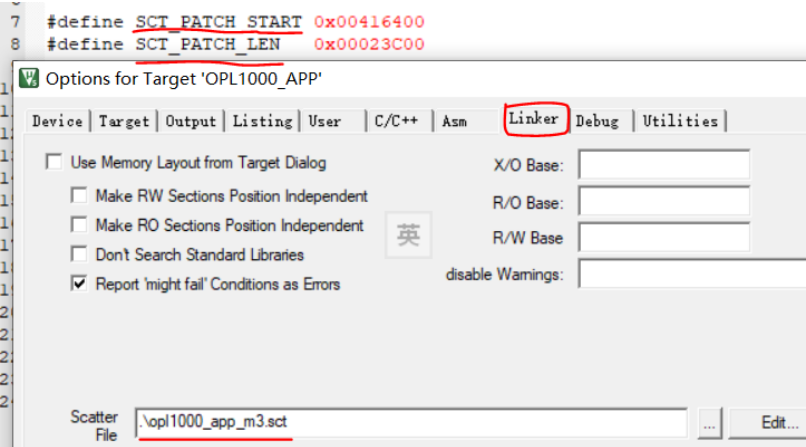


Figure 21: 从 map 文件中获取首末地址



- (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。

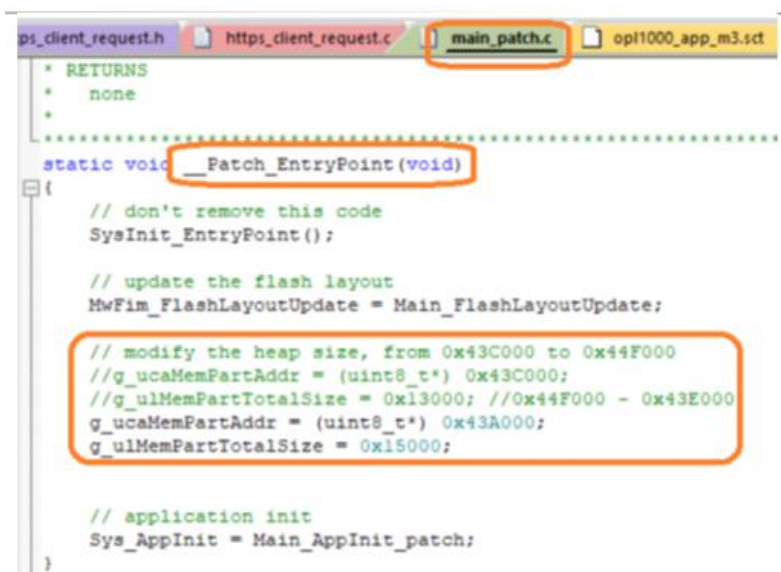
Figure 22: 更新 SCT file 中的 size



(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 23: 更新 heap size 相关的信息



```
ps_client_request.h | https_client_request.c | main_patch.c | opl1000_app_m3.sct
* RETURNS
* none
*
*****
static void __Patch_EntryPoint(void)
{
    // don't remove this code
    SysInit_EntryPoint();

    // update the flash layout
    MwFim_FlashLayoutUpdate = Main_FlashLayoutUpdate;

    // modify the heap size, from 0x43C000 to 0x44F000
    //g_ucaMemPartAddr = (uint8_t*) 0x43C000;
    //g_ulMemPartTotalSize = 0x13000; //0x44F000 - 0x43E000
    g_ucaMemPartAddr = (uint8_t*) 0x43A000;
    g_ulMemPartTotalSize = 0x15000;

    // application init
    Sys_AppInit = Main_AppInit_patch;
}
```

## 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 24: 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](mailto:sales@Opulinks.com)