

OPL1000

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

AWS 云透传应用说明文档



OPULINKS

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

Copyright © 2017-2020, Opulinks. All Rights Reserved.

OPL1000-AWS 云透传应用说明文档 | Version V1.0

版本纪录

日期	版本	更新内容
2020-02-11	1.0	初版

目录

1. 介绍	1
1.1. 文档应用范围	1
1.2. 缩略语	1
1.3. 参考文献	1
2. 工程构成和工作原理	3
2.1. 工程构成	3
2.2. 工作原理	4
3. 编译和运行 AWS 透传范例	6
3.1. 在 AWS 云端创建 IOT 设备	6
3.2. 编译 AWS 透传工程	8
3.3. 下载并执行 AWS 透传固件	11
3.4. 通过 AT 命令完成蓝牙配网	11
3.5. 通过 AT 命令和 AWS 云端通信	15
3.5.1. 连接 AWS 云并查询状态	15
3.5.2. 发布消息到特定主题	16
3.5.3. 订阅特定主题的消息	17
3.5.4. 启用和退出睡眠模式	18
4. 基于 AWS 透传范例开发应用	19
4.1. 修改 AWS IOT 设备模型	19
4.2. 基于已有范例增添功能	19
4.2.1. 连接 AWS 云的工作原理	20
4.2.2. MQTT 消息发布和订阅工作流程	21
4.2.3. Task Handler 处理机制	22
4.3. 工程 Heap Size 调整	23
4.4. 低功耗设置和验证	25
4.5. 扩充 AT 命令	26
4.6. 验证添加的功能	26
5. 进阶：AWS SDK 更新	29
5.1. AWS MQTT 收发 Buffer Size 调整	29
6. FAQ	30

图目录

FIGURE 1: 项目文件	3
FIGURE 2: 工作原理图.....	5
FIGURE 3: 创建 IOT 物联网设备	6
FIGURE 4: 获取事物的信息	7
FIGURE 5: 获取事物证书	7
FIGURE 6: KEIL C 编译结果.....	8
FIGURE 7: GCC 编译结果	9
FIGURE 8: PACK OPL1000.BIN 固件	10
FIGURE 9: PACK OPL1000_OTA.BIN 固件	10
FIGURE 10: 下载 OPL1000.BIN 固件	11
FIGURE 11: 切换为 BLEWIFI 模式	12
FIGURE 12: 写入事物信息到 FLASH	13
FIGURE 13: 重启 OPL1000.....	13
FIGURE 14: 开启蓝牙广播.....	14
FIGURE 15: 打开 OPL1000 APP	14
FIGURE 16: 连接 WIFI AP	15
FIGURE 17: OPL1000 获得 IP.....	15
FIGURE 18: 连接 AWS 云.....	16
FIGURE 19: 发布消息	16
FIGURE 20: 云平台上查看消息	17
FIGURE 21: 订阅主题	17
FIGURE 22: 发布消息	17
FIGURE 23: 查看终端消息	18
FIGURE 24: 切换睡眠模式.....	18

FIGURE 25: 更新事物属性..... 19

FIGURE 26: 初始化函数调用 20

FIGURE 27: MQTT 客户端初始化..... 20

FIGURE 28: 订阅和发布函数的实现 21

FIGURE 29: BLE 消息处理函数映射表 22

FIGURE 30: WIFI 消息处理函数映射表 22

FIGURE 31: BLEWIFI CONTROLLER 消息处理函数映射表 23

FIGURE 32: 获取首末地址..... 24

FIGURE 33: 修改 SCT_PATCH_LEN 值 24

FIGURE 34: 调整 HEAP SIZE 值..... 25

FIGURE 35: SMART SLEEP 相关的宏定义 25

FIGURE 36: AT 命令实现函数映射表 26

FIGURE 37: 定义订阅主题..... 26

FIGURE 38: 云平台上发布消息 27

FIGURE 39: 终端上接收消息 27

FIGURE 40: 云平台上接收消息 28

FIGURE 41: AWS 收发消息用的 BUFFER 大小..... 29

表目录

TABLE 1: 项目文件夹和内容 4

1. 介绍

1.1. 文档应用范围

本文档介绍了一个基于 OPL1000 A2 芯片以透传的方式连接 AWS 云实现物联网应用的范例。该范例包含蓝牙配网功能，OPL1000 作为从设备通过串口接收来自主设备的 AT 命令，实现连接 AWS 云，并通过预先定义的 MQTT Topic 实现信息包的订阅和发布。

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 空中下载技术
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>

Copyright © 2017-2020, Opulinks. All Rights Reserved.
OPL1000-AWS 云透传应用说明文档, UG1-07-1

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

访问连接: <https://github.com/Opulinks-Tech/OPL1000A2-Sensor-Device-Reference-Code-Aws-Cloud-with-MQTT/blob/master/doc/OPL1000-aws-key-app-tool-guide.pdf>

2. 工程构成和工作原理

透传，即透明传输（transparent transmission），指的是在通讯中不管传输的业务内容如何，只负责将传输的内容由源地址传输到目的地址，而不对业务数据内容做任何改变。

2.1. 工程构成

AWS 透传项目主要包含蓝牙配网，MQTT 处理和库文件等目录

Figure 1: 项目文件

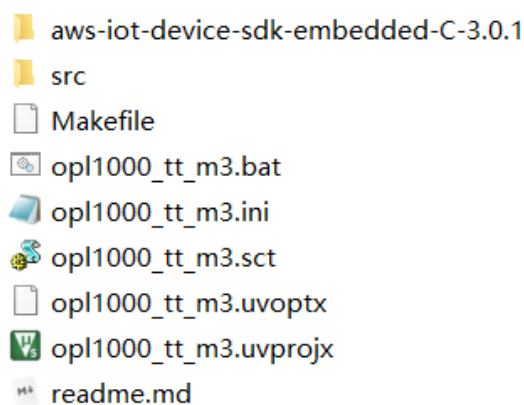


Table 1: 项目文件夹和内容

文件夹和文件	内容说明
aws-iot-device-sdk-embedded-C-3.0.1	包含 aws iot 的库文件
Makefile	用于以 gcc 方式编译的 makefile 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
src/iot_data	存放收发 iot 数据相关的代码
src/app_at_cmd.c	存放透传相关的 at 命令
opl1000_app_tt_m3.bat opl1000_app_tt_m3.ini opl1000_app_tt_m3.sct opl1000_app_tt_m3.uvoptx opl1000_app_tt_m3.uvprojx	编译透传工程文件。

2.2. 工作原理

AWS 透传项目主要包括：主控 MCU(主设备), 透传物联网模块 OPL1000(从设备) · 移动设备 (APP) · 和云端 (亚马逊云) 。

Figure 2: 工作原理图



其中，主控 MCU 可通过下发 at 命令控制 OPL1000 实现下述功能：

- 切换到 BLEWIFI 模式，`at+cwmode=4`；
- 更新 AWS 事物信息，`at+cloudinfo=<hostname>,<client_id>,<thing_name>` 和 `at+writefim=<file_id>,<index>,<data_len>`；
- 开启蓝牙广播，`at+blecast=1`；
- 连 AWS 云，`at+cloudconn`；
- 查询与 AWS 云的连接状态，`at+cloudconnstatus`；
- 发布数据到 AWS 云，`at+cloudpub=<qos>,<pub type>,<data>`
- 从 AWS 云端订阅和接收数据，`at+cloudsub=<qos>,<sub type>,<data>`
- 触发睡眠模式，`at+sleep=<sleep mode>,<I/O>`
- 通过 GPIO 唤醒 OPL1000

3. 编译和运行 AWS 透传范例

3.1. 在 AWS 云端创建 IOT 设备

为了验证 AWS 透传用例，用户需要先在 AWS 云上创建 IOT 物联网设备，示例如下：

Figure 3: 创建 IOT 物联网设备



关于如何创建 AWS IOT 设备模型的更多信息，请参考 [AWS key app tool 使用指南](#)。

在创建好事物后，用户需要在 AWS 云平台上获取事物的客户端标识符 (client id)，事物名称 (thing name)，主机名 (hostname) 如下图。

Figure 4: 获取事物的信息



并在如下界面获取,证书 (certificate) ,和私有密钥(private key)

Figure 5: 获取事物证书

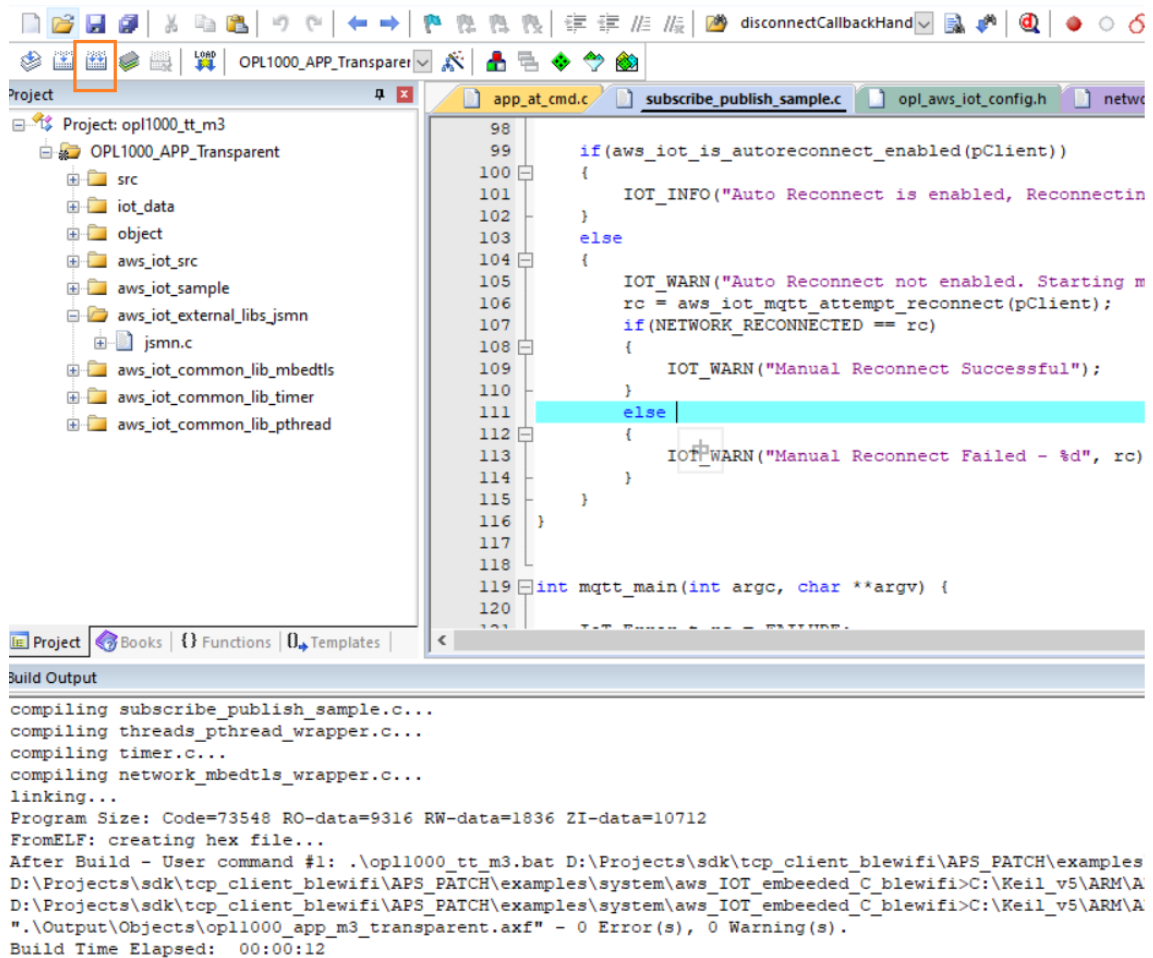


3.2. 编译 AWS 透传工程

AWS 透传工程可通过 Keil C (打开 opl1000_tt_m3.uvprojx 工程文件) 和 Gcc (makefile 文件) 编译两种方式编译，任取其中一种方式即可。编译成功后，会在 Output\Objects 目录下生成 opl1000_app_m3_transparent.bin。并将其 pack 为 OPL1000 固件如下。注意：在编译前，请确保已更新过 MQTT 主题和 IOT 设备的属性。

用 Keil C 编译的结果如下：

Figure 6: Keil C 编译结果



> aws_IOT_embeeded_C_blewifi > Output > Objects		
名称	Keil C	修改日期
mw_iim_default_group13_project.o		2020/2/13
mw_fim_default_group13_project.o		2020/2/13
network_mbedtls_wrapper.crf		2020/2/13
network_mbedtls_wrapper.d		2020/2/13
network_mbedtls_wrapper.o		2020/2/13
opl1000_app_m3_transparent.axf		2020/2/13
opl1000_app_m3_transparent.bin		2020/2/13

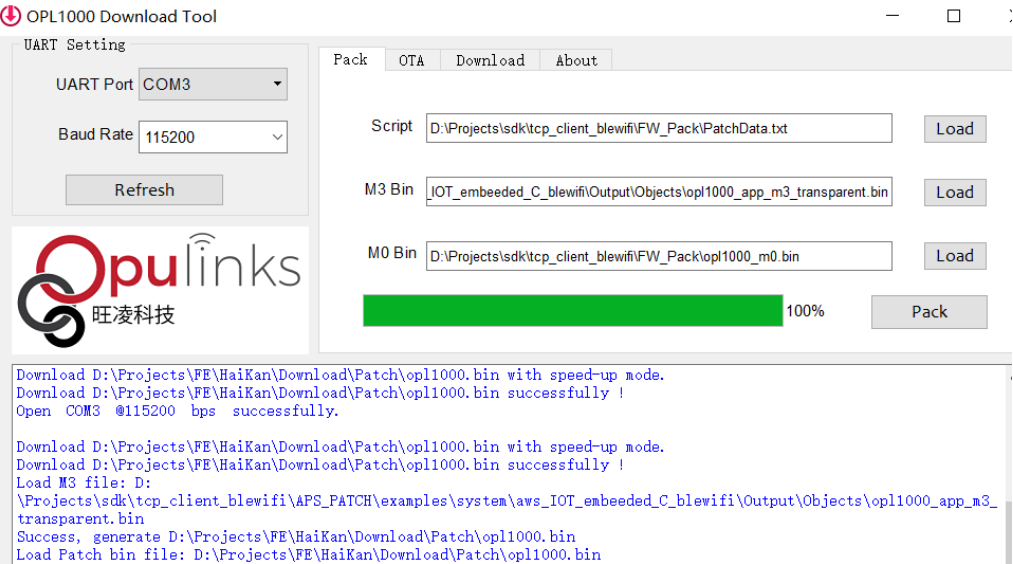
而通过 gcc 编译的结果如下：

Figure 7: gcc 编译结果

C:\Windows\System32\cmd.exe		
Microsoft Windows [版本 10.0.18362.592]		
(c) 2019 Microsoft Corporation. 保留所有权利。		
D:\Projects\sdk\tcp_client_blewifi\APS_PATCH\examples\system\aws_IOT_embeeded_C_blewifi>make		
arm-none-eabi-objcopy -O binary ./Output-GCC/opl1000_app_m3.elf ./Output-GCC/opl1000_app_m3.bin		
arm-none-eabi-objcopy -O binary ./Output-GCC/opl1000_app_m3_transparent.elf ./Output-GCC/opl1000_app_m3_transparent.bin		
D:\Projects\sdk\tcp_client_blewifi\APS_PATCH\examples\system\aws_IOT_embeeded_C_blewifi>dir		
> aws_IOT_embeeded_C_blewifi > Output-GCC		
名称		修改日期
opl1000_app_m3.bin		20
opl1000_app_m3.elf		20
opl1000_app_m3.map		20
opl1000_app_m3_transparent.bin		20
opl1000_app_m3_transparent.elf		20

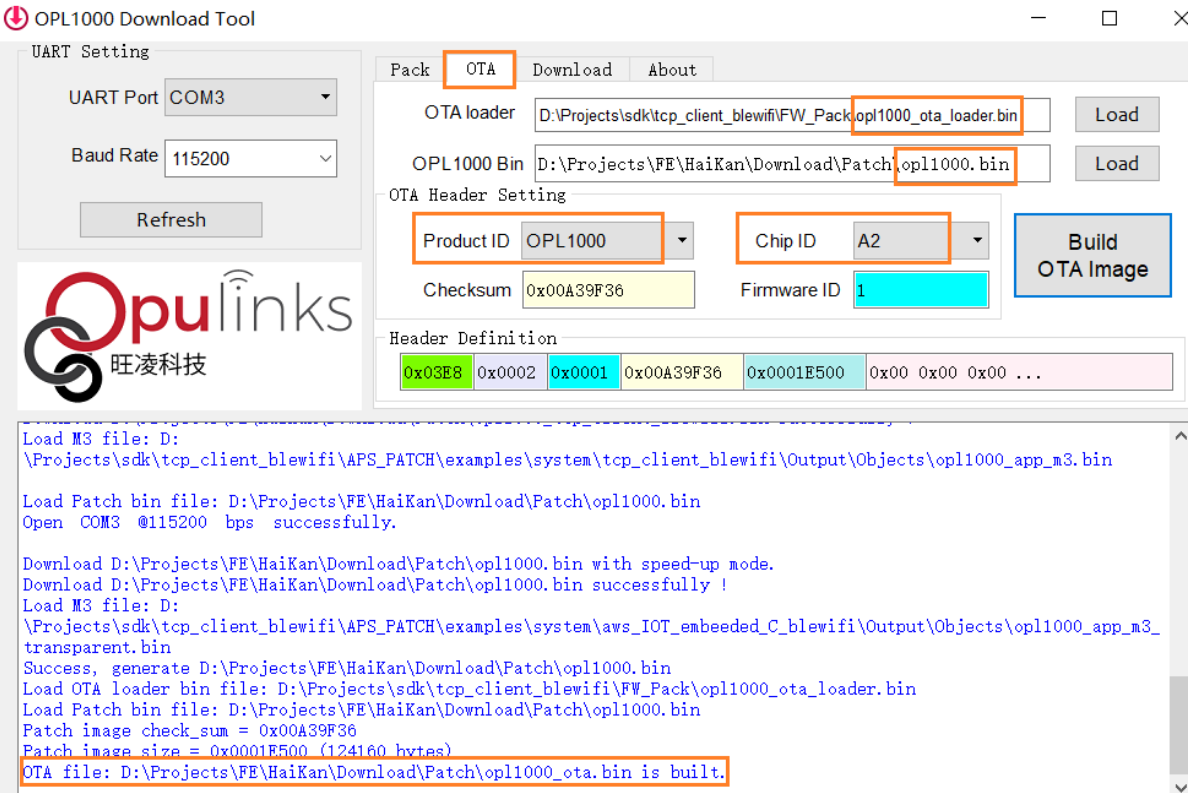
在 download tool 的 pack 界面上，选择相应的 opl1000_app_m3_transparent.bin 文件，将它 pack 成 OPL1000.bin 如下。

Figure 8: pack opl1000.bin 固件



在 pack 好 opl1000.bin 后，用户可根据需要继续 pack 相应的 OTA bin 如下

Figure 9: pack opl1000_ota.bin 固件

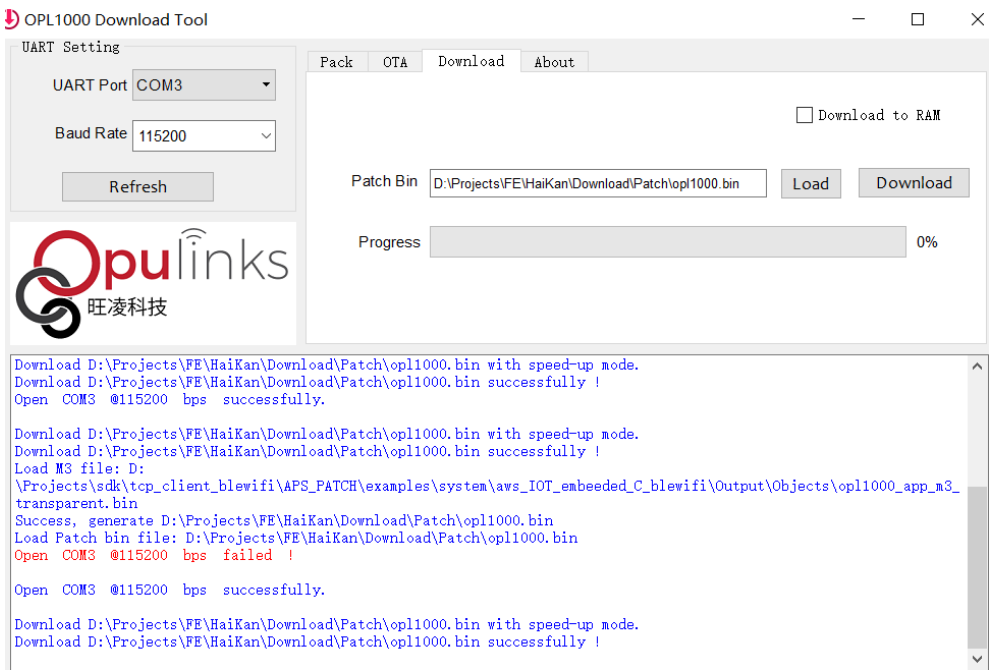


其中，Product ID, Chip ID 需要根据实际使用的模块做相应的选择。

3.3. 下载并执行 AWS 透传固件

在编译并 pack 好 OPL1000 固件后，用 download tool 下载固件到 OPL1000 模块如下：

Figure 10: 下载 OPL1000.bin 固件



关于 download tool 的更多信息，请参考 Download 工具使用指南 [OPL1000-patch-download-tool-user-guide.pdf](#)。

3.4. 通过 AT 命令完成蓝牙配网

在下载完成后，用 AT 命令实现蓝牙配网过程如下：

- 1. 用 at+cwmode=4 命令切换到 BLEWIFI 模式 并重启 OPL1000 模块；

Figure 11: 切换为 BLEWIFI 模式

```
>at+cwmode=4  
  
OK  
  
>at+rst  
  
OK
```

2. 写入在 3.1 节中获取的事物的信息到 flash. 这些信息包括事物的客户端标识符 (client id) , 事物名称 (thing name) , 主机名 (hostname) , 证书 (certificate) , 和私有密钥 (private key) .

at+cloudinfo=<hostname>,<client_id>,<thing_name>

at+writefim=<file_id>,<index>,<data_len>

Figure 12: 写入事物信息到 flash

```
COM104 - Tera Term VT
File Edit Setup Control Window Help

>at+cloudinfo=a14yfgdb6z35u-ats.iot.cn-northwest-1.amazonaws.com.cn,sh_sdk_tt_test1,sh_sdk_tt_test1
OK
      at+cloudinfo=<hostname>,<client_id>,<thing_name>

>at+writefim=0x01030002,0,1679
fetch data  写入private key
OK

write...

OK

>at+writefim=0x01030003,0,1220
fetch data  写入certificate
OK

write...

OK
```

3. 重启 OPL1000，使上面写入 flash 的事物信息生效;

Figure 13: 重启 OPL1000

```
>at+rst
OK
```

4. 开启蓝牙广播，并在手机端打开 OPL1000 APP 进行蓝牙配网，选择合适的 AP，输入密码后显示成功连上 WIFI AP.

Figure 14: 开启蓝牙广播

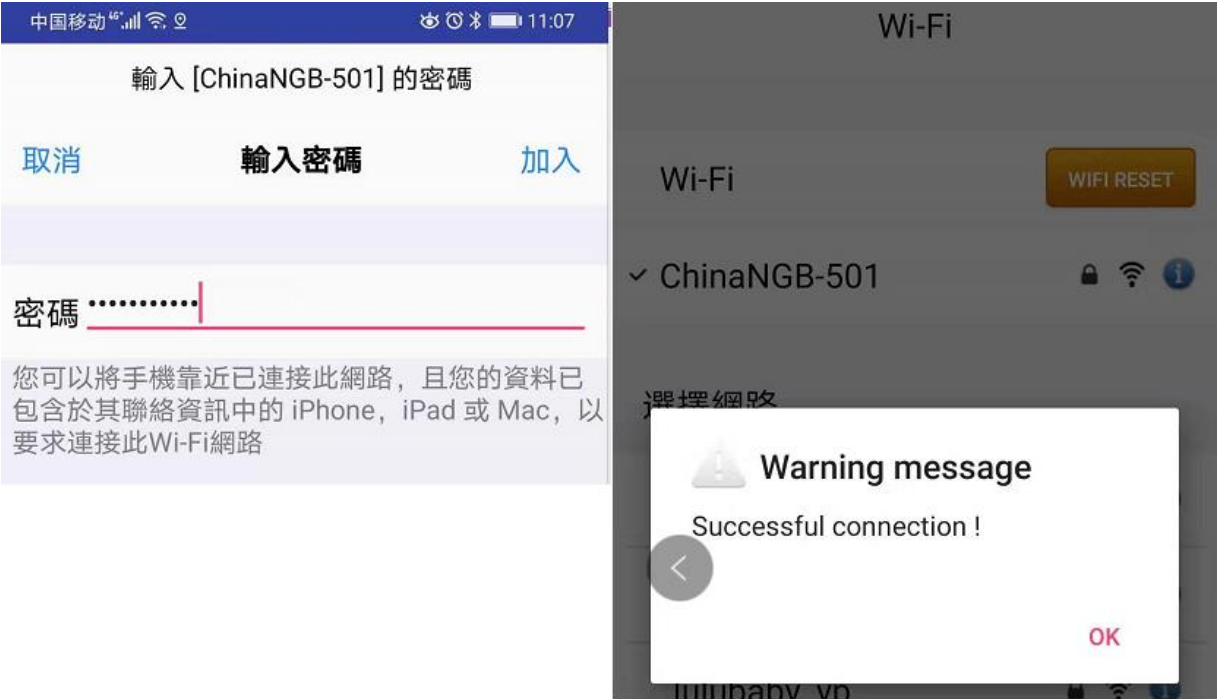
```
>at+blecast=1
+BLECAST:1,0
OK
```

Figure 15: 打开 OPL1000 APP



选择合适的 AP，输入密码后显示成功

Figure 16: 连接 WIFI AP



连上 AP 后，OPL1000 得到 IP

Figure 17: OPL1000 获得 IP



3.5. 通过 AT 命令和 AWS 云端通信

在 OPL1000 配网成功并获得 IP 后，即可开始跟 AWS 云端建立连接并收发消息。

3.5.1. 连接 AWS 云并查询状态

at+cloudconn 命令用于和 AWS 云端建立连接；

at+cloudconnstatus? 命令用于查询和 AWS 云端的连接状态。

Copyright © 2017-2020, Oplinks. All Rights Reserved.

OPL1000-AWS 云透传应用说明文档, UG1-07-1

Figure 18: 连接 AWS 云

```
>at+cloudconn
OK      AT port

MwFim Read AWS_CERT_PEM OK      APS port

sh_sdk_tt_test1 connect to al4yfgdb6z35u-ats.iot.cn-northwest-1.amazonaws.com.cn

-->RX Loop

>at+cloudconnstatus?
+COLUDCONNSATUS: CLOUD CONNECTED

OK      AT port

-->RX Loop
```

3.5.2. 发布消息到特定主题

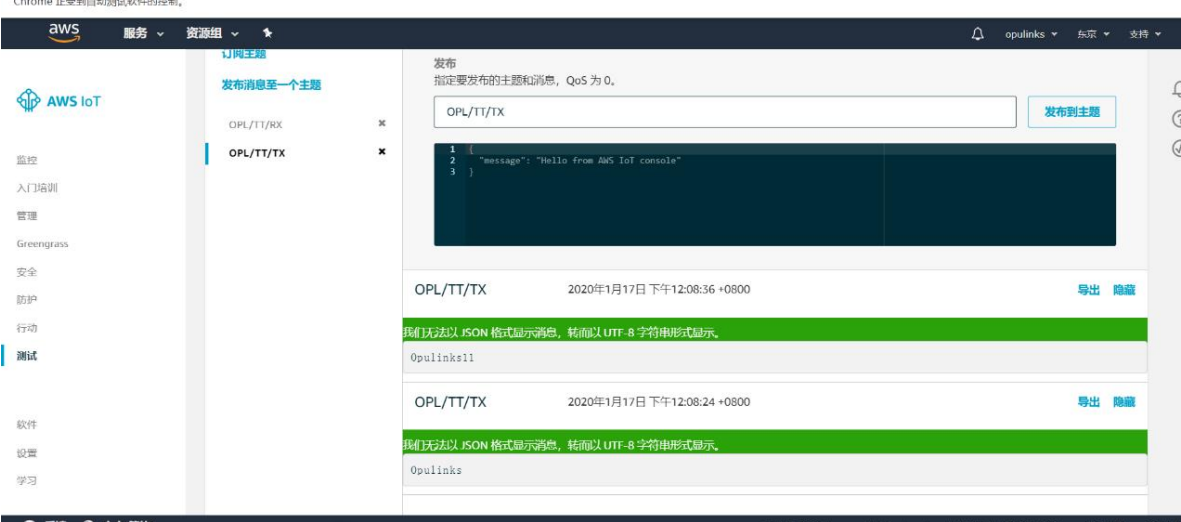
用 `at+cloudpub=0,OPL/TT/TX,Opulinks` 命令发布 Opulinks 到 OPL/TT/TX 这个主题。

Figure 19: 发布消息

```
>at+cloudpub=0,OPL/TT/TX,Opulinks
OK
at+cloudpub=0,OPL/TT/TX,Opulinks
OK
at+cloudpub=0,OPL/TT/TX,Opulinks11
OK
```

在 AWS 云平台上即可收到该消息。

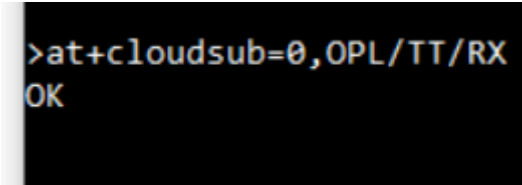
Figure 20: 云平台上查看消息



3.5.3. 订阅特定主题的消息

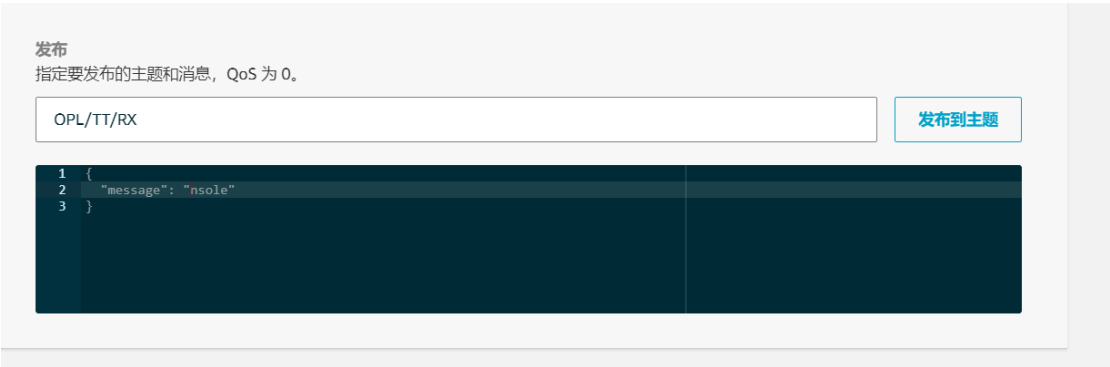
用 `at+cloudsub=0,OPL/TT/RX` 命令订阅 OPL/TT/RX 这个主题的消息

Figure 21: 订阅主题



在 AWS 云平台上发布消息到 OPL/TT/RX 这个主题

Figure 22: 发布消息



在 OPL1000 终端接收到消息如下

Figure 23: 查看终端消息

```
+RECVFROMSRV
Topic:OPL/TT/RX
Data:{
  "message": "nsole"
}
```

3.5.4. 启用和退出睡眠模式

在连上 AWS 云后，用 at+sleep=1,23 命令进入睡眠模式。在把 IO23 拉高后又可唤醒 OPL1000.

Figure 24: 切换睡眠模式

COM3 - Tera Term VT	COM104 - Tera Term VT
在睡眠模式下	>at+cloudconn
	OK
<--RX yield	开启睡眠模式
<--RX Loop	>at+sleep=1,23
-->RX Loop	
-->RX yield	OK
唤醒后	将io23拉高后唤醒OPL1000
<--RX yield	AT IO ISR invoked, io_num: 23
<--RX Loop	

4. 基于 AWS 透传范例开发应用

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 透传范例应用开发过程中需要改动的代码和常见问题的处理方法。

4.1. 修改 AWS IOT 设备模型

在编辑事物界面，可以增添或修改事物的属性如下。

Figure 25: 更新事物属性

付安全应用于此物品

使用物品类型可通过为具有相同类型的物品提供一致的注册表数据来简化设备管理。类型可为物品提供一组常见属性和一个说明，这些属性描述设备的身份和功能。

物品类型

OPL1000A2

创建类型

设置可搜索的物品属性(可选)

为这些属性中的一个或多个属性输入值，以便您可以在注册表中搜索您的物品。

该物品类型没有可搜索的属性

设置不可搜索的物品属性(可选)

您可以使用物品属性来描述设备的身份和功能。

属性键	值	
attr_1	val_1	清除
<div>添加另一个</div>		

取消

更新

关于如何修改 AWS IOT 设备模型的更多信息，请参考 [AWS key app tool 使用指南](#)。

4.2. 基于已有范例增添功能

在现有 AWS 透传范例的基础上增添功能主要涉及以下几个方面：

4.2.1. 连接 AWS 云的工作原理

在配网成功并获得 IP (等到 BLEWIFI_CTRL_EVENT_BIT_GOT_IP 事件) 后 , OPL1000 会在 `iot_data.c` 文件中调用 AWS IOT 库提供的接口函数 `mqtt_main()` 实现初始化并连接 AWS 云。

Figure 26: 初始化函数调用

```
// do the rx behavior
while (1)
{
    if (true == BleWifi_Ctrl_EventStatusWait(BLEWIFI_CTRL_EVENT_BIT_GOT_IP, 0xFFFFFFFF))
    {
        BleWifi_Wifi_SetDTIM(0);
        #ifndef __IOT_AWS_IT
        mqtt_main(NULL, NULL);
        #else
        if (true == BleWifi_Ctrl_EventStatusWait(CAN_DO_IOT_RX, 0xFFFFFFFF))
        {
            mqtt_main(NULL, NULL);
        }
        #endif
    }
}
```

`mqtt_main()` 函数在 `subscribe_publish_sample.c` 文件中实现 , 从 FIM 中读取 AWS 事物相关的信息 , 对 AWS IOT 做初始化并和 AWS 云建立连接。

Figure 27: mqtt 客户端初始化

```
mqttInitParams.enableAutoReconnect = false; // We enable this later below
mqttInitParams.pHostURL = g_tAWSDeviceInfo.host;
mqttInitParams.port = port;
mqttInitParams.mqttCommandTimeout_ms = 5000;
mqttInitParams.tlsHandshakeTimeout_ms = 20000;
mqttInitParams.isSSLHostnameVerify = true;
mqttInitParams.disconnectHandler = disconnectCallbackHandler;
mqttInitParams.disconnectHandlerData = NULL;

uint8_t ubaMacAddr[6];

// get the mac address from flash
wifi_config_get_mac_address(WIFI_MODE_STA, ubaMacAddr);

memset(SubscribeTopic, 0x00, sizeof(SubscribeTopic));
sprintf(SubscribeTopic, SUBSCRIBE_TOPIC, ubaMacAddr[3], ubaMacAddr[4], ubaMacAddr[5]);

rc = aws_iot_mqtt_init(&client, &mqttInitParams);
if (SUCCESS != rc)
```

```

connectParams.keepAliveIntervalInSec = 120;
connectParams.isCleanSession = true;
connectParams.MQTTVersion = MQTT_3_1_1;
connectParams.pClientID = g_tAWSDeviceInfo.client_id;
connectParams.clientIDLen = (uint16_t) strlen(g_tAWSDeviceInfo.client_id);
connectParams.isWillMsgPresent = false;

BleWifi_Ctrl_EventStatusSet(BLEWIFI_CTRL_EVENT_BIT_IOT_INIT, false);

IOT_INFO("%s connect to %s:%d", connectParams.pClientID, mqttInitParams.pHostURL, mqttInitParams.port)

rc = aws_iot_mqtt_connect(&client, &connectParams);

```

4.2.2. MQTT 消息发布和订阅工作流程

MQTT 消息发布和订阅分别由定义在 `aws_iot_mqtt_client_publish.c` 和 `aws_iot_mqtt_client_subscribe.c` 文件中的以下接口函数 `aws_iot_mqtt_publish()` 和 `aws_iot_mqtt_subscribe()` 实现，并在 `app_at_cmd.c` 文件的 AT 命令有调用，用户可根据需要进行修改。

Figure 28: 订阅和发布函数的实现

```

at_command_t g_atAppAtCmd[] =
{
    { "at+readfim",          app_at_cmd_sys_read_fim,          "Read FIM data" },
    { "at+writefim",         app_at_cmd_sys_write_fim,         "Write FIM data" },
    { "at+dtim",             app_at_cmd_sys_dtim_time,         "Wifi DTIM" },
#ifdef IOT_AWS_IT
    { "at+cloudinfo",        app_at_cmd_sys_aws_dev,           "update aws device info" },
    { "at+awskey",           app_at_cmd_sys_aws_key,           "update aws private key" },
    { "at+awscert",          app_at_cmd_sys_aws_cert,          "update aws certificate" },
    { "at+cloudconn",        app_at_cmd_IOT_connect_server,    "connect aws IOT server" },
    { "at+clouddisconnect",  app_at_cmd_IOT_disconnect_server, "disconnect aws IOT server" },
    { "at+cloudconnstatus",  app_at_cmd_IOT_connect_status,    "disconnect aws IOT server" },
    { "at+cloudpub",         app_at_cmd_IOT_publish_data,      "publish data to aws IOT server" },
    { "at+cloudsub",         app_at_cmd_IOT_sub_topic,         "subscribe topic" },
    { "at+cloudunsub",       app_at_cmd_IOT_unsub_topic,       "un-subscribe topic" },
#endif
};

paramsQOS1.qos = (QOS_type == QOS1) ? QOS1 : QOS0;
paramsQOS1.payload = argv[3];
paramsQOS1.isRetained = 0;
paramsQOS1.payloadLen = strlen(argv[3]);

if (true == BleWifi_Ctrl_EventStatusWait(CAN_DO_IOT_DO_JOB, YIELD_TIMEOUT*2))
{
    BleWifi_Ctrl_EventStatusSet(CAN_DO_IOT_DO_JOB, false);
    rc = aws_iot_mqtt_publish(&client, PubTopic, strlen(PubTopic), &paramsQOS1);
    //rc = aws_publish_wrapper(PubTopic, strlen(PubTopic), &paramsQOS1);
}

```

```
strcpy(SubscribeTopic, argv[2]);

printf("Subscribing topic : %s \n", SubscribeTopic);

if (true == BleWifi_Ctrl_EventStatusWait(CAN_DO_IOT_DO_JOB , YIELD_TIMEOUT*2))
{
    BleWifi_Ctrl_EventStatusSet(CAN_DO_IOT_DO_JOB,false);
    //rc = aws_iot_mqtt_subscribe(&client, SubscribeTopic, strlen(SubscribeTopic), (QOS_type == QOS1) ? QOS1 : QOS0, iot_subscribe_options);
    rc = aws_iot_mqtt_subscribe(&client, SubscribeTopic, strlen(SubscribeTopic), (QOS_type == QOS1) ? QOS1 : QOS0, NULL, NULL);
    BleWifi_Ctrl_EventStatusSet(CAN_DO_IOT_DO_JOB,true);
}
```

4.2.3. Task Handler 处理机制

AWS 透传范例启用了三个任务处理句柄

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，在 blewifi_data.c 文件中定义了一组消息处理函数如下：

Figure 29: BLE 消息处理函数映射表

```
static T_BleWifi_Ble_ProtocolHandlerTbl g_tBleProtocolHandlerTbl[] =
{
    {BLEWIFI_REQ_SCAN,          BleWifi_Ble_ProtocolHandler_Scan},
    {BLEWIFI_REQ_CONNECT,       BleWifi_Ble_ProtocolHandler_Connect},
    {BLEWIFI_REQ_DISCONNECT,     BleWifi_Ble_ProtocolHandler_Disconnect},
    {BLEWIFI_REQ_RECONNECT,      BleWifi_Ble_ProtocolHandler_Reconnect},
    {BLEWIFI_REQ_READ_DEVICE_INFO, BleWifi_Ble_ProtocolHandler_ReadDeviceInfo},
    {BLEWIFI_REQ_WRITE_DEVICE_INFO, BleWifi_Ble_ProtocolHandler_WriteDeviceInfo},
    {BLEWIFI_REQ_WIFI_STATUS,    BleWifi_Ble_ProtocolHandler_WifiStatus},
    {BLEWIFI_REQ_RESET,          BleWifi_Ble_ProtocolHandler_Reset},

    #if (BLE_OTA_FUNCTION_EN == 1)
    {BLEWIFI_REQ_OTA_VERSION,     BleWifi_Ble_ProtocolHandler_OtaVersion},
    {BLEWIFI_REQ_OTA_UPGRADE,     BleWifi_Ble_ProtocolHandler_OtaUpgrade},
    #endif
};
```

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。在 blewifi_wifi_api.c 文件中定义了一组消息处理函数如下，每条记录对应一个消息和消息处理函数的映射：

Figure 30: WIFI 消息处理函数映射表

```
static T_BleWifi_Wifi_EventHandlerTbl g_tWifiEventHandlerTbl[] =
{
    {WIFI_EVENT_STA_START,      BleWifi_Wifi_EventHandler_Start},
    {WIFI_EVENT_STA_CONNECTED,  BleWifi_Wifi_EventHandler_Connected},
    {WIFI_EVENT_STA_DISCONNECTED, BleWifi_Wifi_EventHandler_Disconnected},
    {WIFI_EVENT_SCAN_COMPLETE,  BleWifi_Wifi_EventHandler_ScanComplete},
    {WIFI_EVENT_STA_GOT_IP,     BleWifi_Wifi_EventHandler_GotIp},
    {WIFI_EVENT_STA_CONNECTION_FAILED, BleWifi_Wifi_EventHandler_ConnectionFailed},
    {0xFFFFFFFF,                NULL}
};
```

3. BLEWIFI controller Handler

BLEWIFI controller Handler 是跟 OPL1000 BLEWIFI controller 相关的功能。blewifi_ctrl.c 文件中定义了一组消息处理函数如下，每条记录对应一个消息和消息处理函数的映射：

Figure 31: BLEWIFI controller 消息处理函数映射表

```
static T_BleWifi_Ctrl_EvtHandlerTbl g_tCtrlEvtHandlerTbl[] = 每个消息对应一个消息处理函数
{
    {BLEWIFI_CTRL_MSG_BLE_INIT_COMPLETE, BleWifi_Ctrl_TaskEvtHandler_BleInitComplete},
    {BLEWIFI_CTRL_MSG_BLE_ADVERTISING_CFM, BleWifi_Ctrl_TaskEvtHandler_BleAdvertisingCfm},
    {BLEWIFI_CTRL_MSG_BLE_ADVERTISING_EXIT_CFM, BleWifi_Ctrl_TaskEvtHandler_BleAdvertisingExitCfm},
    {BLEWIFI_CTRL_MSG_BLE_ADVERTISING_TIME_CHANGE_CFM, BleWifi_Ctrl_TaskEvtHandler_BleAdvertisingTimeChangeCfm},
    {BLEWIFI_CTRL_MSG_BLE_CONNECTION_COMPLETE, BleWifi_Ctrl_TaskEvtHandler_BleConnectionComplete},
    {BLEWIFI_CTRL_MSG_BLE_CONNECTION_FAIL, BleWifi_Ctrl_TaskEvtHandler_BleConnectionFail},
    {BLEWIFI_CTRL_MSG_BLE_DISCONNECT, BleWifi_Ctrl_TaskEvtHandler_BleDisconnect},
    {BLEWIFI_CTRL_MSG_BLE_DATA_IND, BleWifi_Ctrl_TaskEvtHandler_BleDataInd},

    {BLEWIFI_CTRL_MSG_WIFI_INIT_COMPLETE, BleWifi_Ctrl_TaskEvtHandler_WifiInitComplete},
    {BLEWIFI_CTRL_MSG_WIFI_SCAN_DONE_IND, BleWifi_Ctrl_TaskEvtHandler_WifiScanDoneInd},
    {BLEWIFI_CTRL_MSG_WIFI_CONNECTION_IND, BleWifi_Ctrl_TaskEvtHandler_WifiConnectionInd},
    {BLEWIFI_CTRL_MSG_WIFI_DISCONNECTION_IND, BleWifi_Ctrl_TaskEvtHandler_WifiDisconnectionInd},
    {BLEWIFI_CTRL_MSG_WIFI_GOT_IP_IND, BleWifi_Ctrl_TaskEvtHandler_WifiGotIpInd},
    {BLEWIFI_CTRL_MSG_WIFI_AUTO_CONNECT_IND, BleWifi_Ctrl_TaskEvtHandler_WifiAutoConnectInd},
}
```

4.3. 工程 Heap Size 调整

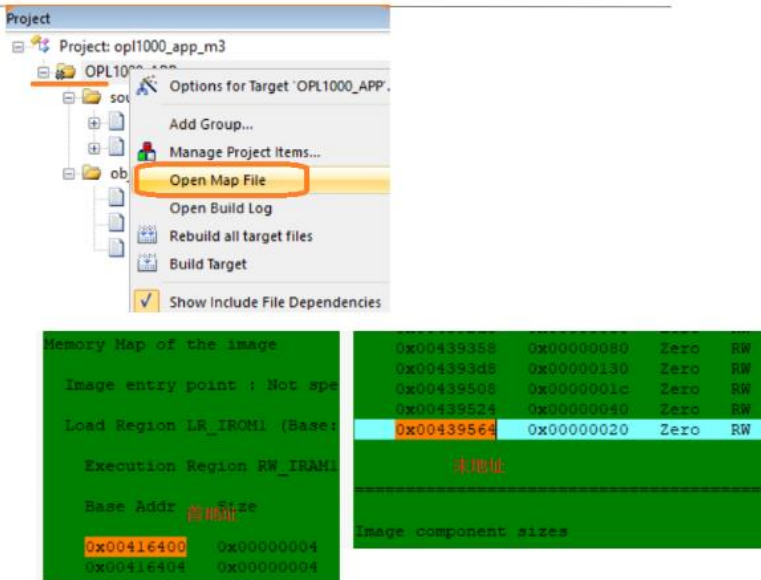
随着功能的不断加入，应用程序可能会碰到由于代码的不断增加，导致空间不够而引起的宕机问题.用户可以参考本节的内容，对 scatter file, heap size 做适当调整，尝试着解决内存不够问题。

(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。

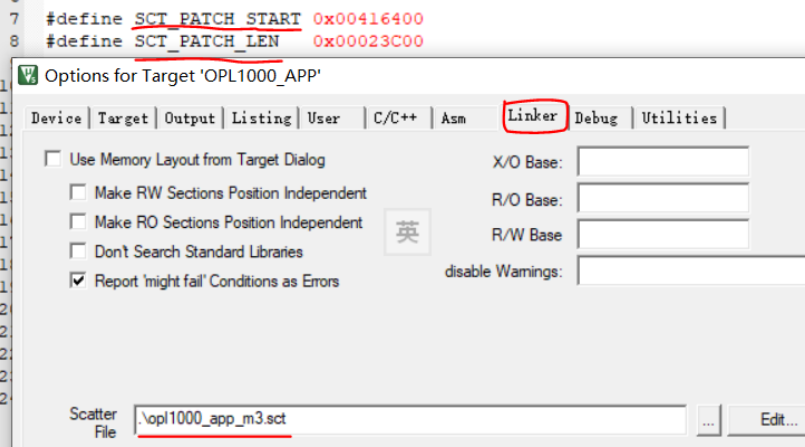
在该例子中，首地址是： 0x00416400 ，末地址是： 0x00439564 。如下图所示。对末地址以 4k 或其它值(1k,512B 等)为单位向上取整，为 0x0043a000.

Figure 32: 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 。

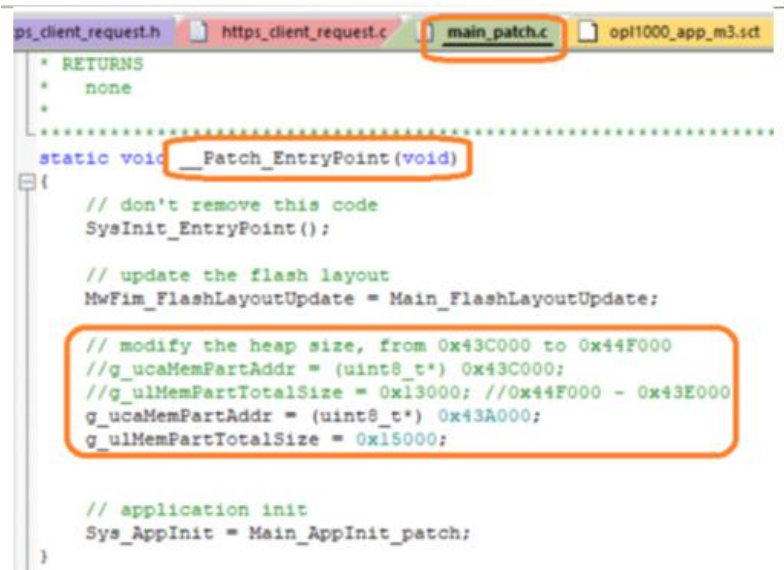
Figure 33: 修改 SCT_PATCH_LEN 值



- (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主文件 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 34: 调整 heap size 值



```
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_ucMemPartAddr = (uint8_t*) 0x43C000;
    //g_ulMemPartTotalSize = 0x13000; //0x44F000 - 0x43E000
    g_ucMemPartAddr = (uint8_t*) 0x43A000;
    g_ulMemPartTotalSize = 0x15000;

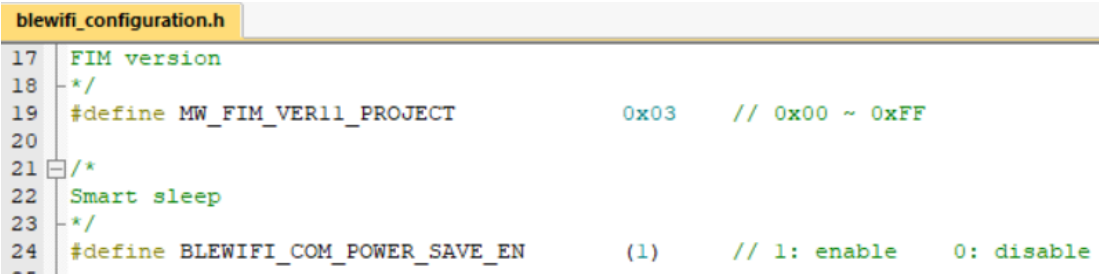
    // application init
    Sys_AppInit = Main_AppInit_patch;
}
```

4.4. 低功耗设置和验证

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 35: 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) ;
```

4.5. 扩充 AT 命令

4.6. 验证添加的功能

在更新了 IOT 设备的功能后，用户可借助 AWS 云平台的测试环境来验证。方法如下：

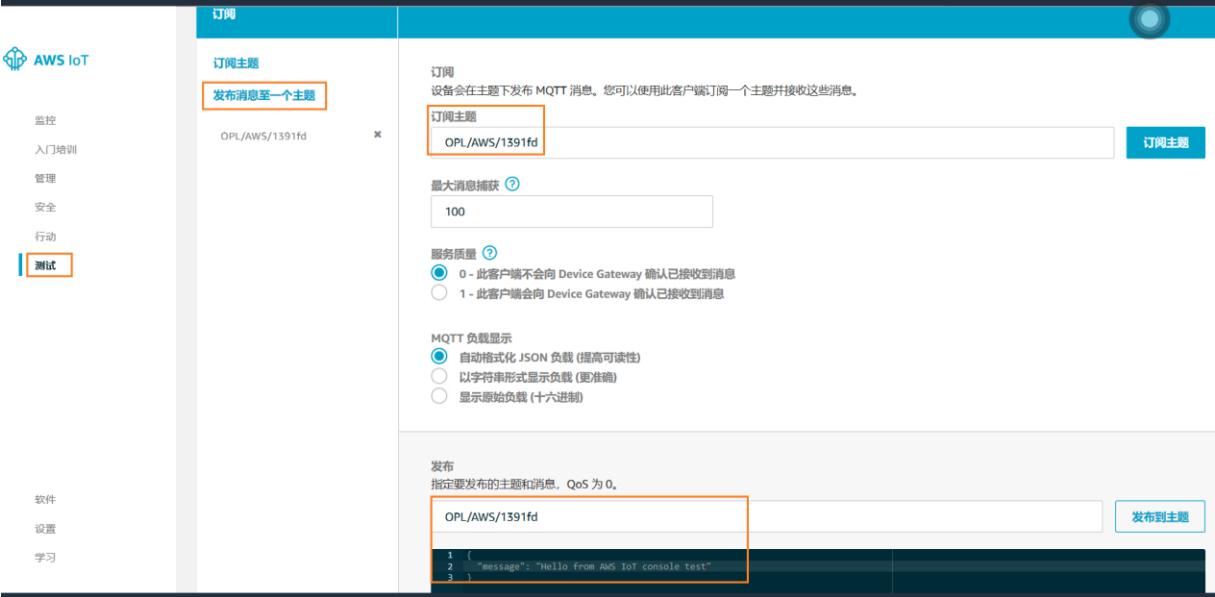
1. 订阅 OPL/AWS/1391fd 主题(该主题在 subscribe_publish_sample.c 文件中定义)，并发布一条消息。

Figure 37: 定义订阅主题

```
#include "cmsis_os.h"

#define HOST_ADDRESS_SIZE 255
#define SUBSCRIBE_TOPIC "OPL/AWS/%02x%02x%02x"
```


Figure 38: 云平台上发布消息



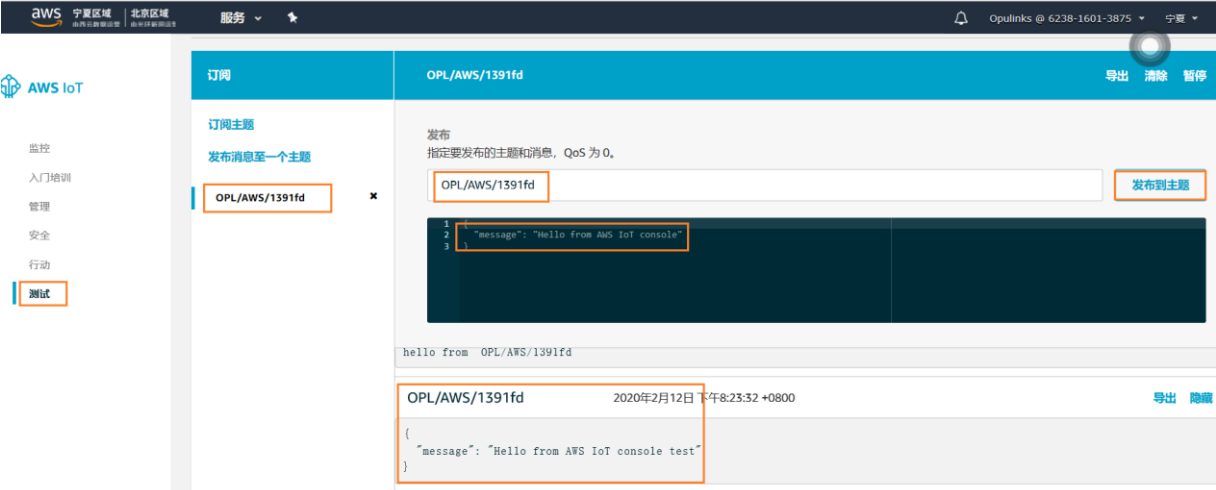
在 OPL1000 终端可收到消息如下

Figure 39: 终端上接收消息

```
-->RX yield
Subscribe callback
OPL/AWS/1391fd {
  "message": "Hello from AWS IoT console test"
}
<--RX yield
-->TX
```

在 AWS 云平台上接收到消息如下

Figure 40: 云平台上接收消息



5. 进阶：AWS SDK 更新

5.1. AWS MQTT 收发 Buffer Size 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 41: 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
```

6. FAQ

N/A

CONTACT

sales@Opulinks.com