

OPL1000

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

SDK 开发者指南



OPULINKS

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

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

OPL1000-SDK 开发者指南 | Version V2.0

版本纪录

日期	版本	更新内容
2018-05-11	0.1	- 初版
2018-05-15	0.2	- 更新章节 2.1 - 加入章节 6.4 介绍输出 log 檔设定
2018-05-23	0.3	- 修改章节 6.4
2018-06-01	0.4	- 更新章节 2 和章节 3.2
2018-06-06	0.5	- 更新章节 2.1, 章节 6 和 7.
2018-07-12	0.6	- 更新章节 2.1 SDK 目录结构和范例条列列表
2018-08-03	0.7	- 更新章节 2, 6,和 7 符合 A1 芯片
2018-08-17	0.8	- 更新章节 2 由于 SDK 组件更新.
2018-10-22	0.9	- 更新章节 2 --Table2, 和 Table4
2018-11-14	1.0	- 更新章节 2 由于多设备下载工具更新
2019-02-14	1.1	- 更新章节 2 --Table 4 由于范例 ancs 已更新
2019-03-05	1.2	- 更新工具内容
2019-03-13	1.3	- 更新章节 2 --Table2 加入 opl1000_m0_ldo.bin
2019-05-30	1.4	- 更新 Table 2 和 Table 3 加入 FW_Pack
2019-06-12	1.5	- 更新 Table 2 更新 bin 檔的描述
2019-08-07	1.6	- 加入设置 GUN ARM GCC 工具链和构建工具章节.
2019-10-12	1.7	- 增加调整 heap size 章节.
2020-02-19	2.0	- 加入章节 2.3, 和 2.4,针对建立 SDK 库和 Ali SDK 库 - 更新章节 8, 基于 Ali TT 应用范例.

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. OPL1000 SDK 软件包	2
2.1. 目录结构	2
2.2. SDK 发布包使用	6
2.3. SDK 库的更新和编译	8
2.4. Ali SDK 库的编译	11
3. 应用程序开发流程	14
3.1. IDE 在线调试开发模式	14
3.2. 串口调试开发模式	15
4. GNU ARM GCC 验证环境的搭建	16
4.1. 工具链和构建工具的下载和安装	16
4.2. 使用 GNU ARM GCC 构建工程	17
5. 嵌入式操作系统	18
6. 外设参数配置	19
7. 应用开发基本配置	22
7.1. Keil uVision 工程配置	22
7.2. 工程预览	23
7.3. Main 函数入口	24
7.4. Log 输出设置	24
7.5. 调整 heap size	27
8. 应用举例 – 基于 Ali 云 透传应用	30
8.1. Ali 云透传工程介绍	30
8.2. 更新工程配置或文件	30
8.3. 根据需要修改文件	31

图目录

FIGURE 1: OPL1000 SDK 发布包目录	2
FIGURE 2 : SDK 库工程目录结构	8
FIGURE 3 : SDK 库工程文件	9
FIGURE 4 : 打开管理工程文件窗口	9
FIGURE 5 : 添加或删除工程文件	10
FIGURE 6 : 构建 SDK 工程文件	11
FIGURE 7 : 阿里 SDK 库的目录结构	12
FIGURE 8 : 阿里 SDK 库工程文件	12
FIGURE 9 : 构建 SDK 工程	13
FIGURE 10: IDE 在线开发模式	14
FIGURE 11: 串口调试开发模式	15
FIGURE 12 : 设置环境变量	17
FIGURE 13: 使用 GNU ARM GCC 构建 HELLOWORLD 工程	17
FIGURE 14: CMSIS RTOS 架构	18
FIGURE 15: 外设配置	19
FIGURE 16: IO 配置	20
FIGURE 17: DEVICE 选择	22
FIGURE 18: SCATTER FILE	23
FIGURE 19: DEBUG 设置	23
FIGURE 20: HELLO WORLD 工程源码结构	23
FIGURE 21: TRACER 命令	25

FIGURE 22 : 在 MAP 文件获取首末地址.....27

FIGURE 23 : 调整 SCATTER FILE.....28

FIGURE 24 : 设置 HEAP 的起始地址和大小29

FIGURE 25 : 打开管理工程文件窗口30

FIGURE 26 : 添加或删除工程文件31

表目录

TABLE 1: OPL1000 SDK 软件第一级目录文件及其功能 3

TABLE 2: FW_BINARY 目录下文件列表..... 3

TABLE 3: FW_PACK 目录下文件列表 4

TABLE 4: SDK 目录下模块功能 4

TABLE 5: EXAMPLE 目录下示例工程实现的功能 5

TABLE 6: UART IO 关系..... 19

TABLE 7: 地址表..... 22

TABLE 8: TRACER_LOG_LEVEL_SET_EXT 函数入口参数定义 26

1. 介绍

1.1. 文档应用范围

本文档介绍如何在 OPL1000 内嵌 M3 MCU 基于 SDK 软件开发应用程序。内容包括：SDK 软件模块的功能介绍，基于 SDK 提供的例程移植并开发应用程序。

1.2. 缩略语

Abbr.	Explanation
ANCS	Apple Notification Center Service
APP	APPLication 应用程序
CMSIS	Cortex Microcontroller Interface Standard Cortex 微控制器接口规范
DEVKIT	DEvelopment Kit 开发板
EVB	Evaluation Board 评估板
FW	FirmWare 固件，处理器上运行的嵌入式软件
LDO	Low dropout regulator 低压差线性稳压器
OTA	Over the Air Technology 空中下载技术
TT	Transparent Transmission 透传

1.3. 参考文献

- [1] DEVKIT 快速使用指南 OPL1000-DEVKIT-getting-start-guide.pdf
- [2] CMSIS-RTOS 介绍 <http://www.keil.com/pack/doc/CMSIS/RTOS/html/index.html>
- [3] Download Tool 使用指南 OPL1000-patch-download-tool-user-guide.pdf
- [4] PinMux Tool 使用指南 OPL1000-pinmux-tool-user-guide.pdf
- [5] Multiple Device Download Tool 使用指南 MP_tool introduction_Chinese.pdf
- [6] SDK 快速使用指南 OPL1000-SDK-getting-start-guide.pdf
- [7] WIFI/BLE API 使用说明 OPL1000-WIFI-BLE-API-guide.pdf
- [8] AT 命令和例程说明 OPL1000-AT-instruction-set-and-examples.pdf
- [9] OPL1000 系统初始化介绍 OPL1000-system-initialization-brief-introduction.pdf

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



























OPL1000-SDK-开发者指南, UG1-02-2

2. OPL1000 SDK 软件包

2.1. 目录结构

OPL1000 SDK 软件包含若干层级目录，在本章节中仅介绍三个层级目录，Figure 1 显示了 SDK 发布包的三级目录层级结构。

Figure 1: OPL1000 SDK 发布包目录

- >  Demo
- >  Doc
 -  FW_Binary
 -  FW_Pack
- ▼  SDK
 - ▼  APS
 - >  apps
 - >  driver
 - >  FreeRtos
 - >  middleware
 - >  project
 - >  targets
 -  tools
 - ▼  APS_PATCH
 - >  apps
 - >  driver
 - >  examples
 - >  FreeRtos
 - >  middleware
 - >  project
 - ▼  Tool
 - >  CH340_Windows_Drivers
 - >  CP210x_Windows_Drivers
 -  Download
 - >  MP_Tool
 -  PinMux

第一层目录下包含的文件以及其功能如表 Table 1 所示。

Table 1: OPL1000 SDK 软件第一级目录文件及其功能

编号	目录名	说明
1	Demo	若干 OPL1000 的功能演示例程，例如使用 BLE 进行 WIFI 配网，TCP Client 连接和通信
2	FW_Binary	存放 pack 好的 opl1000.bin, at.bin 和 at_ext_ldo.bin。
3	FW_Pack	存放 M0，OTA loader 以及 m0_ldo.bin 以及固件合并脚本文件。
4	SDK	存放 SDK 的 include 文件，部分源码，Patch lib 文件和示例代码。
5	Tool	存放管脚复用(Pin-Mux) 设置工具和固件下载工具，多设备固件下载工具，串口驱动程序等。
6	Doc	存放 SDK API 使用说明文档，用户应用程序编译和开发指南文档等。
7	Release_Notes.md	SDK 软件发布说明文件，列出更新事项和发布包支持的功能、特性等。
8	README.md	简略介绍 SDK package 的内容

Fw_Binary 目录下包含 pack 成的 Opl1000_at.bin、Opl1000_at_ext_ldo.bin。

Table 2: Fw_Binary 目录下文件列表

编号	文件名	说明
1	Opl1000_at.bin	原厂提供的 opl1000 固件支持所有的 AT 命令。使用 blewifi 的 transparent target 编译出来的 m3.bin 和 m0.bin 合并成。
2	Opl1000_at_ext_ldo.bin	原厂提供的 opl1000 固件支持所有的 AT 命令。使用 blewifi 的 transparent target 编译出来的 m3.bin 和 m0_ldo.bin 合并成。

FW_Pack 目录下包含 M3，M0 固件补丁 Bin 文件和合并脚本文件。合并脚本文件 PatchData.txt 和发布的 Bin 文件是配合使用的，不同版本的 M3/M0 Bin 文件和 PatchData.txt 不能混用。用户在使用 download tool 下载固件时，一定要使用同一版本发布的合并脚本文件。

Table 3: FW_Pack 目录下文件列表

编号	文件名	说明
1	opl1000_m0.bin	原厂提供的 OPL1000 M0 固件补丁文件。M0 Patch Bin 文件需要和 opl1000_app_m3.bin 一起合并下载到外部 Flash 中，OPL1000 复位后从 Flash 中载入合并的 Bin 文件。使用芯片内部 PMU 供电。
2	Opl1000_ota_loader.bin	OPL1000 OTA loader。用于载入 OTA 固件的引导程序。
3	PatchData.txt	M3,M0 Bin 文件合并脚本文件。它用于定义如何将 M3、M0 的 Bin 文件合并到一个固件文件。该文件被 Download Tool 的 Pack 功能所调用。
4	Opl1000_m0_ldo.bin	支持外部 LDO 电源供应。硬件对应使用外部 LDO 供电。

SDK 目录下的两级子目录包含的功能模块如下表所述。

Table 4: SDK 目录下模块功能

编号	第一级目录名	第二级目录名	说明
1	APS	Apps	ROM code 中与 WIFI 和 LE 应用有关的函数头文件
2		driver	OPL1000 外设驱动
3		FreeRtos	FreeRTOS 移植源码
4		middleware	包括旺凌和第三方（如 Lwip, fatfs, tinycrypt 等）提供的中间件，例如 WIFI、BLE 的协议、AT 指令、控制层等。
5		project	包含用于产生 M3 固件 patch lib 和 SDK lib 的工程文件
6		targets	存放生成的 patch lib 和 SDK lib 的文件
7		tools	包含两个文件：（1）将 bin 文件转成 Hex 的 srec_cat.exe 工具（2）用于自动生成工程文件中头文件定义的 SVN 版本号 subwcrev.exe 工具。
1	APS_PATCH	Apps	存放 OTA loader (boot agent) 实现例程、LE 和 WIFI 应用补丁程序等。
2		driver	OPL1000 外设驱动的补丁程序，当 ROM_CODE 中的外设驱动不需要打补丁时，则其对应的目录是空的。
3		BA	用于产生 OTA loader 的工程文件，包括实现源码。

编号	第一级目录名	第二级目录名	说明
4		examples	包含各种示例代码，如 WIFI、BLE、外设、协议层、系统应用等例程。
5		FreeRtos	FreeRTOS 应用的补丁程序。
6		middleware	中间件的补丁程序。
7		project	与 APS 目录下的 Project 相对应，工程文件的补丁程序。

APS 和 APS_PATCH 存在同名的目录，需要说明的是，APS 目录下源码已经被编译和烧录到 OPL1000 的 ROM 中，APS 目录下的源码仅供用户参考，不可修改。APS_PATCH 目录下对应 APS 同名目录为 SDK 提供的 ROM 补丁代码。

SDK\APS_PATCH\examples 目录下提供若干示例文件，用户可以参考提供的示例工程根据需要移植、开发自己的应用程序。这些示例工程所实现的功能如表

Table 5 所述。

Table 5: Example 目录下示例工程实现的功能

编号	第一级目录	第二级目录	说明
1	bluetooth	ancs	Apple Notification Center Service 功能演示
		ble_adv	BLE Slave 设备发广播
2		gatt_client	GATT Client 功能演示
3		gatt_server	GATT Server 功能演示
4		gatt_server_service_table	GATT Server 服务表功能演示
5	get_started	blink	控制 LED 灯闪烁功能演示
6		hello_world	多任务间通信打印 Hello World 字符串输出
7		log	展示如何打开/关闭固件内部模块以及用户任务的 log 输出信息
8	peripheral	auxadc	辅助 ADC 使用例程
9		gpio	Gpio 设置演示
10		i2c	I2C master 功能演示

编号	第一级目录	第二级目录	说明
11		i2c_slave	I2C slave 功能演示，和外部 I2C Master 通信
12		pwm	PWM 端口设置展示
13		spi_master	SPI master 功能演示，访问 Flash 和外部 SPI slave 设备
14		timer_group	Timer 功能演示
15		uart	UART 配置和通信演示
16	protocols	http_request	http 访问 example.com 功能演示，基于 CMSIS RTOS
17		http_request_freertos	http 访问 example.com 功能演示，基于 Free RTOS
18		tcp_client	TCP client 功能演示
19		sntp_get_calender	SNTP 获取日期的演示用例
20		mdns	mDNS 业务广播演示用例
21		mqtt	Mqtt 发布和订阅消息
22		https_request	https 访问 www.howssmyssl.com 功能演示，基于 CMSIS RTOS
23	wifi	iperf	网络性能测试工具 iperf 的演示用例
24		wpa2_station	AP 为 WPA2 加密模式，OPL1000 作为 WIFI Station 连接 AP 功能演示
25		wpa2_station_gpio	在 wpa2_station 示例基础上添加 GPIO 外设的使用
26	system	ota_wifi	基于 WIFI 的空中下载演示例程
27		ali_blewifi	基于 BLEWIFI 的,OPL1000 作为透传模块和阿里云之间发布/接收 MQTT 消息的演示例程
28		aws_IOT_embeeded_C_blewifi	基于 BLEWIFI 的,OPL1000 作为透传模块和 AWS 云之间发布/接收 MQTT 消息的演示例程
29		tcp_client_blewifi	基于 BLEWIFI 的,OPL1000 作为透传模块和 TCP server 之间收发数据的演示例程

2.2. SDK 发布包使用

用户拿到 SDK 发布包后，建议通过如下步骤了解、掌握 OPL1000 功能。在了解 SDK 提供的示例工程基础上，根据需要选择合适的例程移植、开发自己的应用程序。

Step 1: 了解 DevKit 使用和用户 APP 开发过程

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

OPL1000-SDK-开发者指南, UG1-02-2

首先阅读参考文献[1] “DEVKIT 快速使用指南” 和[5] “SDK 快速使用指南” 两篇文档，了解以下知识：

(1) 在 DEVKIT 板上开发和编译应用程序的流程，知道如何将最新发布的固件下载更新到 OPL1000 DEVKIT 板上。

(2) 用户 APP 工程文件设置方法，以及如何基于 OPL1000 SDK 示例工程开发自己的应用。

Step 2: 使用 Pin-Mux、Download 工具软件 和 mp 下载工具

一般地 IOT 应用都会使用到 OPL1000 的外设资源。OPL1000 提供 SPI、I2C、UART、PWM、GPIO 和 AUX ADC 等外设。这些外设可以在 OPL1000 开放的 18 根 GPIO 管脚上进行分配和定义。用户可以根据自己需要使用 Pin-Mux 工具分配管脚功能并定义外设工作模式。

用户开发的应用程序最终编译为 M3 上执行的固件程序，它需要和厂商提供的 M0 Bin 文件合并为一个固件补丁程序，然后下载到 Flash 中，上电后载入到 RAM 中执行。如果用户开发的应用程序需要支持 OTA 空中下载，会使用到下载工具提供的 OTA Pack 功能，因此用户需要掌握 Download tool 使用。这两个工具的使用手册可以通过点击软件界面提供的 “use manual” 按钮打开。同时在 Doc 目录下也提供 PDF 版本的帮助文件（参考文献[3]和[4]）。

厂商在多设备下载时使用 mp 下载工具，硬件连接后使用多设备下载工具，即可进行多设备同时下载。Tool 目录下包含：校准工具、下载工具、RF 测试工具。此工具的使用在 tool 目录下有 pdf 的使用文档，（参考文献[5]）。

Step 3: 运行 AT 命令或者编译执行示例工程评估使用 OPL1000

通过 DEVLKIT 的 AT 串口可以发出 AT 命令，控制 OPL1000 完成特定的 WIFI、BLE 功能。具体可以参考文献 [7] “AT 命令和例程说明” 文档。用户也可以直接编译 SDK 提供的示例代码，下载到 OPL1000 中执行。第二种评估方法便于用户了解底层提供的 API 功能，如何调用 API 实现特定功能。API 函数的介绍可参考文献 [6] “WIFI/BLE API 使用说明”。

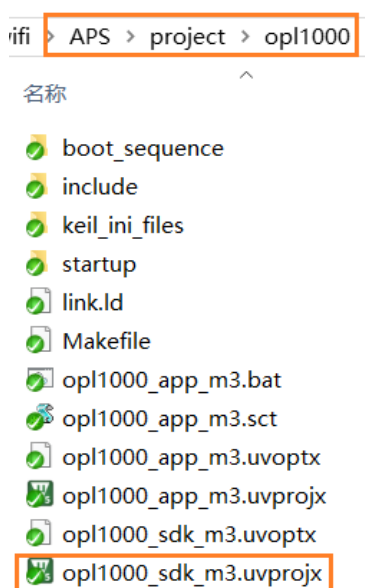
Step 4：基于示例工程移植并开发应用程序

如 2.1 章节所述 SDK 提供了若干示例工程，用户可以根据自己的需要选择其一或者若干，进行裁剪、合并，移植开发自己的应用程序。具体流程可参考本文档第 4 章、第 5 章介绍。

2.3. SDK 库的更新和编译

用户在开发自己的应用程序的过程中，可能会需要对 SDK 库做更新或裁剪。在这种情况下，用户可使用 SDK 库对应的工程文件 (opl1000_sdk_m3.uvprojx) 对其进行更新并重新编译。该文件在 APS/project/opl1000 目录下

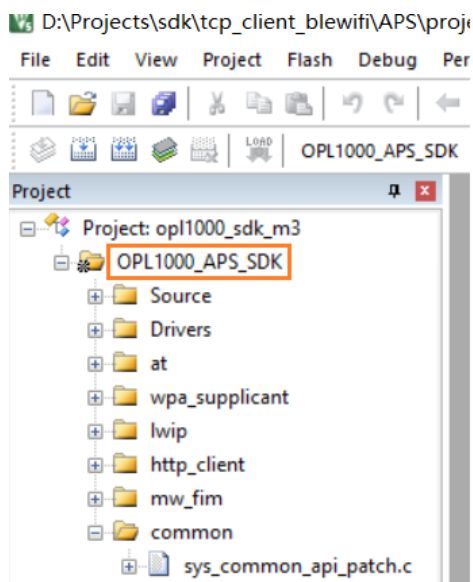
Figure 2: SDK 库工程目录结构



更新并重新编译 SDK 库的通常做法如下：

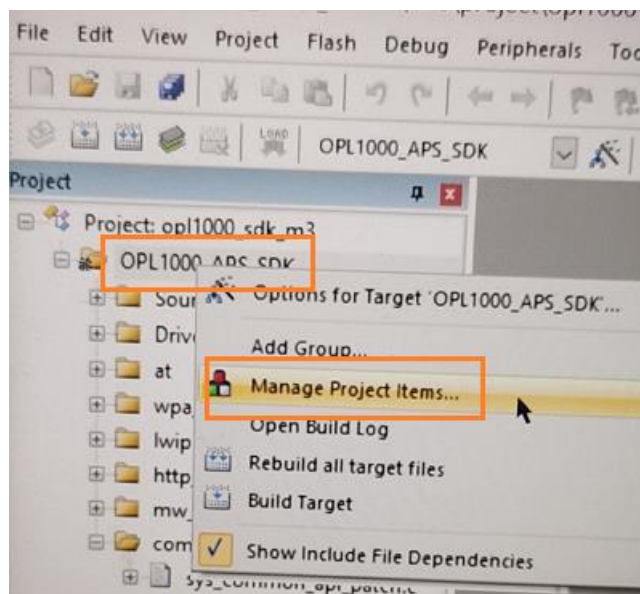
1. 双击 “opl1000_sdk_m3.uvprojx” 文件，打开 SDK 库的工程如下；

Figure 3: SDK 库工程文件



2. 右键点击工程文件名称 “OPL1000_APS_SDK” ,选择 “Manage Project Items...” 选项，打开如下界面

Figure 4: 打开管理工程文件窗口

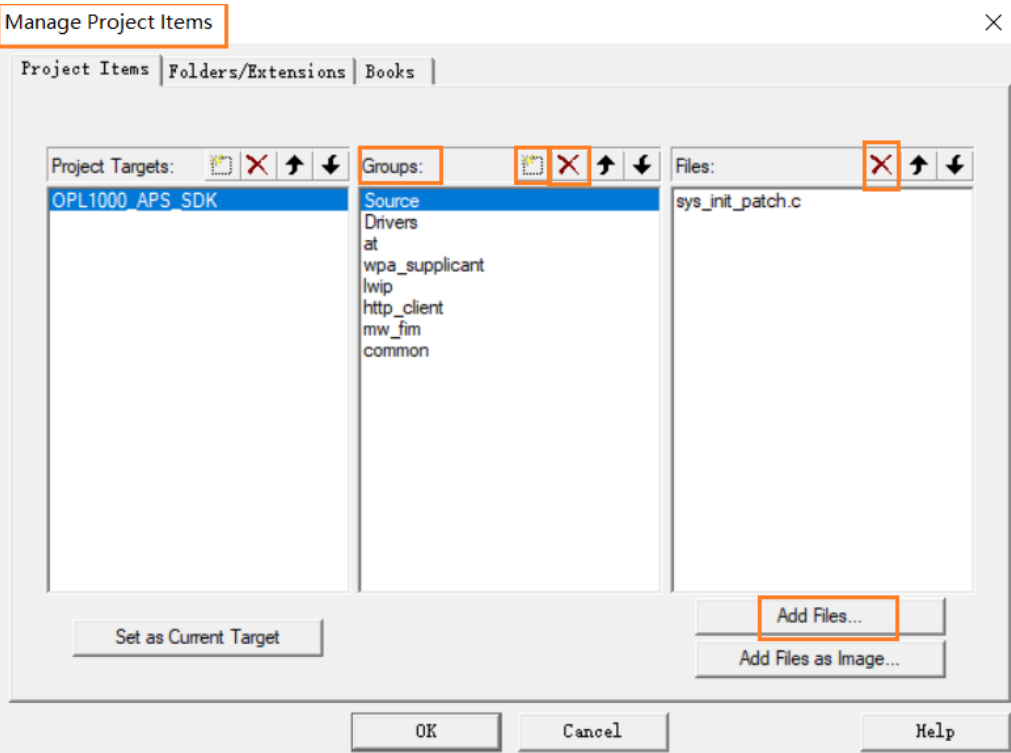


3. 在 Manage Project Items 界面，用户根据需要:

- 添加/删除 Group;
- 在某个 Group 中添加/删除文件。

更新完成后点击“ok”按钮。

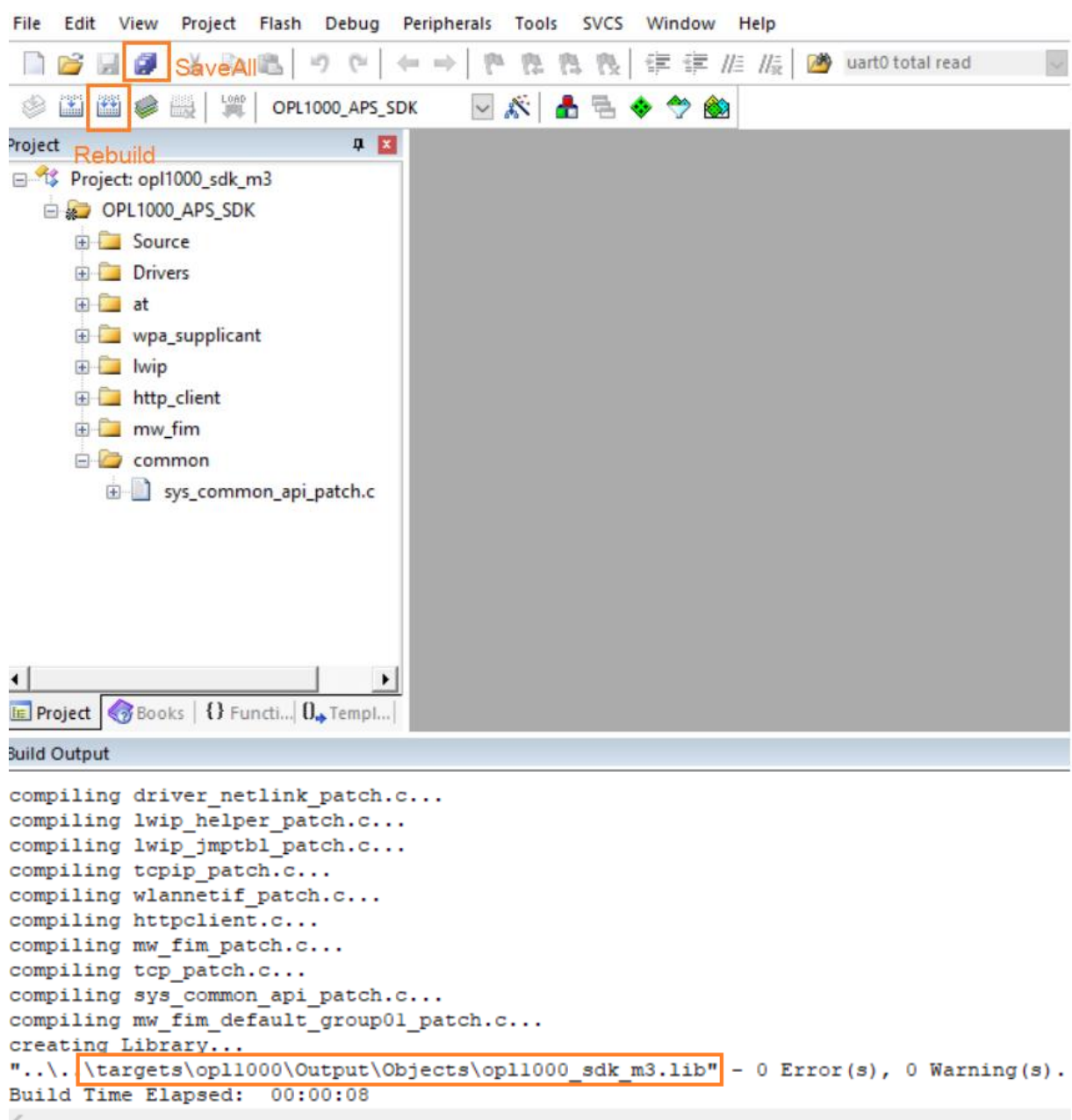
Figure 5：添加或删除工程文件



4. 更新好工程文件的内容后，用户可能还会需要更新某些文件的内容（这点不在此介绍）。在所有的更新完成后，用户即可开始点击“Rebuild”按钮编译 SDK 库，直至编译成功。【注意：在编译完成，准备退出前，确保点击一下“SaveAll”按钮，以保存前面对工程文件所作的更新。】

最终生成的库文件的位置是 `APS\targets\opl1000\Output\Objects\opl1000_sdk_m3.lib`

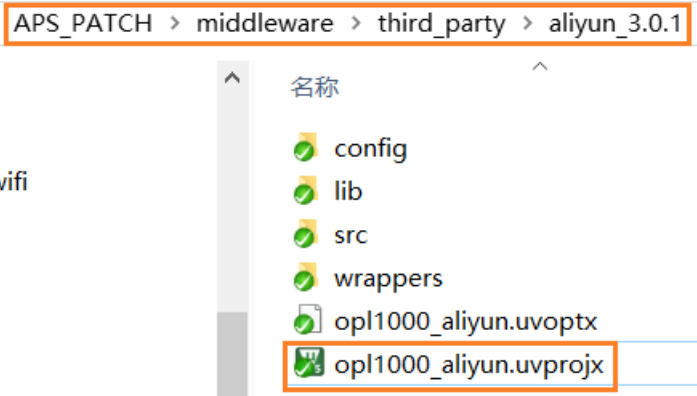
Figure 6: 构建 SDK 工程文件



2.4. Ali SDK 库的编译

用户在开发跟阿里云相关的应用程序时，可能会需要重新编译 Ali SDK 库。在这种情况下，用户可使用它对应的工程文件 (opl1000_aliyun.uvprojx) 进行重新编译。该文件在 *APS_PATCH\middleware\third_party\aliyun_3.0.1* 目录下

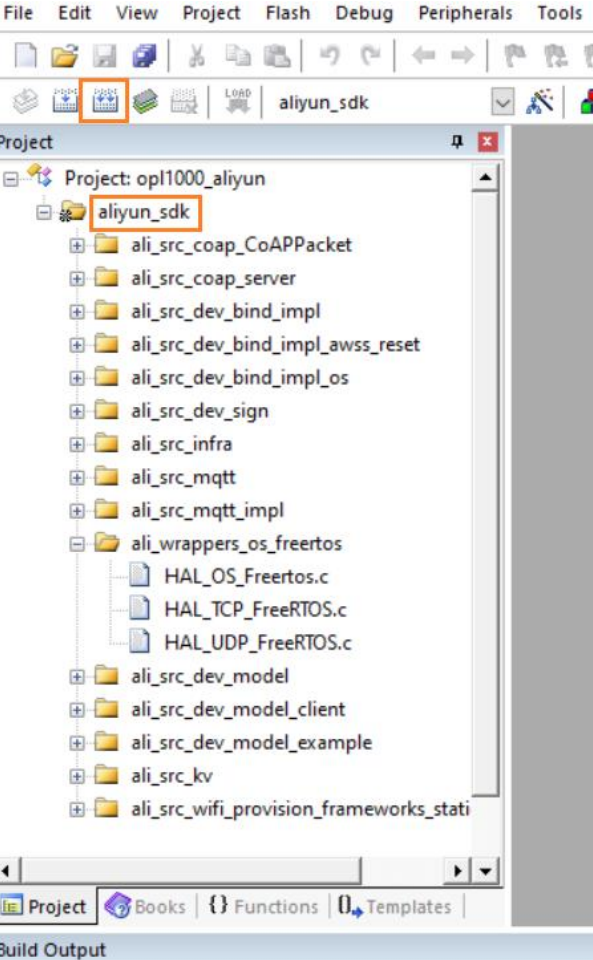
Figure 7: 阿里 SDK 库的目录结构



重新编译 Ali SDK 库的通常做法如下：

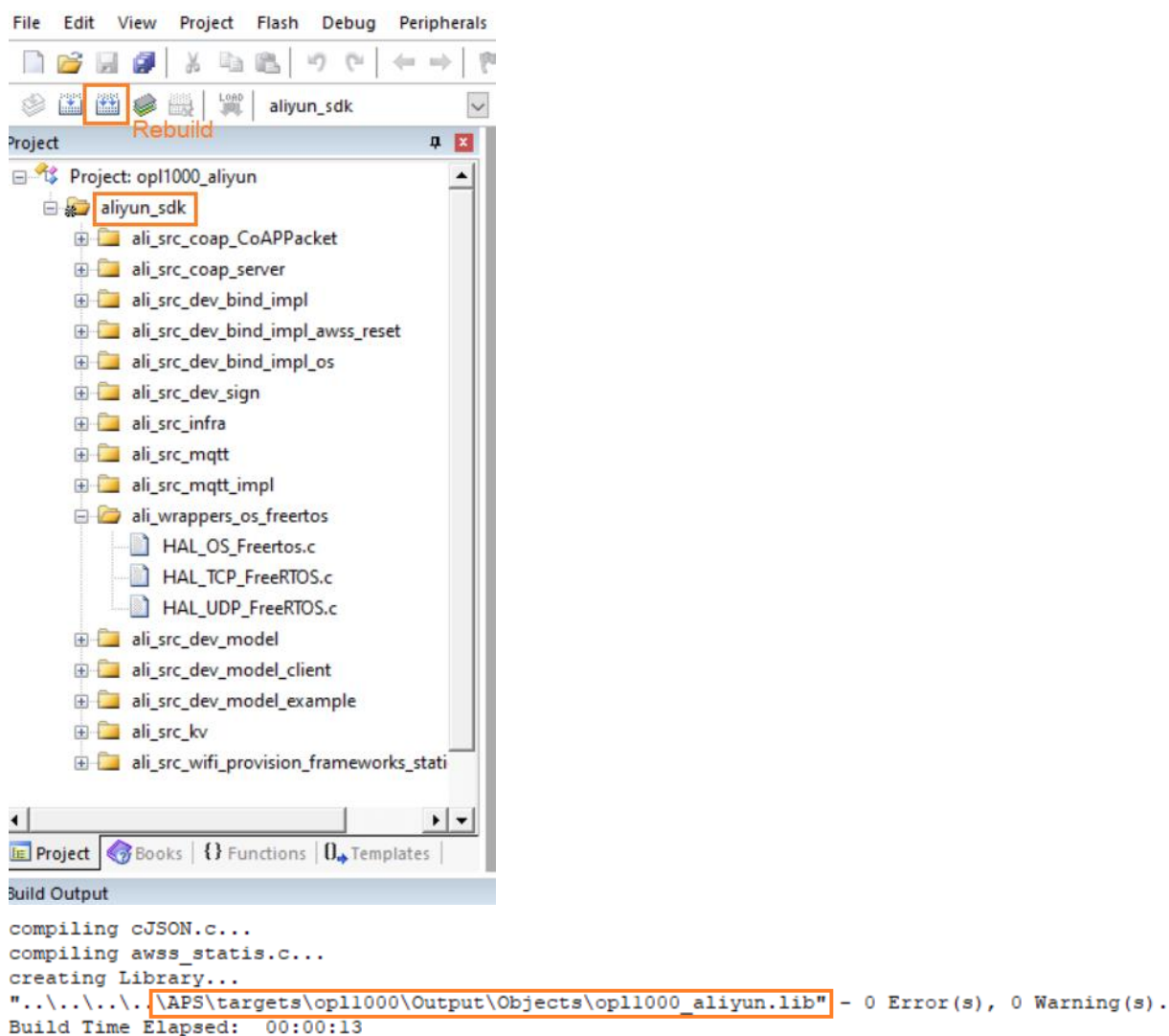
1. 双击 “opl1000_aliyun.uvprojx” 文件，打开该工程如下；

Figure 8: 阿里 SDK 库工程文件



2. 和 SDK 库的工程类似，用户也可根据需要更新 Ali SDK 库包含的文件和文件的内容。在所有的更新完成后，用户即可开始点击 “Rebuild” 按钮编译 Ali SDK 库，直至编译成功。最终生成的库文件的位置是 `APS\targets\opl1000\Output\Objects\opl1000_aliyun.lib`

Figure 9: 构建 SDK 工程



3. 应用程序开发流程

OPL1000 应用程序有两种开发模式，一个是使用 SWD 调试口的 IDE 在线调试模式，另一种是离线的通过串口打印调试模式。前者适用于开发初期应用程序尚不成熟阶段，后者适用于应用程序相对成熟，执行流程相对复杂的情况。

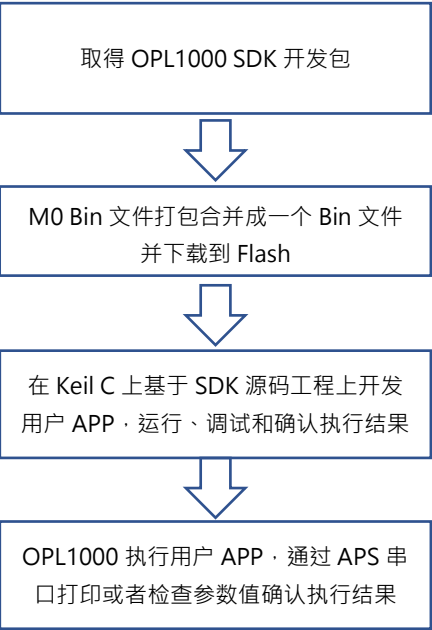
3.1. IDE 在线调试开发模式

IDE 在线调试开发模式包括以下 4 个步骤：

- 1. 从原厂拿到 OPL1000 SDK。
- 2. 将 SDK FWBinary 目录下 M0 Bin 文件按照[文献 \[3\] “Download Tool 使用指南”](#)说明打包成一个 Bin 文件并下载到 Flash 中。
- 3. 在 Keil C 环境基于 SDK 开发用户 APP。
- 4. 在 Keil C 环境下，在线仿真运行、调试，通过 IDE 检查变量值或者 APS 串口打印确认执行结果。注意此时 M3 固件并没有下载到 flash 中，代码仅仅在 RAM 中执行，掉电后会丢失。

这四个步骤可以用下图表示。

Figure 10: IDE 在线开发模式



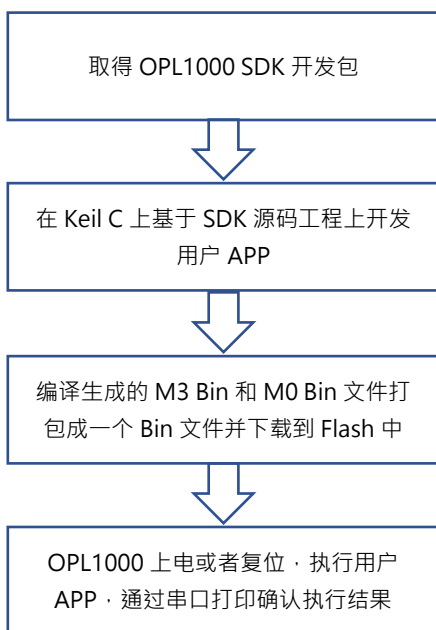
3.2. 串口调试开发模式

串口调试开发模式包含 5 个步骤。

1. 从原厂拿到 OPL1000 SDK。
2. 在 Keil uVision 上开发用户 APP，开发过程可以完全独立于 DEVKIT 板。
3. 源码开发完成，编译获得 M3 Bin 文件。
4. 将 SDK 软件包的 FWBinary 目录下 M0 Bin 文件和用户开发的 M3 bin 文件按照 [文献 \[3\] “Download Tool 使用指南”](#) 说明打包成一个 Bin 文件并下载到 Flash 中。
5. OPL1000 上电或者复位，执行用户 APP，通过串口 log 信息确认执行结果。

整个过程如下图所示。

Figure 11: 串口调试开发模式



4. GNU ARM GCC 验证环境的搭建

Opulinks 为所有的工程引入了 Makefile 文件，以方便用户在 Linux 或类 Linux 环境下验证或二次开发相应的例程。本章主要介绍了在 Windows10 下搭建 GNU ARM GCC 验证环境的过程。

4.1. 工具链和构建工具的下载和安装

步骤 1：工具的下载

工具链（toolchain）和构建工具（buildtool）需要从第三方官网下载。目前最新的版本可通过下面的链接下载：

Toolchain：<https://developer.arm.com/tools-and-software/open-source-software/developer-tools/gnu-toolchain/gnu-rm/downloads/8-2018-q4-major>

Buildtool：<https://gnu-mcu-eclipse.github.io/windows-build-tools/releases/>

步骤 2：工具的安装

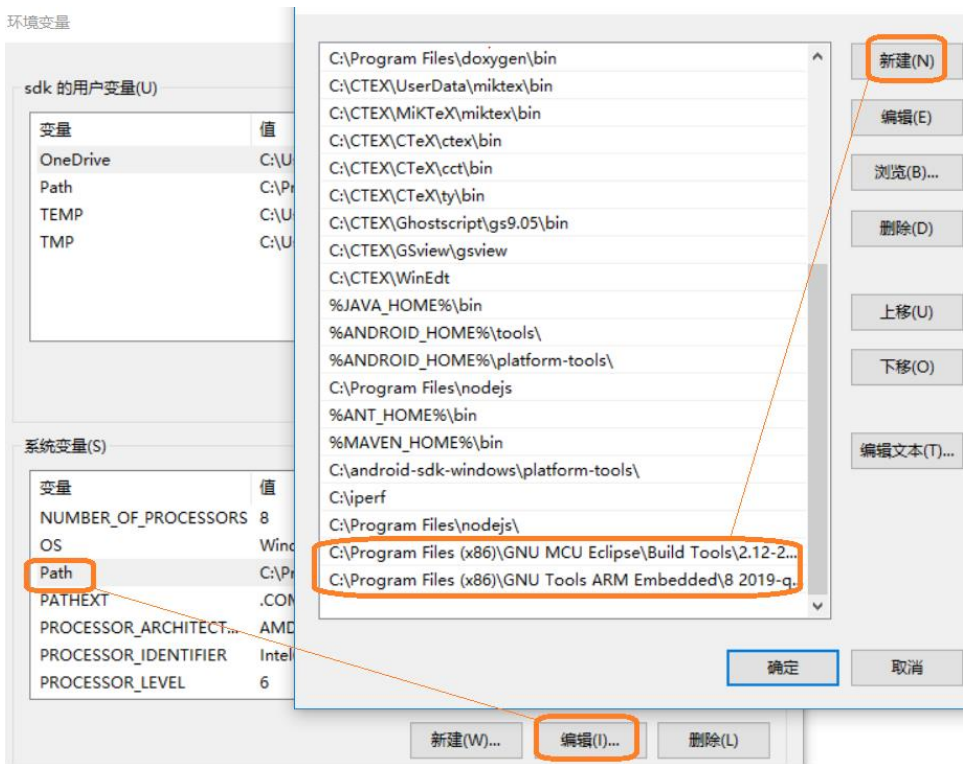
工具链的安装程序是个以 exe 结尾的可执行文件(如，gcc-arm-none-eabi-8-2018-q4-major-win32-sha2.exe)，直接打开它即可安装，安装完毕后记录下它的安装路径，如，C:\Program Files (x86)\GNU Tools ARM Embedded\8 2018-q4-major\bin；

下载的构建工具是个压缩包，将其解压缩后即可使用，如，C:\Program Files (x86)\GNU MCU Eclipse\Build Tools\2.12-20190422-1053\bin。

步骤 3：设置 Path 环境变量

在安装完成后，需要把它们 bin 目录加入到 Path 环境变量如下：

Figure 12: 设置环境变量



4.2. 使用 GNU ARM GCC 构建工程

安装好 GNU ARM GCC 后，用户可以在每个工程的目录下直接执行 make 命令生成该工程的 m3.bin 文件如下：

Figure 13: 使用 GNU ARM GCC 构建 helloworld 工程

```
D:\Projects\A2\release\v007\SDK\APS_PATCH\examples\get_started\hello_world>make
arm-none-eabi-objcopy -O binary ./Output-GCC/opl1000_app_m3.elf ./Output-GCC/opl1000_app_m3.bin

D:\Projects\A2\release\v007\SDK\APS_PATCH\examples\get_started\hello_world>dir Output-GCC
驱动器 D 中的卷是 D
卷的序列号是 525F-DEC4

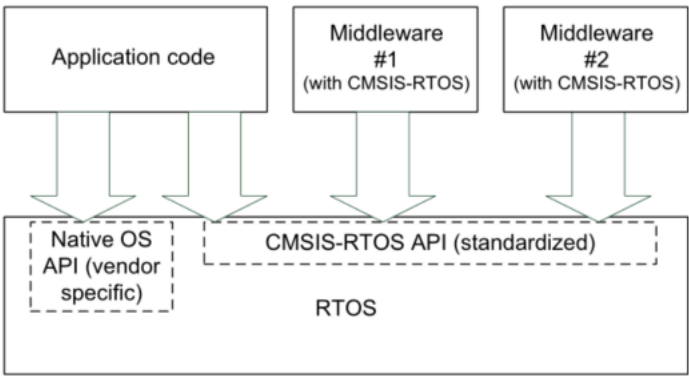
D:\Projects\A2\release\v007\SDK\APS_PATCH\examples\get_started\hello_world\Output-GCC 的目录
2019/08/07 16:03 <DIR>      .
2019/08/07 16:03 <DIR>      ..
2019/08/07 16:03             17,736 opl1000_app_m3.bin
2019/08/07 16:03             329,688 opl1000_app_m3.elf
                2 个文件      347,424 字节
                2 个目录 459,041,296,384 可用字节

D:\Projects\A2\release\v007\SDK\APS_PATCH\examples\get_started\hello_world>
```

5. 嵌入式操作系统

OPL1000 软件的多任务操作系统底层采用 FreeRTOS 嵌入式操作系统，上层使用 CMSIS-RTOS API 对 FreeRTOS 封装，CMSIS 定义了基于 Cortex-M 构架的微控制器标准的 RTOS API。CMSIS RTOS API 不依赖于硬件层接口，支持代码重复使用，降低开发者掌握不同实时嵌入式操作系统学习成本。CMSIS RTOS 和第三方 RTOS 的层次关系如图下图所示：

Figure 14: CMSIS RTOS 架构



SDK 提供的 CMSIS RTOS 接口包括：

- Thread
- Timer
- Mutex
- Semaphore
- Memory Pool
- Message Queue

如果需要了解更多 CMSIS-RTOS 信息，参考文献[2] 提供了更多 CMSIS-RTOS API Version 1 相关内容。需要注意的是，由于 OPL1000 SDK 已经包含了 CMSIS RTOS 源码，因此我们不需要从 Keil PACK 里面导入 CMSIS RTOS，否则可能会因为版本差异引起冲突。

6. 外设参数配置

OPL1000 的 GPIO 可以根据用户的使用场景灵活复用，在使用 GPIO 时，首先需要使用 PinMux 工具配置 OPL1000 外设的对应的 GPIO。以 UART 为例，首先使用 SDK 提供的 PinMux 工具在 UART 选项里使能 UART0 和 UART1，配置波特率、停止位等参数。

Figure 15: 外设配置

IO

UART

SPI

I2C

PWM

AUX/ADC

GPIO

☒ Enable UART0

☒ Normal

☐ Flow Control

Baud Rate

115200

Data Bits

DATA_BIT_8

Stop Bit

STOP_BIT_1

Parity

PARITY_NONE

☒ Enable UART1

☒ Normal

☐ Flow Control

Baud Rate

115200

Data Bits

DATA_BIT_8

Stop Bit

STOP_BIT_1

Parity

PARITY_NONE

设置完参数以后，切换到 IO 选项，配置 IO 为下列对应关系

Table 6: UART IO 关系

外设	IO
UART0_TX	IO2
UART0_RX	IO3
UART1_TX	IO4
UART1_RX	IO5


配置完成以后状态如下图：

Figure 16: IO 配置

Use Manual

SW version: 0.0

IO	UART	SPI	I2C	PWM	AUX/ADC	GPIO														
	pin	IO2	IO3	IO4	IO5	IO6	IO7	IO8	IO9	IO10	IO11	IO16	IO17	IO18	IO19	IO20	IO21	IO22	IO23	
1	UART0_TX	<input checked="" type="checkbox"/>				<input type="checkbox"/>				<input type="checkbox"/>		<input type="checkbox"/>				<input type="checkbox"/>				
2	UART0_RX		<input checked="" type="checkbox"/>				<input type="checkbox"/>				<input type="checkbox"/>		<input type="checkbox"/>				<input type="checkbox"/>			
3	UART1_TX			<input checked="" type="checkbox"/>				<input type="checkbox"/>			<input type="checkbox"/>		<input type="checkbox"/>	<input type="checkbox"/>			<input type="checkbox"/>			
4	UART1_RX				<input checked="" type="checkbox"/>				<input type="checkbox"/>	<input type="checkbox"/>					<input type="checkbox"/>	<input type="checkbox"/>			<input type="checkbox"/>	
5																				
6																				

点击  在 PinMux GUI 当前目录下生成 OPL1000_pin_mux_define.c。此文件包含的结构体 OPL1000_periph 定义了 OPL1000 外设管脚配置和参数设置，其中对应 UART 部分已经填充了刚才设置的 UART0 和 UART1 的管脚参数和属性参数。

```
T_OPL1000_Periph OPL1000_periph = {
2,{
    {UART_IDX_0,
    OPL1000_IO2_PIN,
    OPL1000_IO3_PIN,
    BLANK_PIN,
    BLANK_PIN,
    115200,
    DATA_BIT_8,
    PARITY_NONE,
    STOP_BIT_1,
    UART_SIMPLE},

    {UART_IDX_1,
    OPL1000_IO4_PIN,
    OPL1000_IO5_PIN,
    BLANK_PIN,
    BLANK_PIN,
    115200,
    DATA_BIT_8,
    PARITY_NONE,
    STOP_BIT_1,
    UART_SIMPLE}

},
```

将文件 OPL1000_pin_mux_define.c 添加进用户的 Keil 工程，在外设初始化代码中调用 Hal_PinMux_Uart_Init 函数，传入 UART 结构体参数。

```
void App_Pin_InitConfig(void)
{
    /*UART0 Init*/
    Hal_PinMux_Uart_Init(&OPL1000_periph.uart[0]);
    /*UART1 Init*/
    Hal_PinMux_Uart_Init(&OPL1000_periph.uart[1]);
}
```


该函数从结构体 OPL1000_periph 获取配置参数后，完成 GPIO 的 PinMux 设置和属性参数设置操作。
用户需要配置其它外设操作步骤与前面叙述的设置 UART 相同。

7. 应用开发基本配置

用户开发 APP 时建议使用 Keil uVision 5 IDE 工具开发。本章节介绍基于 Keil uVision 的用户 APP 工程配置方法，并结合 hello_world 示例工程简要说明工程配置，文件结构，RTOS 使用等事项。

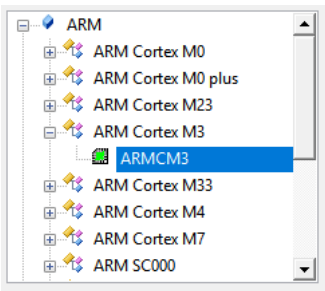
7.1. Keil uVision 工程配置

打开 **hello_world** 示例工程。路径：SDK\APS_PATCH\examples\get_started\hello_world

点击  按钮，配置如下几项。

1. Device 默认选择选择 ARMCM3，用户如果新建工程需要注意此处的选择。

Figure 17: Device 选择



2. Target 标签设置 OPL1000 的 ROM 地址及大小和 RAM 地址及大小。

Table 7: 地址表

类型	起始地址	SIZE
IROM1	0x0	0xC0000
IRAM1	0x400000	0x50000

3. Linker 标签内不需要选择 **use Memory layout From Target Dialog** 选项。

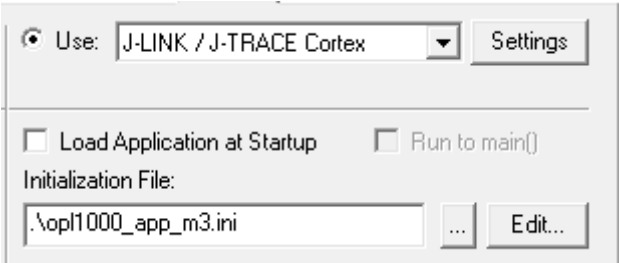
需要设置 Scatter File 文件，如下图。点击  按钮可打开该 Scatter File。

Figure 18: Scatter File



4. Debug 标签不选择 **Load Application at Startup** 选项，但是需要设置 **initialization File** 的内容，如下图所示。如果需要了解 ini 内容，点击 **Edit...** 打开 ini 文件。

Figure 19: debug 设置

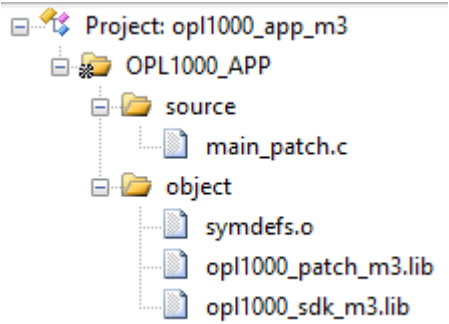


注意：鉴于该 ini 在 patch 代码 和 ROM 代码之间的重要性，不建议修改该文档内容，否则会引起代码无法正常运行的情况。

7.2. 工程预览

6.1 节以 **hello world** 为例介绍了 Keil 工程设置方法，本节大致介绍 **hello_world** 工程代码结构。一个基本工程至少包含这些文件：**main_patch.c**，**sysdefs.o**，**opl1000_patch_m3.lib**、**opl1000_sdk_m3.lib**。

Figure 20: hello world 工程源码结构



其中 `main_patch.c` 为工程示例代码，`sysdefs.o` 和 `opl1000_patch_m3.lib`、`opl1000_sdk_m3.lib` 为库文件。

`sysdefs.o` 的目录地址：SDK\APS\targets\opl1000\object\

`opl1000_patch_m3.lib`、`opl1000_sdk_m3.lib` 所在目录地址：

SDK\APS\targets\opl1000\Output\Objects\

如果用户需要新建自己的工程，请从上述地址添加文件。

7.3. Main 函数入口

打开 `main_patch.c` 文件，定位到 `__Patch_EntryPoint(void)` 函数

```
static void __Patch_EntryPoint(void)
{
    // don't remove this code
    SysInit_EntryPoint();
    // update the flash layout
    Mwfim_FlashLayoutUpdate = Main_FlashLayoutUpdate;
    // application init
    Sys_AppInit = Main_AppInit_patch;
}
```

`SysInit_EntryPoint()` 函数已在 ROM 中实现，此处仅调用该函数进行初始化操作，所以禁止修改或移动此函数。接着代码重定义 `Sys_AppInit` 入口到 `Main_AppInit_patch` 函数，`Sys_AppInit` 是 SDK 留给用户端开发软件的 `main` 函数入口，通过映射 `Sys_AppInit` 的入口函数以后，所有的 APP 初始化如外设初始化、创建多任务等都在映射函数 `Main_AppInit_patch` 内创建。

7.4. Log 输出设置

用户在开发 APP 时，存在两种调试输出信息

- (1) 用户 APP 内部的 log
- (2) OPL1000 SDK firmware 的 log

Log 调试打印信息有助于用户检查应用程序工作执行流程和结果是否正常。firmware log 可以帮助用户快速定位和分析固件模块执行情况，例如 ble 和 wifi 协议栈的运行情况。

为避免 log 输出信息太多，不利于用户 APP 调试，SDK 只保留用户 log 信息输出，关闭固件内部 log 信息输出。如用户需要管理 log 输出机制，有三种方法：

(1) APS 串口 tracer 命令

输入命令以后，APS 串口列出目前所有正在运行的任务。

Figure 21:tracer 命令

```
Tracer Mode      [1]      0:disable/1:normal/2:print directly
Display Task Name [0]      0:disable/1:enable
Priority         [-2]     osPriorityIdle(-3) ~ osPriorityRealtime(3)
StackSize       [128]    number of uint_32
Queue Number     [128]    max number of log
Queue Size      [80]     max length of log
Log Level       [0x00:None/0x01:Low/0x02:Med/0x04:High/0x07:All]

Default Level for App Tasks      [0x07]

Index           Name: Level
-----
----- Internal Tasks (Start from Index 0)
[ 0]            opl_isr_: 0x00
[ 1]            opl_diag: 0x00
[ 2]            opl_wifi_mac: 0x07
[ 3]            opl_suppllicant: 0x00
[ 4]            opl_controller: 0x07
[ 5]            opl_le: 0x00
[ 6]            opl_event_loop: 0x07
[ 7]            opl_tcpip: 0x00
[ 8]            opl_ping: 0x00
[ 9]            opl_iperf: 0x00
[10]            opl_agent: 0x00
[11]            opl_at_wifi_app: 0x00
[12]            opl_at: 0x00
[13]            opl_at_tx_data: 0x00
[14]            opl_at_sock_: 0x00
[15]            opl_at_sockserv: 0x00
----- App Tasks (Start from Index 32)
```

用户创建的任务罗列在 **Start of App Tasks** 项目中，以 **user_app_demo** 任务为例，该任务被用户在主程序中创建，index 为 32，0x07 表示打印该任务所有 log，如用户需要关闭该任务的 log 输出，则在 APS 串口输入命令：**Tracer level 32 0x00**，这种做法也适用其他任务的 log 管理。

注意：这种做法前提是需要用户在主程序初始化时开启 APS 串口的输入功能，否则输入无效。

同时该配置掉电丢失，需要在下一次上电重新配置。

```
#include "hal_dbg_uart_patch.h"
static void Main_AppInit_patch(void)
{
    Hal_DbgUart_RxIntEn(1); // APS uart rx enable
}
```

(2) 使用 SDK 提供的 log 输出设置 API 管理任务 log

log 输出设置 API 相关内容如下：

```
extern T_TracerLogLevelSetFp tracer_log_level_set_ext;
int tracer_log_level_set_ext(uint8_t bIdx, uint8_t bLevel);
```

tracer_log_level_set_ext 函数入口参数定义如表 Table 8 定义

Table 8: tracer_log_level_set_ext 函数入口参数定义

参数	取值	说明
bIdx	0 ~ TRACER_TASK_IDX_MAX	内部模块(task) 索引号
bLevel	LOG_ALL_LEVEL	打印所有 log
	LOG_NONE_LEVEL	关闭 log

用户通过 API 开启 firmware 内部 log 调用方式为：

```
tracer_log_level_set_ext (2, LOG_ALL_LEVEL); // here 2->opl_wifi_mac
```

(3) 多任务 log 管理

如用需要对多个任务 log 输出进行差异化管理，需要编辑 msg_patch.c 的结构体 g_taTracerExtTaskInfoBodyExt。增添用户自定义任务

```
T_TracerTaskInfo g_taTracerExtTaskInfoBody[TRACER_EXT_TASK_NUM_MAX] =
{
    {"demo_app1", LOG_ALL_LEVEL, 0, 0 },
    {"demo_app2", LOG_ALL_LEVEL, 0, 0},
    {"", LOG_NONE_LEVEL, 0, 0},
};
```

其中的 demo_app1、demo_app2 为用户创建任务的名称，可配置定义为：

```
#define LOG_HIGH_LEVEL    0x04
#define LOG_MED_LEVEL    0x02
#define LOG_LOW_LEVEL    0x01
#define LOG_NONE_LEVEL    0x00
#define LOG_ALL_LEVEL    (LOG_HIGH_LEVEL | LOG_MED_LEVEL | LOG_LOW_LEVEL)
```


用户其他任务的 log 管理，仿照上述做法依次在该结构体中添加。

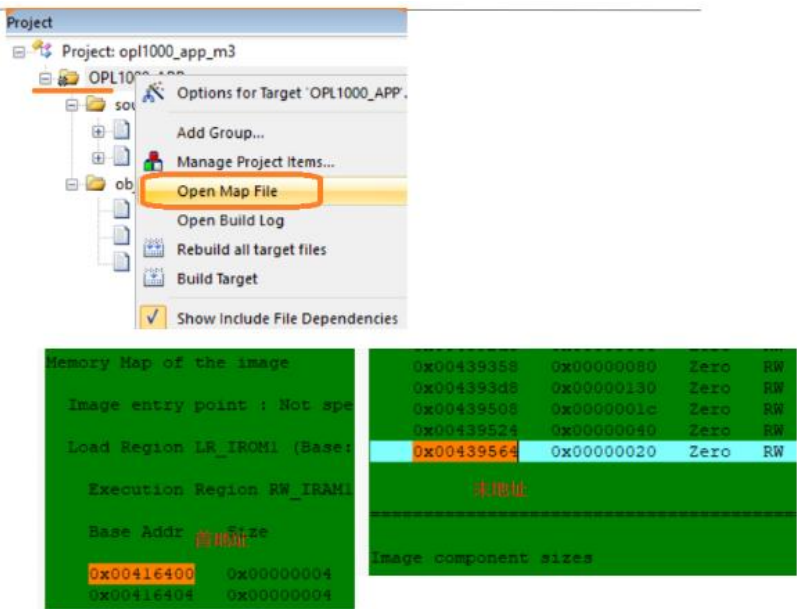
需要了解更多 Log 配置可以参考 log 配置例程，目录为 SDK\APS_PATCH\examples\get_started\log

7.5. 调整 heap size

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 。如右图所示。对末地址以 4k 为单位向上取整，为 0x0043a000.

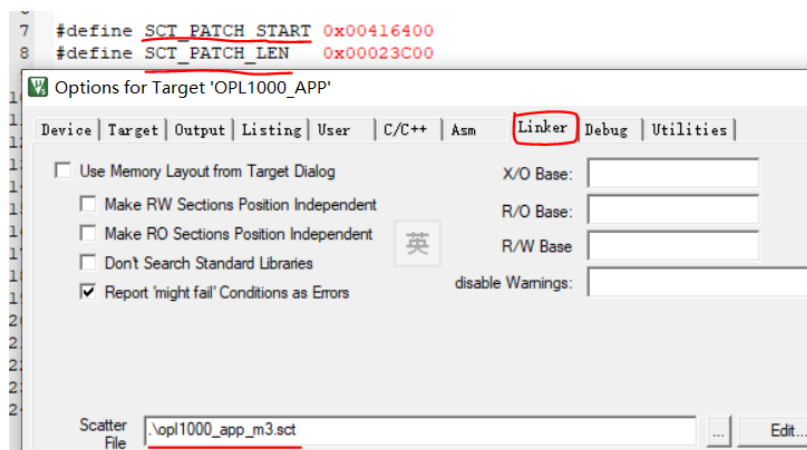
Figure 22: 在 map 文件获取首末地址



- (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,如右图所示

- 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 = $0x0043a000 - 0x00416400 = 0x00023C00$ 。

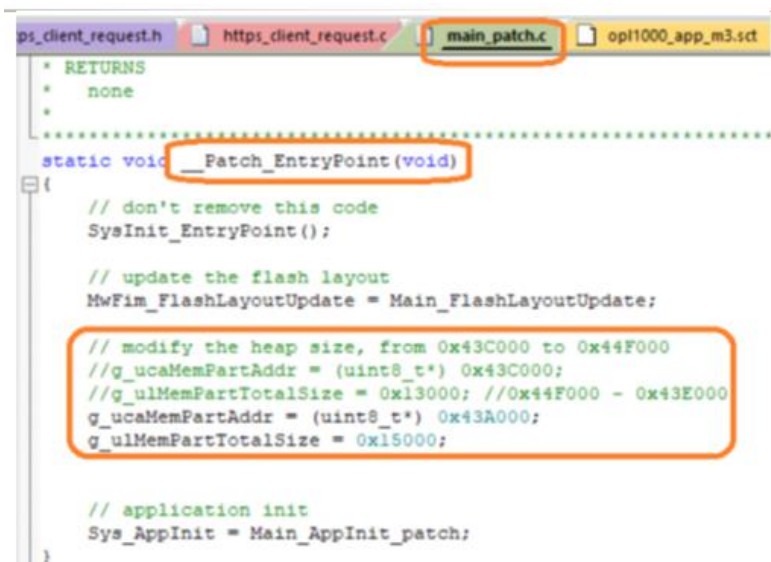
Figure 23: 调整 scatter file



(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，如右图所示。
- 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 = $0x44F000 - \text{末地址}$ ，对于有图中的例子，就是，大小 = $0x44F000 - 0x0043a000 = 0x15000$

Figure 24: 设置 Heap 的起始地址和大小



```
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;
}
```

8. 应用举例 – 基于 Ali 云透传应用

本章节以实现基于 Ali 云加透传为例，说明用户如何基于 OPL1000 SDK 开发一个自己的应用程序。

8.1. Ali 云透传工程介绍

SDK 已经提供了一个 ali_blewifi 示例工程，工程位于：SDK\APS_PATCH\examples\system\ali_blewifi 目录下。关于如何编译，下载和验证该示例工程，请参考 OPL1000-Ali-transparent-transmission-application-guide.pdf

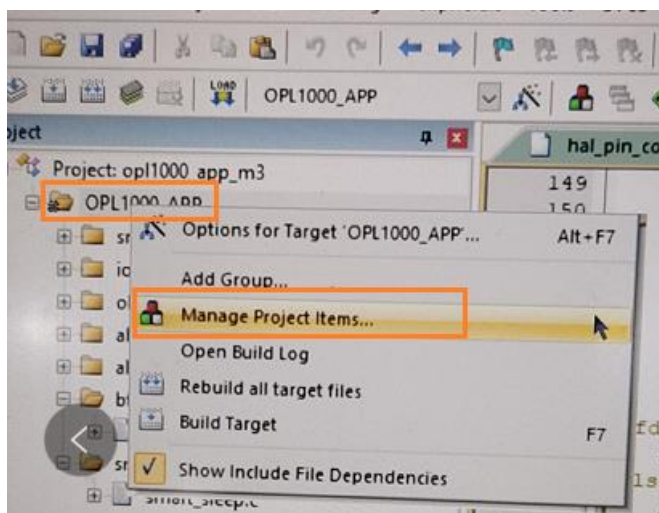
8.2. 更新工程配置或文件

在应用程序开发过程中，用户可能会根据需要添加或删除一些文件，方法如下：

打开 ali_blewifi 示例工程的工程文件：APS_PATCH\examples\system\ali_blewifi\

opl1000_app_m3.uvprojx。右键点击“OPL1000_APP”，选择“Manage Project Items...”选项。

Figure 25: 打开管理工程文件窗口

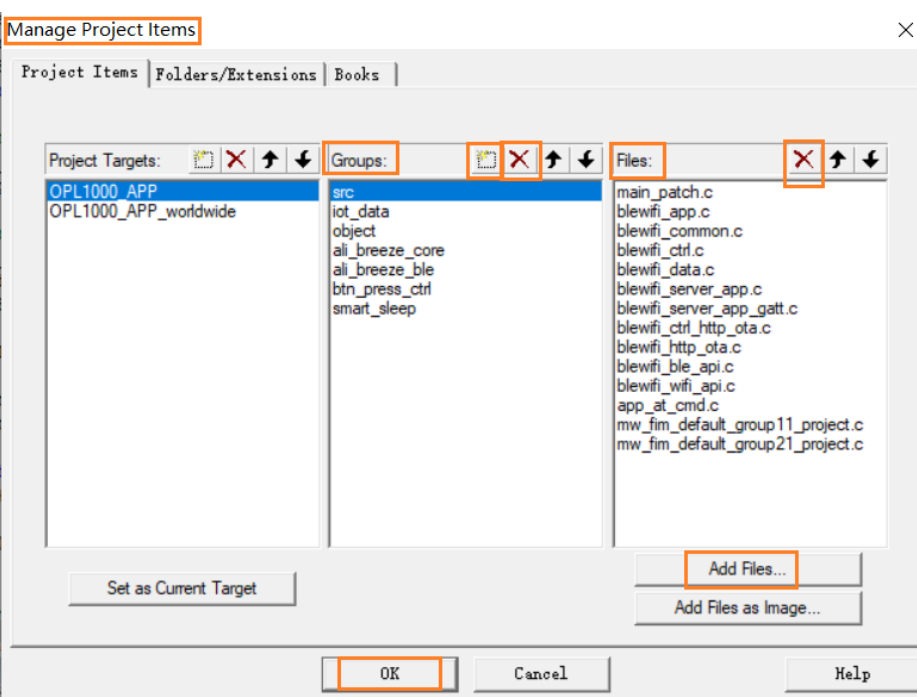


在 Manage Project Items 界面，用户根据需要：

- 添加/删除 Group;
- 在某个 Group 中添加/删除文件。

更新完成后点击“ok”按钮。。

Figure 26: 添加或删除工程文件



8.3. 根据需要修改文件

在应用程序开发过程中，用户也可能会根据需要添加或修改一些功能实现。关于改动代码的更多指导，请参考 [OPL1000-Reference-Sensor-Device-Ali-cloud-guide.pdf](#)。下面以 IO4 引脚为例，说明如何在收到数据（LED 灯的开关状态）后设置该引脚的电平。

- 1. 在 hal_pin_config_project_transparent.h 文件中把 IO4 设置为 PIN_TYPE_GPIO_OUTPUT_LOW

```
// #define HAL_PIN_TYPE_IO_4    PIN_TYPE_NONE           // PIN_TYPE_NONE
#define HAL_PIN_TYPE_IO_4    PIN_TYPE_GPIO_OUTPUT_LOW
```

- 2. 在 blewifi_configuration.h 文件中定义一个宏 PER_LED_IO_PORT 表示使用的是 IO4 引脚。

```
#define PER_LED_IO_PORT                (GPIO_IDX_04)
```

- 3. 在 iot_data\ali_linkkitsdk_impl.c 文件的 user_property_set_event_handler 函数中加入如下代码

```
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;
    /* start: Set IO4 to high if led is on. */
    if(u8On != tStatus.u8LedOn)
    {
        E_GpioLevel_t tLevel = GPIO_LEVEL_LOW;
        //E_GpioLevel_t tLevel = GPIO_LEVEL_HIGH;

        tStatus.u8LedOn = u8On;

        if(tStatus.u8LedOn)
        {
            // active high
            tLevel = GPIO_LEVEL_HIGH;
        }

        Hal_Vic_GpioOutput(PER_LED_IO_PORT, tLevel);
    }
    else
    {
        BLEWIFI_INFO("[%s %d] LED already[%u]\n", __func__, __LINE__,
tStatus.u8LedOn);
    }
    /* end: Set IO4 to high if led is on. */
}
```

文件更新完成后，即可参考上面的文档进行编译，验证相应的功能。

CONTACT

sales@Opulinks.com