

OPL1000

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

SDK Development Guide



OPULINKS

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

Copyright © 2017-2018, Opulinks. All Rights Reserved.

OPL1000-SDK-development-guide-R01 | Version V05

Date	Version	Contents Updated
2018-05-11	0.1	Initial Release
2018-05-15	0.2	- Update section 2.1 - Add section 6.4 to introduce log output setting
2018-05-23	0.3	Modify the section 6.4
2018-06-01	0.4	Update chapter 2 and section 3.2
2018-06-06	0.5	Update section 2.1, chapter 6 and 7.

TABLE OF CONTENTS

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 5

3. 应用程序开发流程 7

3.1. IDE 在线调试开发模式 7

3.2. 串口调试开发模式 8

4. 嵌入式操作系统 9

5. 外设参数配置 10

6. 应用开发基本配置 13

6.1. Keil uVision 工程配置 13

6.2. 工程预览 14

6.3. Main 函数入口 15

6.4. Log 输出设置 15

7. 应用举例 – WIFI+外设配置应用 19

7.1. 开发目标 19

7.2. 基于已有例程创建工程 19

7.3. 完善工程配置 20

7.4. 代码实现 21

7.5. 测试执行结果 22

LIST OF FIGURES

FIGURE 1:OPL1000 SDK 发布包目录.....2

FIGURE 2: IDE 在线开发模式7

FIGURE 3: 串口调试开发模式8

FIGURE 4:CMSIS RTOS 架构9

FIGURE 5:外设配置.....10

FIGURE 6:IO 配置11

FIGURE 7: DEVICE 选择13

FIGURE 8:SCATTER FILE14

FIGURE 9:DEBUG 设置14

FIGURE 10:HELLO WORLD 工程源码结构14

FIGURE 11:TRACER 命令16

FIGURE 12:工程结构.....20

LIST OF TABLES

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: SDK 目录下模块功能4

TABLE 4: EXAMPLE 目录下示例工程实现的功能4

TABLE 5:UART IO 关系.....10

TABLE 6: 地址表.....13

TABLE 7: TRACER_DEF_LEVEL_SET 函数入口参数定义17

TABLE 8: UART0 参数配置.....19

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
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

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] SDK 快速使用指南 OPL1000-SDK-getting-start-guide.pdf
- [6] WIFI/BLE API 使用说明 OPL1000-WIFI-BLE-API-guide.pdf
- [7] AT 命令和例程说明 OPL1000-AT-instruction-set-and-examples.pdf
- [8] OPL1000 系统初始化介绍 OPL1000-系統初始化流程簡介.pdf

2. OPL1000 SDK 软件包

2.1. 目录结构

OPL1000 SDK 软件包包含若干层级目录，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
 - BLE_Config_AP
 - TCP_Client
 - Doc
 - FW_Binary
- ▼ SDK
 - ▼ APS
 - > driver
 - > FreeRtos
 - > middleware
 - > project
 - tools
 - ▼ APS_PATCH
 - > driver
 - > examples
 - > FreeRtos
 - > middleware
 - > project
 - tools
- ▼ Tool
 - Ch340_winXP_Drivers
 - ▼ CP210x_Windows_Drivers
 - > Win7_Win10_x64
 - > WinXP_Vista
 - Download
 - 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 M0 和 M3 Patch 固件，以及固件合并使用的脚本文件。
3	SDK	存放 SDK 的 include 文件，部分源码，Patch lib 文件和示例代码。
4	Tool	存放管脚复用(Pin-Mux) 设置工具和固件下载工具。串口驱动程序。
5	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。
6	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。
7	README.md	简略介绍 SDK package 的内容

Fw_Binary 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 2: Fw_Binary 目录下文件列表

编号	文件名	说明
1	opl1000_app_m3.bin	OPL1000 M3 的固件补丁文件，原初版本支持 WIFI/BLE AT Command。用户如果在 M3 上开发应用程序，编译产生的 M3 bin 文件需要替换该文件。
2	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。
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。

SDK 目录下的两级子目录包含的功能模块如下表所述。

Table 3: 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	driver	OPL1000 外设驱动
2		FreeRtos	FreeRTOS 移植源码
3		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。
4		project	包含用于产生 M3 固件 patch lib 和 SDK lib 的工程文件
5		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	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。
2		examples	包含各种示例代码，如 WIFI、BLE、外设、协议层、系统应用等例程。
3		FreeRtos	FreeRTOS 应用的补丁程序。
4		middleware	中间件的补丁程序。
5		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 4 所述。

Table 4: 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	ble_adv	BLE Slave 设备发广播
2		blewifi	使用 BLE 完成 SSID 获取和 WIFI 连接功能
3		gatt_client	GATT Client 功能演示
4		gatt_server	GATT Server 功能演示

编号	第一级目录	第二级目录	说明
5		gatt_server_service_table	GATT Server 服务表功能演示
6	get_started	blink	控制 LED 灯闪烁功能演示
7		hello_world	多任务间通信打印 Hello World 字符串输出
8		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
9	peripheral	auxadc	辅助 ADC 使用例程
10		gpio	Gpio 设置演示
11		i2c	I2C master 功能演示
12		i2c_slave	I2C slave 功能演示，和外部 I2C Master 通信
13		pwm	PWM 端口设置展示
14		spi_master	SPI master 功能演示，访问 Flash 和外部 SPI slave 设备
15		timer_group	Timer 功能演示
16		uart	UART 配置和通信演示
17	protocols	http_request	http 访问 example.com 功能演示，基于 CMSIS RTOS
18		http_request_freertos	http 访问 example.com 功能演示，基于 Free RTOS
19		tcp_client	TCP client 功能演示
20	wifi	Wpa2_station	AP 为 WPA2 加密模式，OPL1000 作为 WIFI Station 连接 AP 功能演示
21		Wpa2_station_gpio	在 wpa2_station 示例基础上添加 GPIO 外设的使用

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

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 和 Download 工具软件

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 16 根 GPIO 管脚上进行分配和定义。用户可以根据自己的需要使用 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。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\[4\]](#)）。

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。

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。

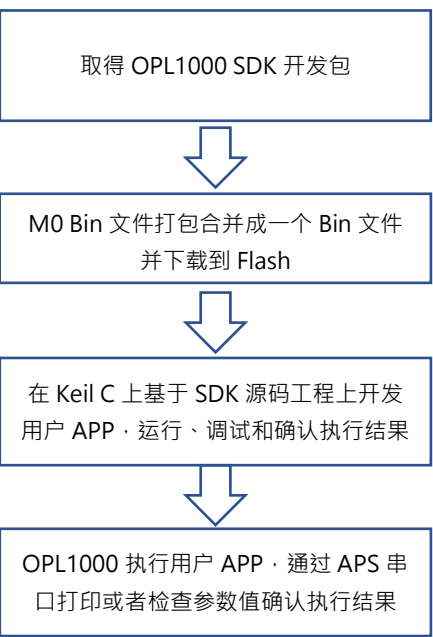
3.1. IDE 在线调试开发模式

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 Figure 2 表示：

Figure 2: IDE 在线开发模式



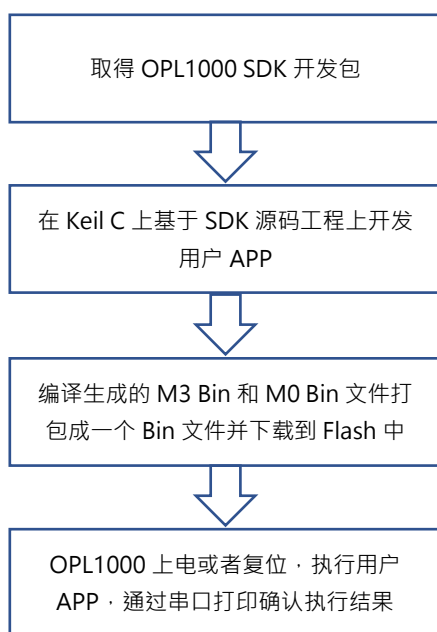
3.2. 串口调试开发模式

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 Figure 3 所示：

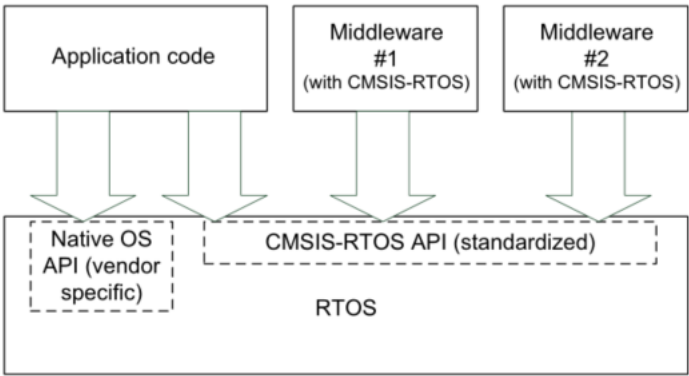
Figure 3: 串口调试开发模式



4. 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 Figure 4 所示：

Figure 4: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。

配置完成以后状态如图 Figure 6：

Figure 6:IO 配置

IO

UART

SPI

I2C


PWM

AUX/ADC

GPIO

SW Version: 0.3

	pin	IO2	IO3	IO4	IO5	IO6	IO7	IO8	IO9	IO10	IO11	IO18	IO19	IO20	IO21	IO22	IO23
1	UART0_TX	<input checked="" 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"/>		
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。

6. 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。

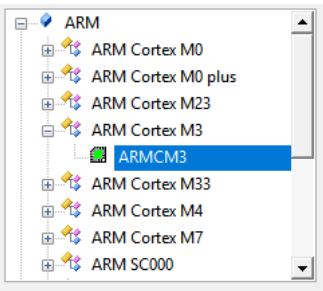
6.1. Keil uVision 工程配置

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 7: 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 6: 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。


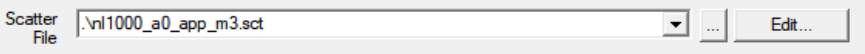
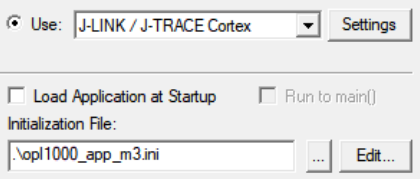
需要设置 Scatter File 文件，如图 Figure 8。点击  按钮可打开该 Scatter File。

Figure 8: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，如图 **Figure 9** 所示。如果需要了解 ini 内容，点击 **Edit...** 打开 ini 文件。

Figure 9:debug 设置

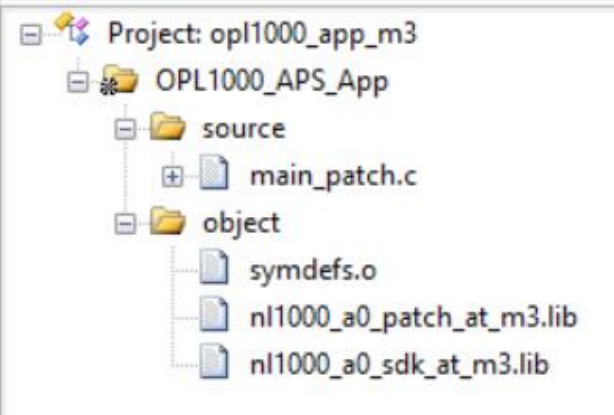


注意：鉴于该 ini 在 patch 代码 和 ROM 代码之间的重要性，不建议修改该文档内容，否则会引起代码无法正常运行的情况。

6.2. 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**，**nl1000_a0_patch_at_m3.lib**、**nl1000_a0_sdk_at_m3.lib**。

Figure 10:hello world 工程源码结构



其中 `main_patch.c` 为工程示例代码，`sysdefs.o` 和 `nl1000_a0_patch_at_m3.lib`、`nl1000_a0_sdk_at_m3.lib` 为库文件。

`sysdefs.o` 的目录地址：SDK\APS\project\nl1000\object

`nl1000_a0_patch_at_m3.lib` 和 `nl1000_a0_sdk_at_m3.lib` 所在目录地址：

SDK\APS\project\nl1000\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。

6.3. 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();
    // application init
    Main_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 `Main_AppInit` 入口到 `Main_AppInit_patch` 函数，`Main_AppInit` 是 SDK 留给用户端开发软件的 `main` 函数入口，通过映射 `Main_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。

6.4. 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 11: tracer 命令

```
Tracer Mode      [1]  0:disable/1:normal/2:print directly
Display Task Name [0]  0:not display/1:display
Tracer Priority  [-2] osPriorityIdle(-3) ~ osPriorityRealtime(3)

Default Level for Internal Tasks [0x07]
Default Level for App Tasks      [0x07]

Index           Name: Level
[ 0]            diag: 0x07
[ 1]            wifi_mac: 0x07
[ 2]            suplicant: 0x07
[ 3]            controller_task: 0x07
[ 4]            le: 0x07
[ 5]            event_loop: 0x07
[ 6]            tcpip_thread: 0x07
[ 7]            ping_thread: 0x07
[ 8]            iperf: 0x07
[ 9]            ISR_: 0x07
[10]            at_wifi_app: 0x07
[11]            AT: 0x07
[12]            at_tx_data: 0x07
[13]            socket: 0x07
[14]            server: 0x07

----- Start of App Tasks -----

[32]            user_app_demo: 0x07
[33]            user_app: 0x07

Tracer Command List:
tracer mode <0:disable/1:normal/2:print directly>
tracer disp_name <0:not display/1:display>
tracer def_level <0:internal tasks/1:app tasks> <level:hex>
tracer level <task_index> <level:hex>
tracer pri <osPriority:-3 ~ 3>
```

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：

```
#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)

typedef enum
{
    TRACER_TASK_TYPE_INTERNAL = 0,
    TRACER_TASK_TYPE_APP,

    TRACER_TASK_TYPE_MAX
} T_TracerTaskType;

tracer_def_level_set(uint8_t bType, uint8_t bLevel);
```

tracer_def_level_set 函数入口参数定义如表 Table 7 定义

Table 7: tracer_def_level_set 函数入口参数定义

参数	取值	说明
bType	TRACER_TASK_TYPE_INTERNAL	firmware 内部 log
	TRACER_TASK_TYPE_APP	用户 APP log
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_def_level_set(TRACER_TASK_TYPE_INTERNAL, LOG_ALL_LEVEL)
```

(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，需要把 APS_PATCH\middleware\netlink\msg\msg_patch.c 添加进工程源码，同时修改 msg_patch.c 的结构体 g_taTracerExtTaskInfoBody 为

```
T_TracerTaskInfo g_taTracerExtTaskInfoBody[TRACER_EXT_TASK_NUM_MAX] =
{
    {"user_app_1", LOG_ALL_LEVEL},
    {"user_app_2", LOG_ALL_LEVEL },
    {"", LOG_NONE_LEVEL},
};
```

其中的 user_app_1、user_app_2 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. 应用举例 – WIFI+ 外设配置应用

本章节以实现 WIFI 加一个 GPIO 控制为例，说明用户如何基于 OPL1000 SDK 开发一个自己的应用程序。

7.1. 开发目标

假设需要实现如下应用场景：

- OPL1000 作为 STA 连接 AP，连接成功后使用串口 0 打印 字符“UART0:OPL1000 connected!”，拉高 IO4。
- 断开 STA 和 AP 的连接，串口 0 打印字符“UART0:OPL1000 disconnect!”，拉低 IO4。

通过这个例子，用户可以了解如何基于已有的 WIFI 示例工程和外设控制范例实现一个简单应用。

7.2. 基于已有例程创建工程

SDK 已经提供了一个 WIFI STA 基本示例 `wpa2_station`，工程位于：

`SDK\APS_PATCH\examples\wifi` 目录下。在同一目录下复制 `wpa2_station` 文件夹并且重新命名为 `wpa2_station_gpio`。

使用 Pin-Mux Tool 配置 UART0 参数如 Table 8。所示。

Table 8: UART0 参数配置

属性	参数值
TX	IO2
RX	IO3
波特率	115200
数据位	8 BITS
停止位	1 STOP
奇偶校验	NONE

配置 IO4 参数为 `GPIO_OUT_PULLUP`。设置完成以后，pinmux 工具 生成 `OPL1000_pin_mux_define.c` 文件，并把该文件复制到 `wpa2_station_gpio` 目录下。

注意：PinMux GUI 具体设置方法请参考 4.4 节“[外设参数配置](#)”。

7.3. 完善工程配置



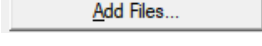
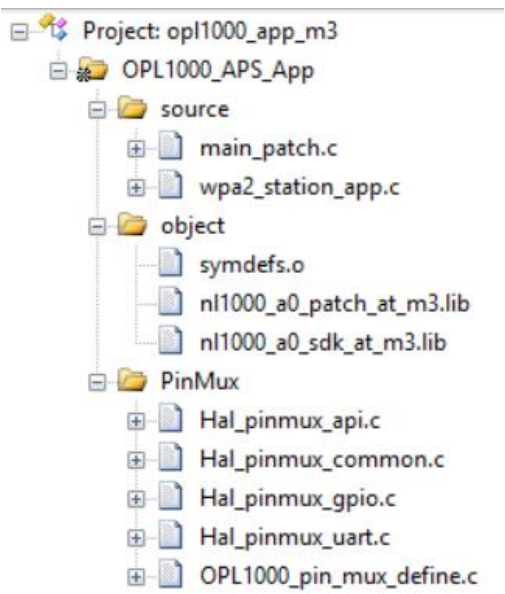



打开 `wpa2_station_gpio` 目录下的 Keil 工程，选择  按钮，在 Project Items -> Groups 容器中选择 ，新建一个名为 **PinMux** 的 Group。新建 Group 后，在 **PinMux** Group 被选中的状态下，点击 ，接着在弹出窗口中依次添加 `SDK\APS_PATCH\driver\chip\hal_pinmux` 文件夹中的 `Hal_pinmux_api.c`、`Hal_pinmux_common.c`、`Hal_pinmux_uart.c`、`Hal_pinmux_gpio.c` 以及 `wpa2_station_gpio` 目录下的 `OPL1000_pin_mux_define.c` 四个源文件到工程。添加完成以后工程文件树结构如图：

Figure 12:工程结构



添加文件完成，还需要在设置新添加源码所依赖的头文件地址，选择  按钮，切换到 C/C++ 标签，对应 include paths 项目选择  按钮，并在弹出对话框中新建地址 ，添加 PinMux 所依赖的头文件在 `SDK\APS_PATCH\driver\chip\hal_pinmux` 到新建的目录中。

7.4. 代码实现

完成前面章节所述步骤后，接下来修改工程代码实现。包括 5 个步骤：

1. 在 wpa2_station_app.h 添加依赖的头文件

```
#include "Hal_pinmux_uart.h"
#include "Hal_pinmux_gpio.h"
```

在 main_station_app.c 新建函数 App_Pin_InitConfig(void)。

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

    /*IO4 Init*/
    Hal_PinMux_Gpio_Init(&OPL1000_periph.gpio[0]);
}
```

2. 把 App_Pin_InitConfig(void) 添加到 Main_AppInit_patch(void) 中。

```
static void Main_AppInit_patch(void)
{
    // init the pin assignment
    App_Pin_InitConfig();

    // wifi init
    WifiAppInit();
}
```

3. 接下来在 wpa2_station_app.c 里面添加如下代码，以便使用 UART0 发送字符串。

```
void uart_printf(E_UartIdx_t port, char data[])
{
    uint8_t index = 0;
    while(data[index] != 0)
    {
        Hal_Uart_DataSend(port, data[index]);
        index++;
    }
}
```

4. 因为需要在 OPL1000 连接到 AP 时用 UART0 打印 "UART0 OPL1000 connected!", 并且拉高 IO4，因此需要在 wifi_event_handler_cb 函数的 WIFI_EVENT_STA_CONNECTED 分支中添加代码：

```
uart_printf(UART_IDX_0, " UART0:OPL1000 connected!  \r\n");
Hal_Vic_GpioOutput(GPIO_IDX_04, 1);
```

并且在 **WIFI_EVENT_STA_DISCONNECTED** 分支中添加代码：

```
uart_printf(UART_IDX_0, " UART0:OPL1000 disconnect! \r\n");  
Hal_Vic_GpioOutput(GPIO_IDX_04,0);
```

5. 最后修改 `wpa2_station_app.h` 里面修改 STA 将要连接的 AP 名称和密码

```
#define WIFI_SSID          "WifiName"  
#define WIFI_PASSWORD     "password"
```

注意:WIFI 相关 API 的头文件路径为：`SDK\APS_PATCH\middleware\netlink\wifi_controller_layer`

7.5. 测试执行结果

对工程编译并且使用 `download tool` 工具下载 M3 bin 到 DEVKIT 开发板。复位开发板，OPL1000 的 APS 端口实时打印相应的连接信息。使用串口调试工具可以在 UART0 端口看到如下 log：

```
UART0: OPL1000 connected!  
UART0: OPL1000 disconnected!  
...
```

至此一个基于 WIFI 示例工程的简单 wifi 应用程序就开发完成了。

CONTACT

sales@Opulinks.com