# OPL1000

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

# SDK Development Guide

**OPULINKS**

**OPL1000-SDK-development-guide-R01︱ Version V10**

| Date | Version | Contents Updated |
|---|---|---|
| 2018-05-11 | 0.1 | Initial Release |
| 2018-05-15 | 0.2 | - Update section 2.1<br>- 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. |
| 2018-07-12 | 0.6 | Update section 2.1, SDK project folder structure and example list are updated. |
| 2018-08-03 | 0.7 | Update Section2, 6, and 7 for A1 chip |
| 2018-08-17 | 0.8 | - Update section 2 because SDK package is updated. |
| 2018-10-22 | 0.9 | - Update section 2 --Table2 and Table4 |
| 2018-11-14 | 1.0 | - Update section 2 because multiple device download tool has been added |

# TABLE OF CONTENTS

## LIST OF FIGURES

## LIST OF TABLES

# 1. INTRODUCTION

## 1.1. Scope of Document Application

This document introduces how to embed M3 MCU, based on SDK software development application procedure. The content includes, SDK software module function introduction, based on example transfer and development application procedure provided by SDK.

## 1.2. Abbreviations

| Abbr. | Explanation |
|---|---|
| APP | APPlication |
| CMSIS | Cortex Microcontroller Interface Standard Cortex |
| DEVKIT | DEVelopment Kit |
| EVB | Evaluation Board |
| FW | FirmWare Embedded software operating on CPU |
| OTA | Over the Air Technology |

## 1.3. References

[1] OPL1000-DEVKIT-getting-start-guide.pdf

[2] CMSIS-RTOS  http://www.keil.com/pack/doc/CMSIS/RTOS/html/index.html

[3] OPL1000-patch-download-tool-user-guide.pdf

[4] OPL1000-pinmux-tool-user-guide.pdf

[5] OPL1000-multiple-dev-download-tool-user-guide .pdf

[6] OPL1000-SDK-getting-start-guide.pdf

[7] OPL1000-WIFI-BLE-API-guide.pdf

[8] OPL1000-AT-instruction-set-and-examples.pdf

[9] OPL1000-system-initialization-brief-introduction.pdf

# 2.    OPL1000 SDK SOFTWARE PACK

## 2.1.  Folder Structure

OPL1000 SDK software include certain layers of folders, and this chapter will introduce 3 layers of folders, as Figure 1 display the 3-layer folder structure of SDK publication pack.

Figure 1: OPL1000  SDK Publication Pack Folder

```
> 📁 Demo
  📁 Doc
  📁 FW_Binary
∨ 📁 SDK
  ∨ 📁 APS
    > 📁 apps
    > 📁 driver
    > 📁 FreeRtos
    > 📁 middleware
    > 📁 project
    > 📁 targets
      📁 tools
  ∨ 📁 APS_PATCH
    > 📁 apps
    > 📁 driver
    > 📁 examples
    > 📁 FreeRtos
    > 📁 middleware
    > 📁 project
∨ 📁 Tool
    📁 Ch340_winXP_Drivers
  > 📁 CP210x_Windows_Drivers
    📁 Download
    📁 Mass_Production
    📁 PinMux
```
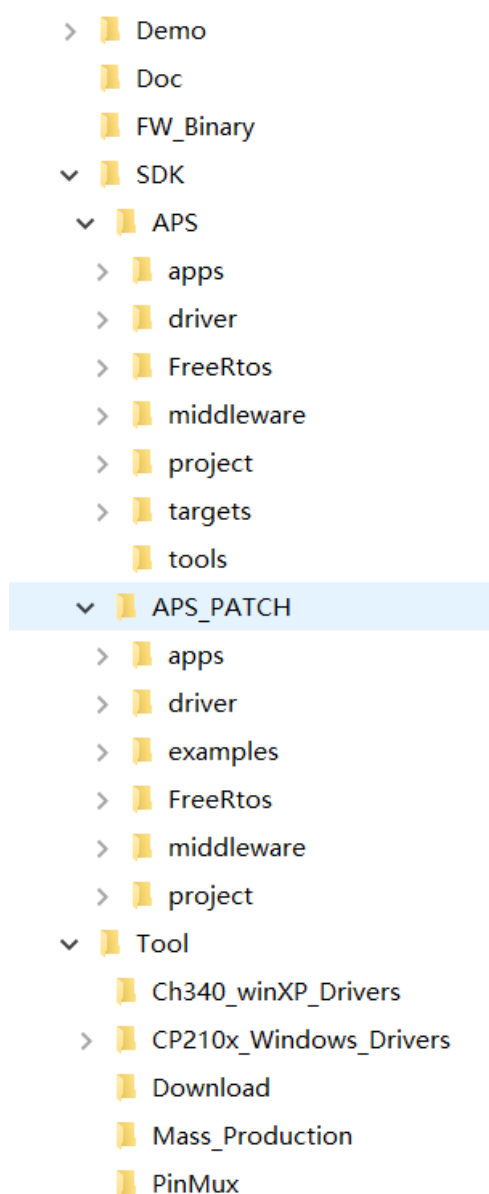
Documents and their functions include in Layer-1 folder are shown in Table 1.

Table 1: OPL1000 SDK Software Class-I Folder and Its Functions

| Number | Folder Name | Description |
|---|---|---|
| 1 | Demo | Some of the function demonstration examples of OPL1000, e.g. WIFI connection via BLE, TCP Client Connection and communication |
| 2 | FW_Binary | Contains M0 & M3 Patch firmware, OTA loader and firmware combined with script documents |
| 3 | SDK | Contains include document of SDK, some source codes, Patch lib document and example codes |
| 4 | Tool | Contains pin-Mux set-up tool and firmware download tool, multiple-device firmware download tool, and serial-port driver, etc. |
| 5 | Doc | Contains SDK API usage guide, user application procedure coding and development guide, etc. |
| 6 | Release_Notes.md | SDK software publication document, listed update items, and functions and characteristics supported by the publication pack |
| 7 | README.md | Brief introduction of the content of SDK package |

Fw_Binary folder contains M3, M0 firmware patch Bin document and combined script document. The combined scrip document, "PatchData.txt" is complementary to the published Bin document, as different version of M3/M0 Bin document and PatchData.txt cannot be mixed up in usage. When user is using download tool to download firmware, the same version of the published combined script document.

OPL1000-SDK-development-guide-R01, Version V10

**3**

Table 2: Fw_Binary Document List under FW_Binary Folder

| Number | Document Name | Description |
|---|---|---|
| 1 | opl1000_app_m3.bin | For the firmware patch document of OPL1000 M3, the initial version supports WIFI/BLE AT Command. If the user develops application procedure on M3, the code-generated M3 bin document should replace this document. |
| 2 | opl1000_m0.bin | OPL1000 M0 firmware patch document provided by original manufacturer. M0 Patch Bin document needs to be combined with opl1000_app_m3.bin to be downloaded to external Flash, and load the combined Bin document from Flash after OPL1000 is reset. |
| 3 | PatchData.txt | M3, M0 Bin document combined with script document. It is used to define how to combine Bin documents of M3 and M0 to a single patch document. This document is adopted by the Pack function of Download Tool. |
| 4 | Opl1000_ota_loader.bin | Used as the guiding procedure to load OTA firmware. |
| 5 | Opl1000_at.bin | Opl1000 firmware provided by original manufacturer supports all AT commands. |

The functional module contained in the Class-II sub-folder under SDK folder.

Table 3: SDK Module Function Under SDK Folder

| Number | Class-I Folder Name | Class-II Folder Name | Description |
|---|---|---|---|
| 1 | APS | Apps | Function header file contained in ROM code related to WIFI and LE applications |
| 2 | | driver | OPL1000 external driver |
| 3 | | FreeRtos | FreeRTOS transplant source code |
| 4 | | middleware | Middleware file, such as WIFI and BLE protocols, AT commands and control layer, etc., provided by Opulinks and third-parties (e.g. Lwip,fatfs,tinycrypt, etc.) |

| Number | Class-I Folder Name | Class-II Folder Name | Description |
|---|---|---|---|
| 5 | | project | Including patch lib and SDK lib project documents used to generate M3 firmware |
| 6 | | targets | Storing generated patch lib and SDKlib documents |
| 7 | | tools | Containing 2 documents, (1) srec_cat.ext tool that converts bin document to Hex, (2) SVN version number subwcrev.exe defined by header file in the automatically generated project documents |
| 1 | APS_PATCH | Apps | Contain OTA loader (boot agent) materialization example, LE and WIFI application patch procedure, etc. |
| 2 | | driver | Patch procedure driven externally of OPL1000, when the external driver in ROM_CODE does not need to be patched, the corresponding folder would be empty. |
| 3 | | BA | Project items used to generate OTA loader, including materializing source code. |
| 4 | | examples | Including every type of example codes, such as example procedures such as WIFI, BLE, external, protocol layer, system application, etc. |
| 5 | | FreeRtos | Patch procedure of FreeRTOS application |
| 6 | | middleware | Patch file of middle ware |
| 7 | | project | Patch procedure of project document corresponding to Project under APS folder |

Regarding the folders of the same name containing APS and APS_PATCH, what needs to be explained is that, the source codes under APS folder has been coded and burned to ROM of OPL1000, and the source code under APS folder can only not be modified, but be reference for users. ROM patch code provided by SDK corresponds to the APS folder of the same name under APS_PATCH folder.

Regarding certain demonstrating document provided under the folder of "SDK\APS_PATCH\examples", user can reference the demonstrating document to transplant and develop his/her own application procedure. The functions realized by theses demonstrating projects are outlined in Table 4.

Table 4: Functions Realized by Example Projects under Example Folder

| Number | Class-I Folder | Class-II Folder | Description |
|---|---|---|---|
| 1 | bluetooth | ble_adv | BLE Slave Device Broadcast |
| 2 | | gatt_client | GATT Client Function Demonstration |
| 3 | | gatt_server | GATT Server Function Demonstration |
| 4 | | gatt_server_service_table | GATT Server Service Table Function Demonstration |
| 5 | get_started | blink | Function Demonstration of the blinking of LED control light |
| 6 | | hello_world | "Hello World" characters of multi-mission communication print-out |
| 7 | | log | Display how to activate/de-activate the internal module of firmware and log output message of user missions. |
| 8 | peripheral | auxadc | Usage demonstration of Assistance to ADC |
| 9 | | gpio | Gpio set-up demonstration |
| 10 | | i2c | I2C master function demonstration |
| 11 | | i2c_slave | I2C slave function demonstration, and external I2C Master communication |
| 12 | | pwm | PWM port set-up demonstration |
| 13 | | spi_master | SPI master function demonstration, access Flash and external SPI slave device |
| 14 | | timer_group | Timer Function Demonstration |
| 15 | | uart | UART set-up and communication demonstration |
| 16 | protocols | http_request | Function Demonstration of http interview example.com, based on CCMSIS RTOS |
| 17 | | http_request_freertos | Function Demonstration of http interview example.com, based on Free RTOS |
| 18 | | tcp_client | TCP client Function Demonstration |
| 19 | | sntp_get_calender | Demonstration example of SNTP date acquisition |

OPL1000-SDK-development-guide-R01, Version V10

6

| Number | Class-I Folder | Class-II Folder | Description |
|---|---|---|---|
| 20 | | mdns | Demonstration example of mDNS sales broadcast |
| 21 | | https_request | Function Demonstration of http interview example.com, based on CMSIS RTOS |
| 22 | wifi | iperf | Demonstration example of network performance testing tool iperf |
| 23 | | wpa2_station | AP as WPA2 encryption mode, as OPL1000 being the functional demonstration of WIFI Station connecting to AP. |
| 24 | | wpa2_station_gpio | Addition of external usage of GPIO based on wpa2_station demonstration. |
| 25 | system | ota_wifi | Demonstration example based on WIFI OTA download |
| 26 | | blewifi | Completion of SSID acquisition and WIFI connective function, BTA OTA and WIFI OTA functions, via BLE. |

## 2.2.  SDK Publication Pack Usage

After users receive SDK publication pack, we recommend going through the following steps to understand and grasp OPL1000 functions. On the basis of understanding example projects provided by SDK to develop customized application procedure based on suitable example transplant selected.

**Step 1: Understand DevKit usage and user APP development process**

First, by reading reference (1) "DEVKIT Quick Usage Guide", and reference (5) "SDK Quick Usage Guide", to understand the following knowledge:

（1） Procedure to develop and code application software on DEVKIT board, and get to know how to download the latest published firmware to update OPL1000 DEVKIT.

（2） Ways to set up user APP engineering documents, and how to develop customized applications based on OPL1000SDK demonstration projects.

**Step 2:  Use Pin-Mux, Download tool software and MP download tool**

Normally, IOT application will adopt peripheral resources of OPL1000, such as SPI, I2C, UART, PWM, GPIO, and AUX ADC, which can be distributed and defined on 18 GPIO pin featured on

OPL1000. Users can adopt Pin-Mux tool to distribute pin function and defined peripheral operating mode, based on their needs.

The user-developed application procedure is ultimately compiled into a firmware program executed on M3. It needs to be combined with the M0 Bin document provided by vendor to generate a firmware patch procedure, before being downloaded to Flash, and then loaded into RAM for execution after power-on. If the user-developed application procedure needs to support OTA download, the OTA Pack function provided by download tool shall be adopted, therefore, the users need to master the usage of Download tool.

The manuals for these two tools can be accessed by clicking on the "use manual" button provided in the software interface. At the same time, A PDF version of the help file is also available in the Doc directory (References [3] and [4]).

During multiple-device-download, the vendor uses mp download tool, and after the hardware is connected, the multiple-device download tool can be used to download on multiple devices simultaneously. The use of this tool is available in PDF format in the tool directory. (Ref. [5]).

Step 3: Execute AT command or compile the execution example project evaluation by using OPL1000

Through the AT serial port on DEVLKIT, the AT command can be issued to control OPL1000 to execute specific WIFI and BLE functions. For more details, please refer to the reference [7] "AT Command and Example Description" document. Users can also directly compile the sample code provided by the SDK, before downloading it to OPL1000 for execution. The second evaluation method is convenient for users to understand the API functions provided by the underlying layer, and how to adopt API to implement specific functions. The introduction on API functions can be found in Reference [6] "WIFI/BLE API Instructions for Use".

**Step 4: Transplant and develop applications based on the demonstration projects**

As described in Section 2.1, the SDK provides several demonstration projects, and users can select one or several of them based on their needs, to tailor, combine and transplant development of their own application. Specific procedure can be referred to Chapters 4 and 5 of this document.

OPL1000-SDK-development-guide-R01, Version V10

9

# 3. APPLICATION PROCEDURE DEVELOPMENT FLOW

There are two development modes for OPL1000 application procedure, as one is the IDE online debugging mode using SWD debug port, and the other is the print debugging mode through serial port offline. The former applies to the initial, not so mature, stage of development of the applications, and the latter applies to the relatively mature applications with relatively more complex implementation.
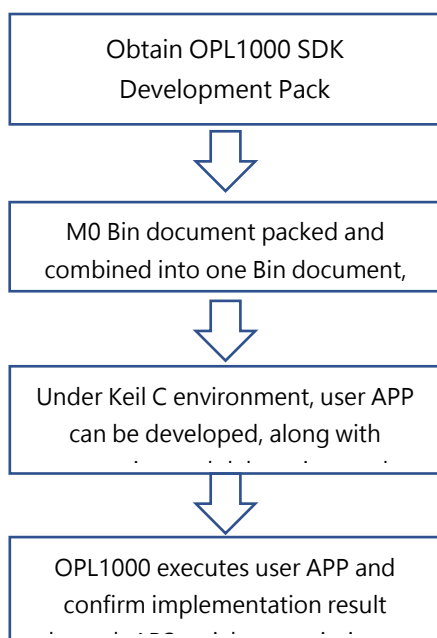
## 3.1. IDE Online Debug Development Mode

IDE online debugging development mode includes the following four steps:

1. Obtain OPL1000 SDK from the Opulinks.

2. Combine M0 Bin file in the "SDK FWBinary" directory into a Bin file and download it to Flash, according to reference [3] "Download Tool Usage Guide".

3. Develop user APP based on the SDK under Keil C environment.

4. Under Keil C environment, the online simulation operations and debugging confirms implementation result through IDE check of variation values, or through APS serial-ports. Note that, at this time, M3 firmware is not downloaded to flash, as the code is only executed in RAM, and may be lost after power-down.

These 4 steps can be illustrated with Figure 2.

Figure 2: IDE Online Development Mode

```
┌─────────────────────────┐
│   Obtain OPL1000 SDK    │
│    Development Pack      │
└─────────────────────────┘
            ⇓
┌─────────────────────────┐
│  M0 Bin document packed │
│   and combined into one │
│      Bin document,      │
└─────────────────────────┘
            ⇓
┌─────────────────────────┐
│ Under Keil C environment,│
│    user APP can be       │
│    developed, along with │
└─────────────────────────┘
            ⇓
┌─────────────────────────┐
│ OPL1000 executes user   │
│ APP and confirm         │
│ implementation result   │
└─────────────────────────┘
```

## 3.2. Serial-Port Debug Development Mode

The serial-port debug development mode includes the following five steps.

1. Obtain OPL1000 SDK from the original manufacturer.

2. Develop user APP on Keil uVision, the development process can be completely independent operating from DEVKIT board.

3. After the source code development is completed, M3 Bin file can be obtained after the compilation is completed.

4. Combine M0 Bin file in the "FWBinary" directory of the SDK package and the M3 bin file developed by the user into a Bin file, before downloaded to Flash, according to reference [3] "Download Tool Usage Guide".

5. Power-on or reset OPL1000 to execute user APP, and confirm the implementation result through serial port log information.

The whole process is illustrated in Figure 3.

Figure 3: Serial-Port Debug Development Mode

```
┌─────────────────────────────────┐
│  Obtain OPL1000 SDK development  │
│              pack               │
└─────────────────────────────────┘
                 │
                 ▼
┌─────────────────────────────────┐
│ Under Keil C environment, user APP │
│   can be developed based on SDK   │
└─────────────────────────────────┘
                 │
                 ▼
┌─────────────────────────────────┐
│  Coded M3 Bin combined with M0   │
│    Bin document into one Bin     │
└─────────────────────────────────┘
                 │
                 ▼
┌─────────────────────────────────┐
│ Power-on or reset OPL1000 to execute │
│     user APP, and confirm the    │
│ implementation result through serial │
└─────────────────────────────────┘
```

OPL1000-SDK-development-guide-R01, Version V10

12
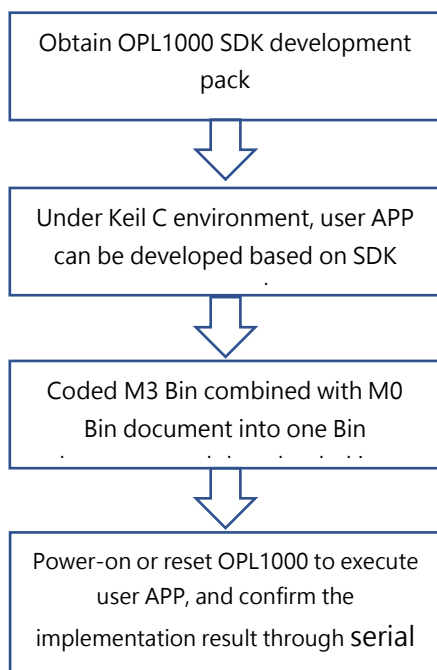
## 4.    EMBEDDED OPERATING SYSTEM

The bottom layer of the multitasking operating system of OPL1000 software adopts the FreeRTOS embedded operating system, with the upper layer featuring CMSIS-RTOS API to package FreeRTOS, while CMSIS defines the RTOS API, conforming to micro-processor standard, based on Cortex-M architecture. The CMSIS RTOS API does not rely on hardware-layer interfaces, but supports code re-use, while reducing the learning cost for mastering real-time the different embedded operating systems. The hierarchical relationship between CMSIS RTOS and third-party RTOS is shown in Figure 4:

Figure 4: CMSIS RTOS Architecture
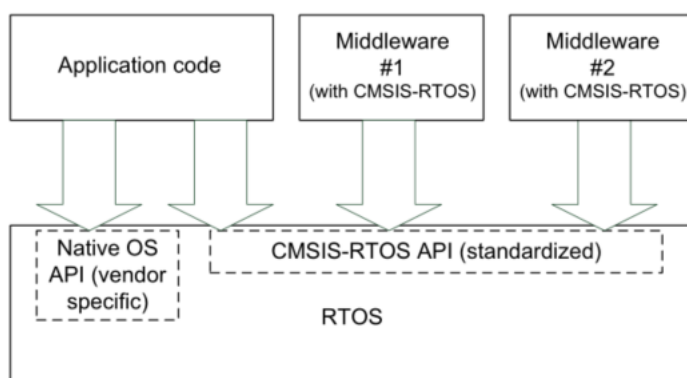


CMSIS RTOS interface, provided by SDK, includes:

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

If more insight into CMSIS-RTOS is needed, reference [2] provides more CMSIS-RTOS API Version 1 related content. And it should be noted that, since OPL1000 SDK already includes the CMSIS RTOS source code, we do not need to import the CMSIS RTOS from Keil PACK, otherwise it may result in conflicts due to version differences.

## 5.   PERIPHERAL DESIGNATED PARAMETER SETUP

The GPIO of the OPL1000 can be flexibly used in repetition according to the users' usage scenarios, and when using GPIO, the corresponding GPIO set up external to OPL1000 needs to be configured first by using PinMux tool. In the example of UART, firstly, PinMux tool provided by the SDK can be used to enable UART0 and UART1 in the UART option in configuring parameters such as baud rate and stop bit.

Figure 5: Peripheral Setup



After parameters are set, it can be switched to IO option in setting up IO in the following corresponding relationship.

| Externally Set | IO |
| --- | --- |
| UART0_TX | IO2 |
| UART0_RX | IO3 |
| UART1_TX | IO4 |
| UART1_RX | IO5 |

Once set-up is completed, the status can be shown in Figure 6:

Figure 6: IO Setup



Click [ Build ] to generate "OPL1000_pin_mux_define.c" in the current directory of the PinMux

GUI. The structure OPL1000_periph contained in this document defines OPL1000 peripheral pin

configuration and parameter settings, and the corresponding UART part has been filled with the

pin parameters and attribute parameters of UART0 and UART1 just set up.

```
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}
},
```

Add the file, "OPL1000_pin_mux_define.c" to the user's Keil project, call up the "Hal_PinMux_Uart_Init" function in the peripheral initialization code, before transmitting in the UART structure parameters.

```
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]);
}
```

After the function obtains the configuration parameters from the structure of OPL1000_periph, it completes the PinMux setting and attribute parameter setting operation of GPIO.

Users need to configure other peripheral operation steps to be the same as the set-up UART described above.

## 6. BASIC SETUP OF APPLICATION DEVELOPMENT

When users develop APP, it is recommended to develop with the Keil uVision 5 IDE tool. This chapter introduces the Keil uVision-based user APP project configuration method, and combines the hello_world demonstration project to briefly explain the project configuration, document structure, and RTOS usage, etc.
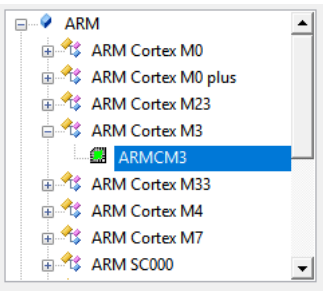
### 6.1. Keil uVision Project Setup

Open hello_world demonstration project.

Path: DK\APS_PATCH\examples\get_started\hello_world

Click the button, "  ", with the following set-up.

Device, by default select ARMCM3, and for new project, users should be aware of the selection of this setting.

Figure 7: Device Selection



1. Target page sets up the address of size of the ROM address and the RAM address.

Table 6:  Address Table

| Type | Initialization Address | SIZE |
|------|------------------------|---------|
| IROM1 | 0x0 | 0xC0000 |
| IRAM1 | 0x400000 | 0x50000 |

2.  Linker label does not need to select "use Memory layout From Target Dialog" option.

Scatter File document needs to be set up, as shown in Figure 8. By clicking Edit... button, this Scatter File can be opened up.

Figure 8: Scatter File



3.  Debug label does not select "Load Application at Startup" option, but the content of "initialization File" needs to be set up, as shown in Figure 9, and if there is need to understand the content of "ini" further, click " Edit... ", and open "ini" document.

Figure 9: debug Setup



Note: In view of the importance of "ini" between patch code and ROM code, it is recommended NOT to modify the content of this document; otherwise it may result in abnormal operation of the codes.

## 6.2.  Project Preview

In Chapter 6.1, by using the example of "hello world",  Kei project set-up method was outlined, and this chapter will be devoted to introduce the code structure of "hello_world" project. One basic project at least includes these documents such as "main_patch.c", "sysdefs.o" , "opl1000_patch_m3.lib", and "opl1000_sdk_m3.lib".

Figure 10: hello world Project Source Code Strucure
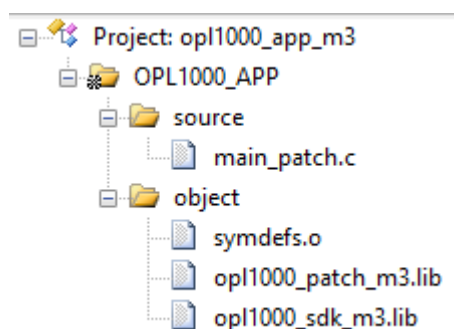


Amongst those, "main_patch.c" is project demonstration code, and "sysdefs.o", "opl1000_patch_m3.lib" and "opl1000_sdk_m3.lib' are stored documents.

The folder address for sysdefs.o : SDK\APS\targets\opl1000\object\

Address of folder for "**opl1000_patch_m3.lib**"& "**opl1000_sdk_m3.lib**": SDK\APS\targets\opl1000\Output\Objects\

If users need to newly build their own project, please add documents based on addresses outlined above.

## 6.3.  Main Function Entrance

Open up "main_patch.c" document, and jump to "_Patcch_EntryPoint(void)" function.

```
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;
}
```

As the SysInit_EntryPoint() function has been implemented in ROM. This function is only called to implement initialization, so the function should not be modified or move. Then as the code redefines the "Sys_AppInit" entry to the "Main_AppInit_patch" function, "Sys_AppInit" is the main function entry reserved by the SDK for software development on the user-end, and after mapping the entry function of "Sys_AppInit", all APP initializations, such as peripheral initialization, creation of multitasking, etc. are all created internally in the mapping function of Main_AppInit_patch.

## 6.4.  Log Output Setup

When users are developing APP, there are two types of debug message output.


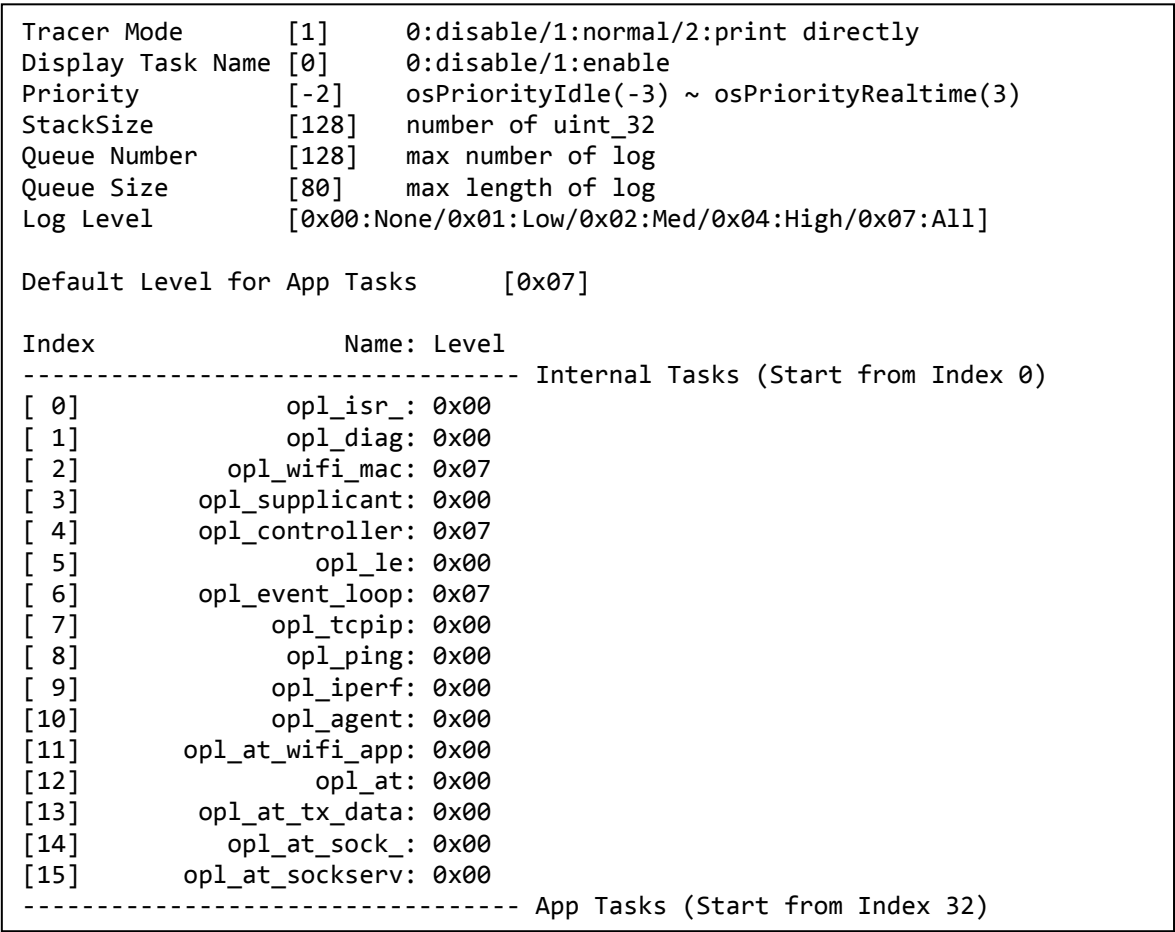(1)  Internal log of user APP

(2)  OPL1000 SDK firmware log

Log debugging print-out information is instrumental for users to check whether the application execution process and results are normal. The firmware log can help users quickly locate and analyze firmware module execution status, such as the operation of the ble and wifi protocol stacks.

In order to avoid excessive log output information that is not constructive for APP debugging, SDK only keeps the user log message output, and closes the internal log information output inside the firmware. If users need to manage log output mechanism, there are three methods:

(1)  APS serial-port tracer command

After command is input, APS serial-port lists out all of missions that are currently carried out.

Figure 11:TRACER Command

```
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_supplicant: 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)
```

The user-created tasks are listed in the Start of App Tasks project, and with the example of user_app_demo task, it is created by users in the main procedure with index of 32, while 0x07 denotes that printing all the logs of the task, so if users need to close the log output of the task. , command of "Tracer level 32 0x00" can be input in APS serial-port, which can also be applied in the log management of other tasks.

Note: The premise for using this method is that the user needs to enable the input function of the

APS serial-port when the main program is initialized, otherwise the input becomes invalid.

At the same time, when the configuration loses power, it needs to be reconfigured when it is

powered on next.

```
#include "hal_dbg_uart_patch.h"
static void Main_AppInit_patch(void)
{

    Hal_DbgUart_RxIntEn(1); // APS uart rx enable

}
```

(2)  Using log output provided by SDK to configure API management task log

API relevant content configured by log output is as follows:

```
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" function entry parameters are defined in Table 7.

Table 7: tracer_log_level_set_ext Function Entry Parameter Definition

| Parameter | Value | Description |
|-----------|-------|-------------|
| bIdx | 0 ~ TRACER_TASK_IDX_MAX | Internal module index number |
| bLevel | LOG_ALL_LEVEL | Print all log |
| | LOG_NONE_LEVEL | Close log |

The method users through API enable internal log of firmware:

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

(3)  Multi-task log management

If there needs to be differentiated management towards log output of multiple tasks, the structure, "g_taTracerExtTaskInfoBodyExt", of "msg_patch.c"needs to be edited. Adding user-defined tasks.

```
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},
};
```

The name of task, created for users by demo_app1 and demo_app2, can be configured and defined as:

```
#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)
```

The log management of users' other tasks is added to the structure in the manner as outlined above.

If needed to know more about the Log configuration, please refer to the log configuration demonstration with the directory of "SDK\APS_PATCH\examples\get_started\log"

# 7. APPLICATION EXAMPLES – WIFI+ EXTERNALLY DESIGNATED SET-UP APPLICATION

This chapter takes WIFI being added a GPIO control as an example to illustrate how users can develop their own applications based on OPL1000 SDK.

## 7.1. Development Objectives

If the following application scenarios need to be realized:

- As OPL1000 is connected to the AP as an STA, after the connection is successful, the serial-port 0 is used to print the characters, "OPL1000 connected, set GPIO4 as high", and IO4 is pulled high

  .
- Disconnect the connection between STA and AP, the characters of "OPL1000 disconnected, set GPIO4 low" will be displayed in serial-port 0, before pulling down IO4

Through this example, users can learn how to implement a simple application based on the existing WIFI example project and peripheral control examples.

## 7.2. Creating Projects based on Existing Examples

SDK already provide a WIFISTA basic example, "**wpa2_station**", and the project can be found in the folder of "**SDK\APS_PATCH\examples\wifi**", and under the same folder, "**wpa2_station**" document can be copied and opened, before renaming it as "**wpa2_station_gpio**".

To adopt Pin-Mux Tool set-up with UART0 parameters, please refer to Table 8.

Table 8: UART0 Parameter Setup

| Attribute | Parameter Value |
|---|---|
| TX | IO2 |
| RX | IO3 |
| Baudrate | 115200 |
| Data bit | 8 BITS |
| Stop bit | 1 STOP |
| odd-even calibration | NONE |

Configure the IO4 parameter as "GPIO_OUT_PULLUP", and after the setup is complete, pinmux tool generates "**OPL1000_pin_mux_define.c**" document, and copies it to the folder, "**wpa2_station_gpio**".

Note: Please refer to Section 4.4 of "Peripheral Parameter Configuration" for the specific setting method of PinMux GUI.

## 7.3.   Complete Project Setup
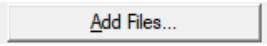
By opening Keil project in the folder of "**wpa2_station_gpio**" , the button, " ", should be selected, before creating a new Group called "**PinMux**" in "Project Items -> Groups" container. After creating a new group, with the "**PinMux Group**" selected, click " Add Files... ", and then respectively add the "Hal_pinmux_api.c, Hal_pinmux_common.c", "Hal_pinmux_uart.c", "Hal_pinmux_gpio.c" documents in the "SDK\APS_PATCH\driver\chip\hal_pinmux" folder in the prompted window. And the four source files of OPL1000_pin_mux_define.c in the "wpa2_station_gpio" folder to the project. After the addition is completed, the project file tree structure would be that as follows:

Figure 12: Project Structure
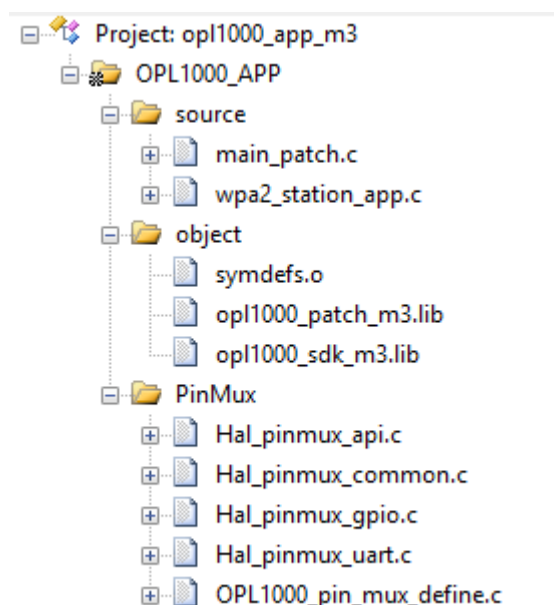


Once the document addition is complete, it is necessary set the header file address on which the new source code is added, by selecting the "  " button, and switching to the C/C++ tag, in correspondence to the "include paths" item and selecting "  " button, while creating new address, "  ", in the prompted dialog box to add the header document that PinMux depends on, which is the folder of "SDK\APS_PATCH\driver\chip\hal_pinmux", to the newly created folder.

## 7.4.  Designated Code Realization

After completing the described steps outlined in the previous chapter, subsequently the modification of project code realization involves 5 steps:

1.   Add the reliant header document in "**wpa2_station_app.h**"

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

Build new function of "**App_Pin_InitConfig(void)**" in "**main_station_app.c**"

```
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.  Add "App_Pin_InitConfig(void)" to "App_Pin_InitConfig(void)"

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

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

3.  For outputting the log output message of the internal module of wifi mac, add the following

    code in "Main_AppInit_patch".

```
static void Main_AppInit_patch(void)
{
    // …
    Internal_Module_Log_Config("opl_wifi_mac",true);
    Internal_Module_Log_Config("opl_controller_task",true);
    Internal_Module_Log_Config("opl_event_loop",true);
    // …
```

4.  As it is necessary to output messages of "OPL1000 connected, set GPIO4 high" and

    "OPL1000 disconnected, set GPIO4 low" when OPL1000 connects/disconnects with AP, the

    corresponding messages shall be output in user_wifi_app_entry, based on g_connect_flag

    (recording the status of connection or disconnection with AP。

```
static void user_wifi_app_entry(void *args)
{
    /* Tcpip stack and net interface initialization,  dhcp client process
initialization. */
    lwip_network_init(WIFI_MODE_STA);
    /* Waiting for connection & got IP from DHCP server */
    lwip_net_ready();
    while (1) {
       if(g_connect_flag == true )
       {
         printf("OPL1000 connected, set GPIO4 high \r\n");
         Hal_Vic_GpioOutput(GPIO_IDX_04,GPIO_LEVEL_HIGH);
       }
       else
       {
         printf("OPL1000 disconnected, set GPIO4 low \r\n");
         Hal_Vic_GpioOutput(GPIO_IDX_04,GPIO_LEVEL_LOW);
       }
        osDelay(2000);
    }
}
```

5. Lastly, by modifying AP name and password STA will be connected to in "wpa2_station_app.h".

```
#define WIFI_SSID           "Opulinks-TEST-AP"
#define WIFI_PASSWORD       "1234abcd"
```

## 7.5.  Testing of Execution Result

Compile the project and use download tool to download the M3 bin to the DEVKIT development board. Once the development board is reset, the OPL1000 connects to the AP first, before closing AP. Use the serial-port debugging tool to see the following log:

```
OPL1000 connected, set GPIO4 high
Deauth get 2
controller_wifi_cmd_handler!!!
Assoc Reject: startup Fast Connect progress 0
MLME_CMD_FAST_CONNECTAuto Connect !!
[AC]ssid=Opulinks-TEST-AP
WIFI JOIN REQ ! 0
controller_wifi_cmd_handler!!!
OPL1000 connected, set GPIO4 high
AUTH timeout 0
Receive AUTO CONNECT FAILED IND
Wi-Fi Disconnected
State: INACTIVE -> SCANNING
controller_wifi_cmd_handler!!!
WIFI SCAN REQ BY CFG!
wifi disconnected
OPL1000 disconnected, set GPIO4 low
Get Scan Result
```

At this stage, the development of a simple wifi application procedure based on WIFI demonstration project is completed.

OPL1000-SDK-development-guide-R01, Version V10

30

# CONTACT

sales@Opulinks.com