

BL1820

BLE SDK(Data Xfer) Quick Start Guide

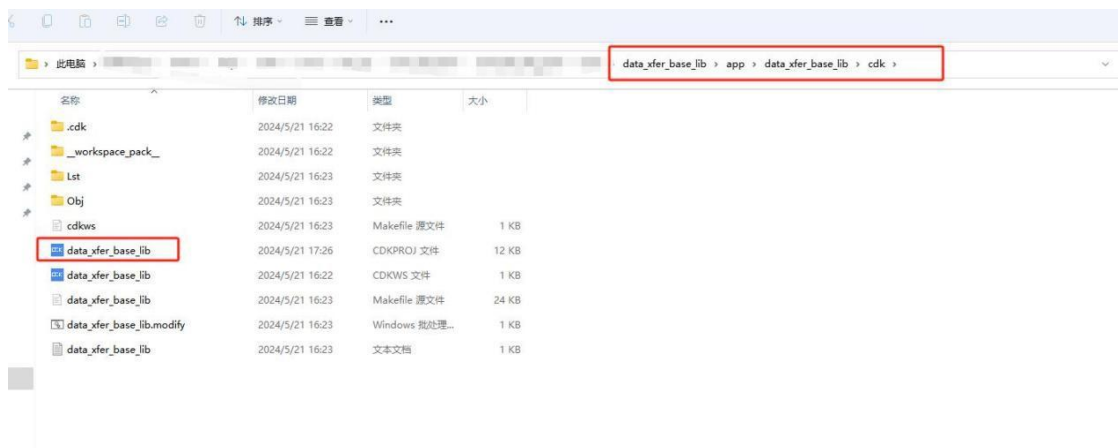
目录

1 概述	3
2 目录介绍	5
2.1 app.....	5
2.2 modules	5
2.3 睡眠唤醒功能.....	6
2.4 HCI_RAM 功能.....	7
2.5 OTP 版本编译	7

1 概述

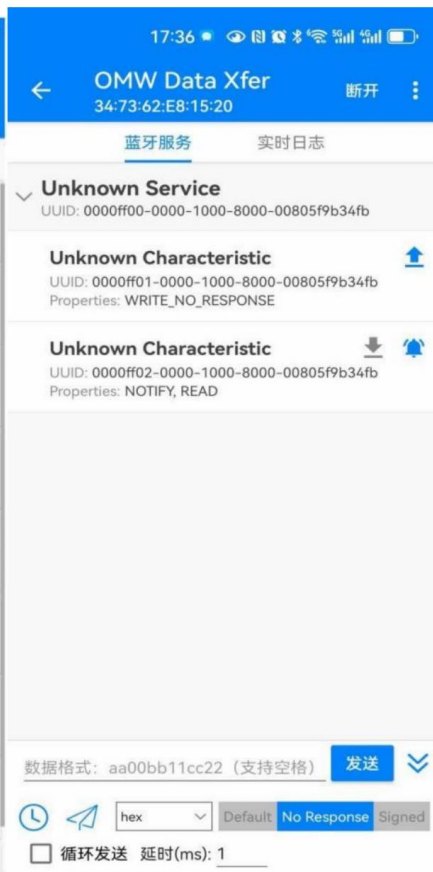
SDK 基于 RiscV MCU 平台开发。建议使用平头哥最新 CDK IDE 进行调试，代码编辑可用 vscode 等其他工具。调试器使用 CK-LINK。

打开如下图的工程文件即可在 CDK 工具（需要先安装）里打开工程，进行开发调试。 用户可以直接基于 demo 工程进行修改实现自己的功能。

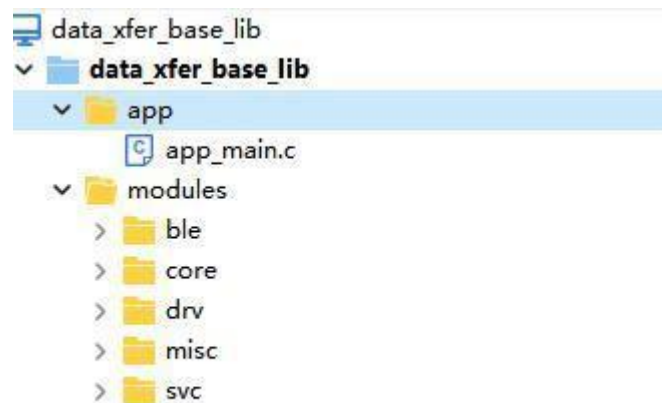


用户应用接口极其简单，只需要实现 app 目录下 app_main.c 文件里的相关函数即可完成 BLE 数传功能。代码里有相关的注释说明，实际实现时可参考 SDK 里的 demo 工程（data_xfer_base_lib），demo 程序实现了简单的数传，系统运行起来后，接上串口工具，可看到定期的 log 输出。通过“BLE 调试宝”测试工具，可以搜索到 demo 的 BLE 设备，可进行连接，连接上之后，通过工具发送数据给 BLE 设备，demo 程序里会直接将收到数据环回，在串口调试工具里能看到数据收发的 log。





2 目录介绍



2.1 app

用户实现的代码所在目录，app_main.c 主要包含以下几个可能需要用户实现的函数：

omw_app_connected_cb, BLE 协议栈在连接建立时会调用该回调通知用户。

omw_app_disconnected_cb, BLE 协议栈在连接断开时会调用该回调通知用户。

omw_app_rx_process_cb, BLE 协议栈在收到数据时会调用该回调将数据传递给应用层。

omw_app_read_usr_data_cb, master 主动读取数据，BLE 协议栈会调用该回调将数据发送给 master。

omw_app_init, 应用层初始化函数，该函数会被系统调用一次。 omw_app_can_sleep, 系统在会调用该函数判断应用层是否允许系统进入休眠。 omw_app_main, 该函数会被系统大循环调用，每次循环都会调用一次。该函数不能长时间阻塞或

死循环，否则会影响 BLE 协议栈的工作。

omw_config.h 文件，主要定义了一些系统及 BLE 用到的宏，包括 BLE 协议内部版本（目前只支持 v0 版本）、是否打开 UART log、FLASH、HCI 方式、BT 地址等，可以根据实际需要进行配置。

2.2 modules

ble: BLE 协议栈相关的接口、驱动、lib 等代码。

部分 API 说明，更多 API 及详情参考 DEMO 代码(app_main.c, omw_svc.h, omw_svc.c 等)及注释：

omw_svc_set_adv_intvl: 设置广播间隔。 omw_svc_set_addr:

设置设备地址。

omw_set_adv_data: 设置广播数据，数据内容需要由用户自行构造，需要符合广播数据格式。

omw_set_scan_rsp_data: 设置 scan respond 数据，数据内容需要由用户自行构造，需要符合 scan rsp 数据格式要求。

omw_svc_start_adv: 开始广播。 omw_svc_stop_adv: 停止广播。

omw_svc_updata_cnnt_param: 更新连接间隔，返回 0 表示参数合法并发送给 master 了，其他值表

示参数不合法。在参数合法的情况下，是否能更新成功依赖与 master 协商的结果。

core: CPU 及系统设备相关的实现。 drv: 外设驱动，
如 GPIO、FLASH、UART 等。 misc: 杂项，实现如
printf 用于 log 打印的 API。 svc: 数传以及通用服务相
关 API 实现。

2.3 睡眠唤醒功能

app 层需要实现 omw_app_can_sleep 函数，底层通过调用该函数判断是否可以进入 BLE 低功耗状态。如果可以进入 BLE 低功耗，由于底层根据 BLE 的业务及相关参数配置自动决定睡眠与唤醒的时机 与行为，在该状态下，外设（如 UART）可能是不可用的，若有上位需要通信，需要通过 GPIO（可 复用外设通信 PIN，如 UART 的 RX PIN），用户需要实现 t1001_cfg_wakeup_gpio 函数，配置相关 的唤醒 GPIO。当复用 UART 的 RX PIN 时，在唤醒前，通过上位 UART 的发送的数据会丢失掉， 所以用户需要自定义实现一套 UART 数据传输协议，用于正确的数据收发。只有唤醒之后，外设才 处于可用的状态。

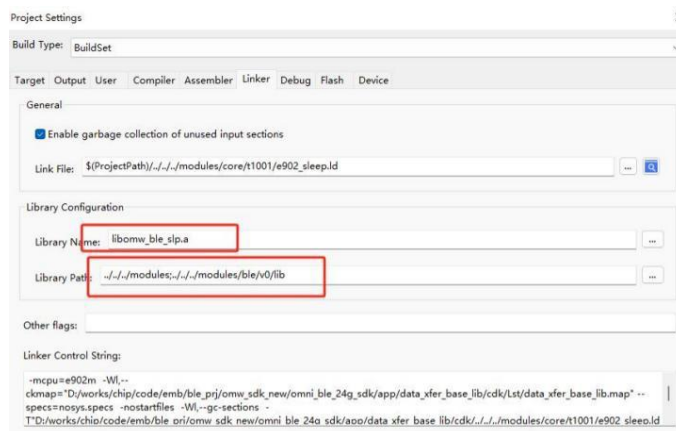
当不可进入 BLE 低功耗时，底层将处于运行或 CPU WFI 状态，在此状态下，外设（如 UART）等 都是活的，可以直接通信。

用户可以调用 omw_sleep_set_force_sleep 强制让整个系统进入睡眠状态，在此状态下底 BLE 将不工作，无法保持连接，也无法定期广播。而且只能通过 GPIO 唤醒，唤醒后，系统将按正常的 boot 过程进行重启并重新进行初始化。

要使用睡眠唤醒功能，需要全能 omw_config.h 文件中的 OMW_EN_DEEP_SLEEP 宏，当有唤醒 GPIO 时，需要这设置 OMW_GPIO_WAKEUP_MASK 或 OMW_GPIO_nWAKEUP_MASK 宏，一个是调电平唤醒，另外一个为低电平唤醒。同时需要实现 t1001_cfg_wakeup_gpio 函数。该文件里还有其他相关的一些宏定义，如 FLASH 相关的 PIN 配置，不使用的 GPIO 配置，以及睡眠后不使用的 GPIO 配置等。不使用的 GPIO 会在系统初始化时就自动被配置成不使用，以节省功耗。睡眠后不使用 GPIO 会在睡眠 会配置成不使用，以节省功耗。

用户需要根据实际情况（如不同封装，不同 PIN 配置等）实现 t1001_sleep.c 文件里如下的一些函数：void qspi_regs_restore()，当有 FLASH 时，需要实现该函数以配置 FLASH 的相关的 PIN，包括电源 PIN。void gpio_regs_restore_before_rel_gpio_hold()，当有需要在睡眠配过来后，并且在释放 GPIO HOLD 之前恢复的 GPIO 配置信息，在这个函数里实现。

开启 OMW_EN_DEEP_SLEEP 宏时， 需要使用 libomw_ble_slp.a 库， 而不开启时， 需要使用 libomw_ble.a， 二个库不能同时使用。



2.4 HCI_RAM 功能

用户可以实现自己的 BLE HOST 协议栈，通过 HCI 与 Controller 进行通信。该 HCI 是 RAM 接口，非 H4 接口，HOST 层及以上完全由用户控制。

HOST 协议栈需要实现如下几个接口函数，参考 app 目录下的 app_main.c 以及 host_dummy.c 文件：
void omw_app_init()，该函数里需要调用 omw_ctrlr_hci_ram_reg_host_rx_cb 注册 HOST 侧的 HCI 包接收接口，注册接口只能在这个函数里调用，不能在下面的 omw_host_init 里调用。

void omw_host_init()，该函数实现 host 其他必要的初始化，无则空函数。

void omw_host_work_polling()，该函数会在系统的 while(1) 大循环被调用，host 的业务处理在该函数里实现，该函数必须调用 hci_driver_polling_work() 函数，若用户在 omw_host_work_polling 里写了自己的 while(1) 大循环，则 hci_driver_polling_work() 必须每次循环都被调到一次。如用户使能了 DEEP SLEEP 功能，则 omw_host_work_polling 函数里不能有 while(1) 这样无法退出的循环，只能依赖系统的 while(1) 大循环。

当 HOST 有 HCI 包需要发送给 Controller 时，调用 omw_ctrlr_hci_ram_send2ctrl 函数进行发送。

当 Controller 有 HCI 包需要发给 HOST 时，会通过注册的 HOST 接收接口将 HCI 包发送给 HOST。

2.5 OTP 版本编译

要运行 OTP 空间上的程序的前提：芯片非空片（otp 上 bootloader 的配置区域有配置值）；若有 FLASH 则 FLASH 空间的开始第一个 WORD 值必须为 0 或 0xFFFFFFFF，或者 otp 上 bootloader 的配置区域里的 FLASH 配置区域不配置值（默认全是 0xFF）。

可用于程序的 OTP 空间有：16KB-512 编译 OTP 版本时，将 e902.ld 文件里的 ROM_BASE 改成 0x1F800000：

```
1  __ROM_BASE = 0x1F800000;
2  __ROM_SIZE = 0x00040000;
```

为了尽可能省空间，编译 OTP 版本时，将 omw_config.h 里的 CONFIG_FLASH_PROGRAM 宏注释掉，同时需要使用 non_flash 的 lib 库：

```
2
3  #ifdef OMW_EN_DEEP_SLEEP
4  #define CONFIG_OTP_PROGRAM
5  // #define CONFIG_FLASH_PROGRAM
6
```

名称	修改日期	类型	大小
bt_init	2024/8/20 14:50	C Source file	6 KB
libomw_ble.a	2024/8/31 15:18	A 文件	38 KB
libomw_ble_flash.a	2024/8/31 15:17	A 文件	38 KB
libomw_ble_non_flash.a	2024/8/31 15:14	A 文件	38 KB

当在 FLASH 下调试时，需要放开 CONFIG_FLASH_PROGRAM 宏，并且使用 flash 版本的 lib 库，同时还需要将 ld 文件里 ROM_BASE 改成 flash 的开始地址。为了省代码空间，中断向量表支持非静态的定义方式，在该方式，用户如果有新增加中断，需要参考源码里的方式添加相应的中断处理向量：


```

50
51
52 #ifndef OMW_USE_STATIC_IRQ_TBL
53 #define MAX_IRQ_ID RTC_IRQn
54
55 VECTOR_TABLE_Type __VECTOR_TABLE[MAX_IRQ_ID + 1] __attribute__((aligned(64)));
56
57 #define ADD_IRQ_HANDLE(irq_id, handle) __VECTOR_TABLE[irq_id] = handle
58 static void irq_handle_init()
59 {
60     ADD_IRQ_HANDLE(RADIO_DMA_IRQn, RADIO_DMA_Handler);
61     ADD_IRQ_HANDLE(RADIO_TICK_IRQn, RADIO_TICK_Handler);
62     ADD_IRQ_HANDLE(UART1_IRQn, UART1_Handler);
63     ADD_IRQ_HANDLE(RTC_IRQn, RTC_Handler);
64 }
65 #else
66 //MAX 48
67 VECTOR_TABLE_Type __VECTOR_TABLE[] __attribute__((aligned(64))) =
68 {

```

如果要使用静态的定义方式，要将 omw_config.h 里的 OMW_USE_STATIC_IRQ_TBL 宏放开：

```

20 // #define OMW_COMID_PATCH
21 // #define OMW_SVC_ERR_CHK
22 #define OMW_DATA_BUF_SIZE 0xA00 //KB
23 #define OMW_SVC_XFR_HAS_ADV_SCAN_DATA
24
25 // #define OMW_USE_STATIC_IRQ_TBL
26 // #define OMW_EN_FORCE_SLEEP
27
28 // #define OMW_EN_WDG

```