

POL632N-SDK

介绍说明

版本

| 版本 | 日期 | 说明 |
|--------|------------|--|
| V1.0 | 2023-05-25 | 1、初步拟稿，整理底层代码 2、支持 BLE 主/从机功能，分时使用； 3、添加三种 BLE 应用模式示例：BLE 从机数据传输应用模式，BLE 主机数据传输应用模式，基于非连接 BLE 广播数据收发应用模式 |
| V2.0 | 2023-11-01 | 1、调整 BLE 从机功能应用流程处理 2、BLE 从机支持自定义修改 gatt_services 服务 |
| V2.1 | 2024-03-29 | 1、优化 sdk 驱动文件代码，添加应用代码示例 |
| V2.2 | 2024-05-16 | 1、添加电池充电参数配置与获取函数 |
| V2.3 | 2024-05-20 | 1、添加 RTC 时钟获取与设置 |
| V2.3.1 | 2024-05-20 | 1、修复内置充电识别不到的问题 |
| V2.3.2 | 2024-06-15 | 1、修复 RTC 获取函数返回值异常问题 |
| V2.4.0 | 2025-03-11 | 1、修复 AC6328 芯片在休眠是功耗高的问题 2、修复应用层修改 BLE 地址后导致 OTA 升级时回连失败的问题 |
| V2.4.1 | 2025-04-07 | 1、开启 ota 双备份升级 2、更新原厂发布的蓝牙兼容性补丁 |
| V2.4.2 | 2025-04-18 | 1、添加 IR 功能 2、添加脉冲检测 demo |

目录

| | |
|---------------------------------------|----|
| 1. SDK 开发环境的说明和工具使用 | 4 |
| 2. SDK 主体程序架构 | 7 |
| 3. Timer 定时器的使用 | 9 |
| 4. PWM 接口使用 | 10 |
| 5. ADC 接口的使用 | 11 |
| 6. Uart 串口接口的使用 | 12 |
| 7. VM 存储模块的使用 | 13 |
| 8. 低功耗唤醒的使用 | 14 |
| 9. BLE 从机数据传输应用 | 15 |
| 10. BLE 主机数据传输应用 | 17 |
| 11. 基于非连接 BLE 广播数据收发应用 | 19 |
| 12. 自定义函数程序 XXX.c 和 XXX.h 的添加说明 | 20 |
| 13. 底层串口打印辅助开发 | 21 |
| 14. 电池充电参数配置与获取 | 22 |
| 15. RTC 时钟获取与设置 | 23 |

1. SDK 开发环境的说明和工具使用

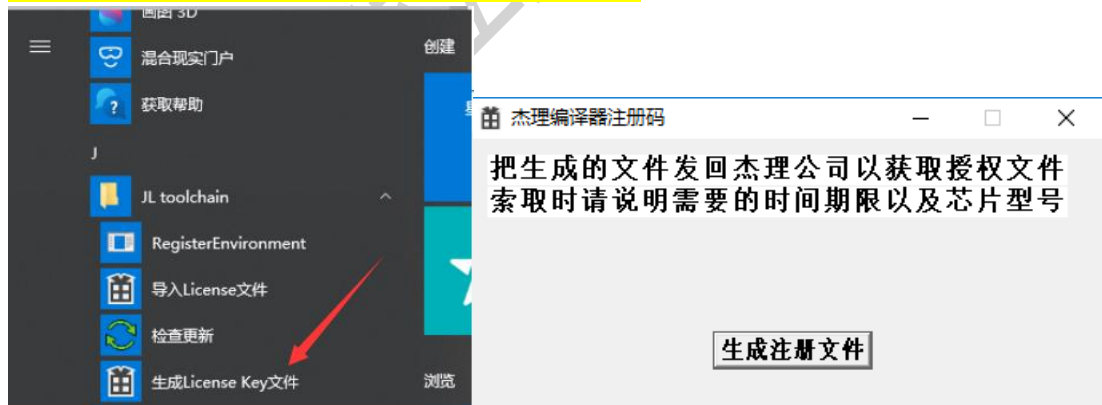
1.1 关闭杀毒软件和安全卫士

1.2 先安装 codeblocks-16.01mingw-setup.exe，默认配置，安装完成后，先运行一次 codeblocks，然后关闭。（先运行一次是为了自动生成配置文件）

1.3 然后接着安装 jl_toolchain_pi32v2_lto_2.1.8.exe



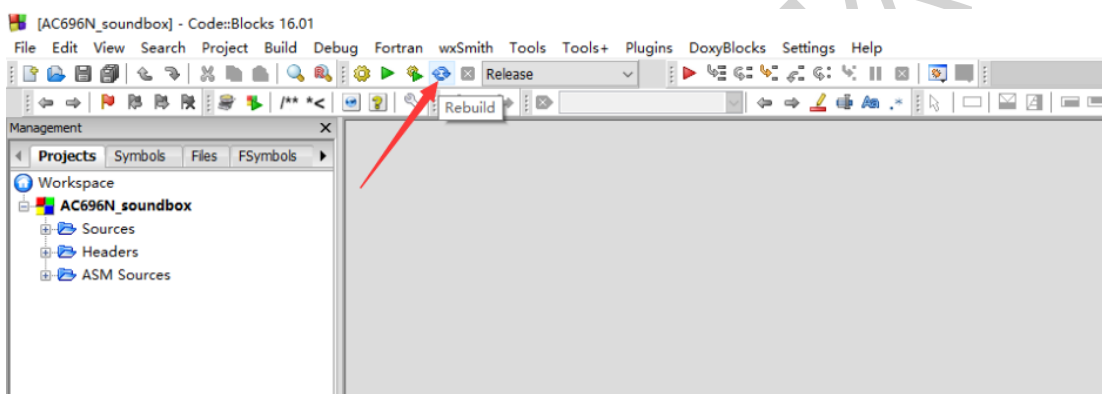
在开始 -> 程序 -> JL toolchain 中打开”生成 License Key 文件”，命好名字，保存为 XXX.key 的文件，例如我们保存为 123.key，并把 123.key 文件发给我们，我们申请生成对应的 123.lib 发给你们导入编译器中，编译器才可以通过 codeblocks 编译对应的程序。注：电脑生成的 XXX.key 和后面申请对应的 XXX.lib 是一一对应的，每次更新电脑系统或者更改电脑安装新的编译器都需要重新生成 XXX.key 和申请新的对应 XXX.lib。



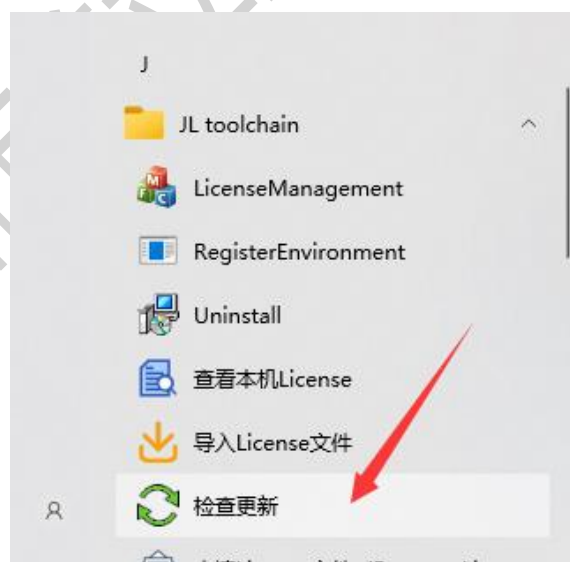
收到 XXX.lib 后，导入方式为，开始 -> 程序 -> JL toolchain 中打开”导入 License 文件”



打开 jl_toolchain_update_2.4.6.exe，更新成最新的编译环境。
至此，编译环境安装完成，可以打开 CodeBlocks 并打开对应程序，每次更新程序都要 Rebuild 一下代码。



后续，如果编译器版本需要更新，只需点击开始 -> 程序 -> JL toolchain 中打开“检测更新”，下载下来更新就可以了。



1.4 工程 SDK 打开路径在 polyc632n_mcu_bt_sdk\中的
polyc632n_mcu_bt_sdk.cbp。代码整体编译后，生成烧录文件为路径
polyc632n_mcu_bt_sdk\cpu\bd19\tools\download\data_trans 中的
pol632n_lto8_file.fw。生成的升级文件路径为 polyc632n_mcu_bt_sdk\cpu\

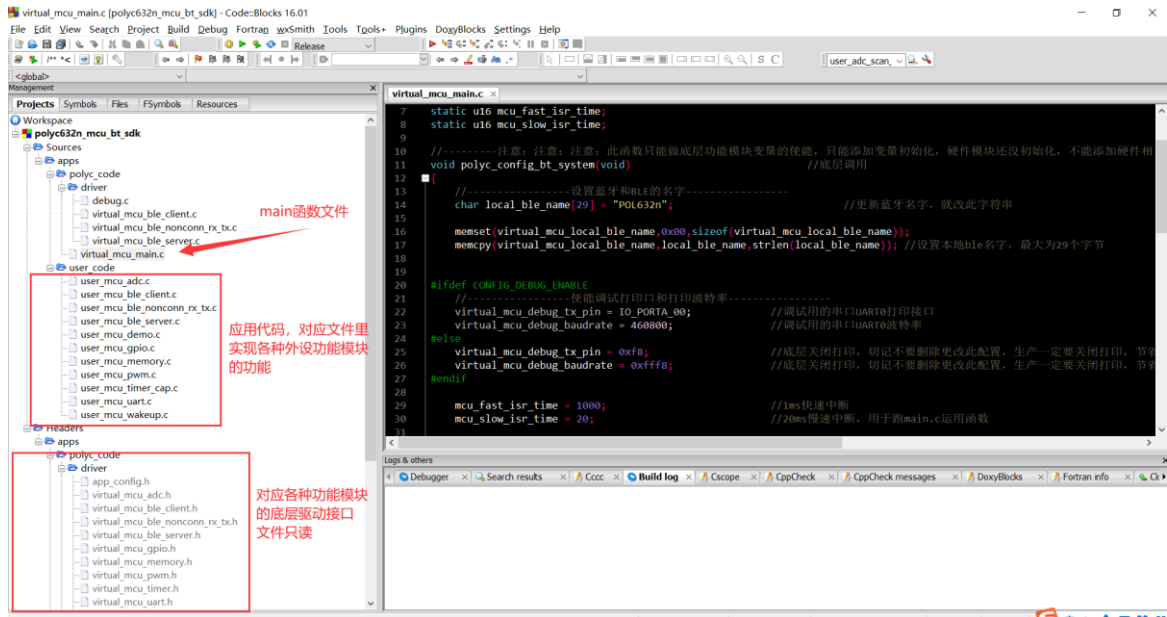
版权所有 2023 深圳市普立晶电子有限公司，未经许可，禁止转载

bd19\tools\download\data_trans 中的 updata. ufw。

| polyc632n_mcu_bt_sdk | | | | |
|-----------------------------|------------------|--------------|-------|--|
| 名称 | 修改日期 | 类型 | 大小 | |
| apps | 2023/5/19 16:56 | 文件夹 | | |
| cpu | 2023/5/13 14:18 | 文件夹 | | |
| include_lib | 2023/5/13 14:18 | 文件夹 | | |
| obj | 2023/5/13 15:11 | 文件夹 | | |
| tools | 2023/5/13 14:18 | 文件夹 | | |
| default.workspace | 2022/11/23 16:08 | WORKSPACE 文件 | 2 KB | |
| Makefile | 2022/11/23 16:08 | 文件 | 5 KB | |
| polyc632n_mcu_bt_sdk.cbp | 2023/5/24 14:55 | project file | 11 KB | |
| polyc632n_mcu_bt_sdk.depend | 2023/5/24 18:02 | DEPEND 文件 | 24 KB | |
| polyc632n_mcu_bt_sdk.layout | 2023/5/24 17:22 | LAYOUT 文件 | 6 KB | |

| polyc632n_mcu_bt_sdk > cpu > bd19 > tools > download > data_trans | | | | |
|---|------------------|----------------|----------|--|
| 名称 | 修改日期 | 类型 | 大小 | |
| app.bin | 2023/5/24 18:02 | BIN 文件 | 131 KB | |
| bd19loader.bin | 2022/11/23 16:08 | BIN 文件 | 23 KB | |
| burnfile_1to8.bat | 2023/5/24 18:02 | Windows 批处理... | 1 KB | |
| burnfile_1to8_v213.exe | 2022/5/10 16:44 | 应用程序 | 5,571 KB | |
| cfg_tool.bin | 2023/5/13 15:19 | BIN 文件 | 1 KB | |
| code.cfg | 2022/12/13 9:40 | CFG 文件 | 1 KB | |
| download.bat | 2023/5/22 16:42 | Windows 批处理... | 2 KB | |
| jl_isd.bin | 2023/5/24 18:10 | BIN 文件 | 144 KB | |
| jl_isd.fw | 2023/5/24 18:10 | FW 文件 | 362 KB | |
| jl_isd.key | 2022/12/13 9:39 | KEY 文件 | 1 KB | |
| p11_code.bin | 2022/11/23 16:08 | BIN 文件 | 4 KB | |
| pol632n_1to8_file.fw | 2023/5/24 18:10 | FW 文件 | 362 KB | |
| script.ver | 2023/5/13 15:19 | VER 文件 | 1 KB | |
| tone.cfg | 2022/11/23 16:08 | CFG 文件 | 5 KB | |
| update.ufw | 2023/5/24 18:10 | UFW 文件 | 503 KB | |

1.5 打开 polyc632n_mcu_bt_sdk. cbp，编写整改代码和全编译如下图



2.2 整个代码 SDK 最小系统在 virtual_mcu_main.c 中，主要包括四部分

①底层某些功能使能和变量初始化，函数为

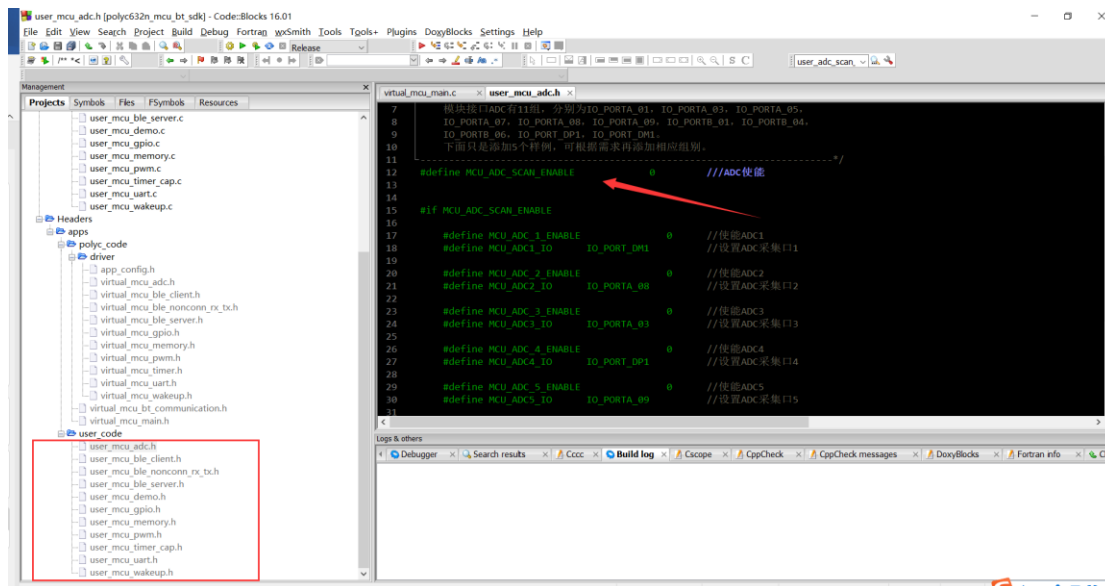
`void polyc_config_bt_system(void);`

②快速中断定时器 timer 应用，在 `void fast_period_timer_callback(void)` 中，对需要运行的快速计数中断变量进行计数，一般定义 1ms 触发一次，也可根据需求做对应更改。

③慢速中断定时器 timer 应用，在 `void slow_period_timer_callback(void)` 中，主要应用主函数 main 需要跑的功能，对相应接口函数进行处理，一般定义 20ms 处理一次，也可根据需求做时间的更改。

④应用层驱动的初始化，在 `void polyc_init(void)` 中，如 timer 定时器，PWM，ADC，uart 和 VM 存储。

2.3 本 SDK 已经做好各个驱动测试使能代码，可以通过 `user_mcu_xxx.h` 打开对应的测试模块，客户后期开发也可以参考该模块调用编写自己的代码。



3. Timer 定时器的使用

3.1 快速定时器中断的使用

①初始化:

```
polyc_fast_period_timer_init(1000);
```

//初始化参数 1000 表示 1000us=1ms, 可根据需求初始化的时候更改这个时钟参数, 最低不能低于 200us, 因为底层内部还有许多中断需要调用, 此中断太快会影响其他功能应用。

```
polyc_fast_period_timer_register_callback(fast_period_timer_callback); //注册中断中调用的应用层函数
```

②中断调用:

根据初始化注册的中断调用函数, 每 1ms 将触发一次中断执行相对应代码

```
void fast_period_timer_callback(void) //1ms
{
}
}
```

3.2 慢速定时器或者主函数 main 执行程序的使用

①初始化:

```
polyc_lib_timer_init(20); //
//初始化参数 20 表示 20ms 定时中断, 最低 1ms, 可根据需求更改主函数 main 循环时间
```

```
polyc_lib_timer_register_callback(slow_period_timer_callback); //
//注册中断中调用的应用层函数
```

②中断调用:

此函数主要是 main 函数功能的执行调用, 因底层是中断注册, 已经严格按时间规定执行, 所以应用层在这个函数中不要添加过长的延时 delay 时间或者通

过变量计数完成超过 20ms 的延时。

```
void slow_period_timer_callback(void)    ///20ms
{
    main();
}
```

3.3 中断定时器 timer0 或者中断捕捉应用（timer0 中断捕捉口为 PA6）

①初始化：（使能中断下降沿捕捉功能）

```
polyc_timer0_init(TIMER_FALLING_EDGE, 1000);
///下降沿捕捉
polyc_timer0_register_callback(mcu_timer_cap_callback);
///注册中断中调用的应用层函数
```

②中断调用：

根据初始化注册的中断调用函数，下降沿时触发

```
void mcu_timer_cap_callback(void)
```

4. PWM 接口使用

```
#define MCU_PWM_CHANNEL0    0
#define MCU_PWM_CHANNEL1    1
#define MCU_PWM_CHANNEL2    2
#define MCU_PWM_CHANNEL3    3
```

POL632N 最大支持 4 路硬件 PWM，通过 void polyc_pwm_init(u8 ch, u32 fre, u32 duty, u32 port); 初始化设置对应的通道，频率，占空比和输出 PWM 的对应 IO 口。

参数说明：

- * @param ch : PWM 通道 0-3
- * @param fre : 频率，单位 Hz，不小于 400
- * @param duty : 初始占空比，0~10000 对应 0~100%
- * @param port : pwm 脚，除了 USB 口外，可选其他任意 GPIO。

初始化完成后，后续调用，只需在对应的通道上设置占空比就可以了，函数块为 void polyc_set_pwm_duty(u8 ch, u32 duty);

参数说明：

- * @param ch : PWM 通道 0-3
- * @param duty : 占空比，0~10000 对应 0~100%

例如：

①初始化 PWM：

```
polyc_pwm_init(MCU_PWM_CHANNEL0, 20000, 0, IO_PORTA_07);    //初始化 PWM 通道 0，频率为 20K，0%占空比，通过 PA7 输出。
```

②设置占空比：

```
polyc_set_pwm_duty(MCU_PWM_CHANNEL0, 1000);                //设置 PWM 通道 0 的占空比为 10%高电平
```

5. ADC 接口的使用

模块接口 ADC(10 位, 0-1023) 有 11 组, 分别为 IO_PORTA_01, IO_PORTA_03, IO_PORTA_05, IO_PORTA_07, IO_PORTA_08, IO_PORTA_09, IO_PORTB_01, IO_PORTB_04, IO_PORTB_06, IO_PORT_DP1, IO_PORT_DM1。

通过函数 void polyc_adc_set_init(u32 ch); 初始化设置对应的 ADC 通道, 内部 5ms 扫描转化 ADC 通道, 再通过 u16 polyc_get_adc_value(u32 ch); 对应通道获取有效的 ADC 值。

例如:

①初始化 ADC 接口:

```
void mcu_adc_init(void)
{
    polyc_adc_set_init(IO_PORTA_05);    //设置 ADC 采样接口
}
```

②中断扫描获取 ADC 值:

```
void mcu_adc_scan_deal(void)    //20ms 扫描一次 ADC
{
    static u16 ad_value1;
    static u8 cnt;

    cnt++;
    if(cnt >= 20)
    {
        cnt = 0;
        ad_value1 = polyc_get_adc_value(IO_PORTA_05);    //ADC 采集
        的值(10 位 ADC 采样 0-1023)
        mcu_adc_deal_1(ad_value1);
    }
}
```

③根据 ADC 值做相应函数处理:

```
void mcu_adc_deal_1(u16 value)    //10 位 ADC 0-1023
{
    ///ADC 数据处理函数
    ///printf("adc value1 = %d\n", value);
}
```

6. Uart 串口接口的使用

模块 UART 应用层支持三组串口，UART0 为底层打印驱动，应用层不能使用，剩下 UART1 和 UART2，分别对应 4 组和 3 组接口可选择，分别为

UART1--GROUP1 对应接口 632N_UART1_GROUP_TXPB04_RXPB05

UART1--GROUP2 对应接口 632N_UART1_GROUP_TXPB00_RXPB01

UART1--GROUP3 对应接口 632N_UART1_GROUP_TXPA07_RXPA08

UART1--GROUP4 对应接口 632N_UART1_GROUP_USB_TXDP_RXDM

UART2--GROUP1 对应接口 632N_UART2_GROUP_TXPA03_RXPA04

UART2--GROUP3 对应接口 632N_UART2_GROUP_TXPB06_RXPB07

UART2--GROUP4 对应接口 632N_UART2_GROUP_TXPDP1_RXPDM1

波特率支持：BAUD_2400, BAUD_4800, BAUD_9600, BAUD_19200, BAUD_38400, BAUD_57600, BAUD_115200, BAUD_230400, BAUD_460800。

例如：

①初始化：

```
polyc_set_BT_System_uart_open(UART2, GROUP1, BAUD_9600);
```

//选择 UART2 的 GROUP1 的 PA3 和 PA4 做串口通讯，采样率为 9600Hz

②RX 接收回调函数处理：

```
Void      polyc_uart2_recieved_data_func_callback(u8      *buff, u16
buff_size)
{
    //等待 RX 数据的带来，接收到后数据处理
    printf_buf(buff, buff_size);
}
```

③TX 发送数组：

```
polyc_uart_send_data(UART2, uart_tx_data_buff, 16, 0);
```

//通过 UART2 的 GROUP1 发送 16 个 byte 数据。

7. VM 存储模块的使用

7.1 模块的存储 ID 分为 0-29，总共的 30 个，每片存储 ID 可存储 64 个字节，即最大的存储量个 $30 \times 64 = 1920$ 个字节。

```
//-----  
//  
//                      MCU 内部存储,掉电不丢失  
//-----  
#define VM_MCU_MEM_ITEM_ID_MIN                (0)  
#define VM_MCU_MEM_ITEM_ID_MAX                (30)  
#define VM_MCU_MEM_ITEM_DATA_LEN_MAX          (64)  
#define VM_ERR_MEM_ITEM_ID_OUT_OF_RANGE        (0xFFFF)  
#define VM_ERR_MEM_ITEM_ID_DATA_LEN_OUT_OF_RANGE (0xFFFE)
```

7.2 模块存储 VM 通过下面两个函数进行读写。

```
int polyc_write_memory(u8 item_id,u8 *buff,u16 buff_size);  
//此写函数带返回值，写入存储正确反馈写入 buff_size 长度，写入错误，  
反馈对应的错误码，具体查看源代码说明  
int polyc_read_memory(u8 item_id,u8 *buff,u16 buff_size);  
//此读函数带返回值，如果此 ID 有记忆过反馈读取数组有效 buff_size 长  
度，读取数据失败或错误，反馈对应的错误码，具体查看源代码说明
```

8. 低功耗唤醒的使用

8.1 函数说明

```
#define WAKEUP_RISING_EDGE      0
#define WAKEUP_FALLING_EDGE    1
#define WAKEUP_BOTH_EDGE       2
/**-----
    函数名: polyc_set_wakeup_io
    参数  : u8 pullup_down_enable, u8 edge, u8 gpio
    说明  : 初始化唤醒 IO 口
    * @param pullup_down_enable : 内部上拉或者内部下拉使能, 例如下降
    沿触发, 就使能内部上拉, 上升沿触发, 就使能内部下拉
    * @param edge               : 设置上升沿触发 WAKEUP_RISING_EDGE,
    下降沿触发 WAKEUP_FALLING_EDGE, 或者上升和下降沿都可以触发
    WAKEUP_BOTH_EDGE
    * @param gpio               : 设置唤醒 IO 口, 如 IO_PORTB_01
    返回值: void
    -----*/
void polyc_set_wakeup_io(u8 pullup_down_enable, u8 edge, u8 gpio);
/**-----
    函数名: polyc_power_off_io_keep
    参数  : u8 gpio
    说明  : 进入睡眠之前设置维持的 IO 口
    返回值: void
    -----*/
void polyc_power_off_io_keep(u8 gpio);
/**-----
    函数名: polyc_power_off_to_sleep
    参数  :
    说明  : 关机进入低功耗模式, 芯片本身只剩下 3uA
    返回值: void
    -----*/
void polyc_power_off_to_sleep(void);
```

8.2 示例如下

```
polyc_set_wakeup_io(1, FALLING_EDGE, IO_PORTB_01);
//设置 PB1 口为输入检测口, 下降沿唤醒, 内部上拉使能

polyc_power_off_io_keep(IO_PORTB_01); //保持原有上拉输入状态
polyc_power_off_to_sleep(); //关机进入低功耗, 等待 PB1 下降沿唤醒
```

9. BLE 从机数据传输应用

9.1 BLE 从机广播周期和广播包配置

广播周期：修改宏定义 `MCU_BLE_SERVER_ADV_INTERVAL_MIN` 的值，该值最小为 20ms，配置时最好不小于 100ms。

广播包数据：没有特殊要求一般不需要修改，只有在特定的广播包需求下才进行修改。了解过蓝牙 BLE 知识的开发人员，adv 包数据修改可参考 `ble_server_make_set_adv_data()` 函数，rsp 包数据修改可参考 `ble_server_make_set_rsp_data()` 函数。

9.2 BLE 从机自定义 UUID 服务说明

熟悉了解 BLE 知识的开发人员，可以在 `virtual_mcu_ble_server.c` 文件里修改数组 `mcu_ble_profile_data[]`，修改为自定义的 UUID 服务。

自定义 UUID 范围从 0x8000-0xFFFF；其中 BLE_SERVER 从机 0xAE00-0xAE02 用于 OTA 升级，不可使用

SDK 默认的自定义 UUID 服务为：

服务 UUID (service UUID)：0xFFFF0（默认）可通过软件更改

特征值 UUID (characteristics UUID)：0xFFFF1（默认）可通过软件更改
属性：Notify

【该通道数据方向为：BLE 从机角色 -> BLE 主机角色（如手机 APP）】

特征值 UUID (characteristics UUID)：0xFFFF2（默认）可通过软件更改
属性：Write Without Response

【该通道数据方向为：BLE 主机角色（如手机 APP） -> BLE 从机角色】

若不熟悉 BLE 相关知识的开发人员，但需要修改 UUID 服务，去适配手机 APP 和微信小程序，可以联系我司的开发人员，协助修改。

9.3 BLE 从机数据传输函数的使用

在 `virtual_mcu_ble_server.c` 文件里，已经做好 BLE 从机功能的流程，`virtual_mcu_ble_server.h` 头文件里，已经封装好了可能会使用到的函数，可供外部调用。

//BLE_SERVER 从机应用模式开启和关闭 1:开启 0:关闭

```
void mcu_ble_server_enable(u8 enable);
```

//ble_server 从机端发数接口

```
int mcu_ble_server_data_send(u8 *buff, u16 buff_size);
```

//ble_server 从机端数据接收回调函数

```
static void mcu_ble_server_data_receive_callback(u16 att_handle, u16 offset, u8 *buffer, u16 buffer_size);
```

版权所有 2023 深圳市普立晶电子有限公司，未经许可，禁止转载

```
virtual_mcu_ble_server.h ×
23
24 extern u8 curr_server_peer_addr_info[7]; //当前连接对方地址信息 1位addr_type + 6位address
25
26 u8 mcu_get_ble_server_connect_flag(void); //获取ble_server从机是否已经连接
27
28
29 //ble_server从机初始化函数
30 void mcu_ble_server_init(void);
31
32 //BLE_SERVER从机应用模式开启和关闭 1:开启 0:关闭
33 void mcu_ble_server_enable(u8 enable);
34
35 //ble_server从机获取链路对方的信号强度, 范围-127 ~ 127
36 s8 mcu_get_ble_server_curr_rssi(void);
37
38 //断开ble_server从机连接
39 void mcu_ble_server_disconnected(void);
40
41 //ble_server从机端发数接口 返回0,发送成功
42 int mcu_ble_server_data_send(u8 *buff, u16 buff_size);
43
44 //注册ble_server从机端收数接口
45 void mcu_ble_server_data_receive_register_callback(void *cbk(u16 value_handle, u16 offset, u8 *buffer, u16 buffer_size));
46
```


10. BLE 主机数据传输应用

10.1 BLE 主机扫描参数和连接参数配置

BLE 主机连接上 BLE 从机后，搜索并使能该指定的 UUID 表，只有当 BLE 从机的 UUID 在 BLE 主机配置的指定 UUID 表里时，才能通过该指定的通道进行收发数据。

SDK 默认将 BLE 主机的 UUID 表和 BLE 从机的 UUID 配置为一致，使其 BLE 主机连接上 BLE 从机后，通过该配置的通道进行相互收发数据。

```
virtual_mcu_ble_client.c x
243 }
244
245
246 //设置BLE主机连接搜索指定uuid
247 static void mcu_ble_client_set_search_uuid_table(void)
248 {
249     //读取远端server设备名device_name, 不可修改
250     client_search_uuid_table[0].services_uuid16 =0x1800;
251     client_search_uuid_table[0].characteristic_uuid16 =0x2a00;
252     client_search_uuid_table[0].opt_type =POLYC_ATT_PROPERTY_READ;
253
254     //UUID16_NOTIFY 下行数据通道:ble_client主机接收远端BLE从机的数据通道
255     client_search_uuid_table[1].services_uuid16 =MCU_BLE_SERVICES_UUID16;
256     client_search_uuid_table[1].characteristic_uuid16 =MCU_BLE_CHARACTERISTIC_UUID16_NOTIFY;
257     client_search_uuid_table[1].opt_type =POLYC_ATT_PROPERTY_NOTIFY; //不可修改
258
259     //UUID16_WRITE 上行数据通道:ble_client主机发送给远端BLE从机的数据通道
260     client_search_uuid_table[2].services_uuid16 =MCU_BLE_SERVICES_UUID16;
261     client_search_uuid_table[2].characteristic_uuid16 =MCU_BLE_CHARACTERISTIC_UUID16_WRITE;
262     client_search_uuid_table[2].opt_type =POLYC_ATT_PROPERTY_WRITE_WITHOUT_RESPONSE; //不可修改
263 }
264
```

10.2 BLE 主机扫描参数和连接参数配置

一般不需要修改。SDK 开放了主机扫描参数和连接参数配置，只有在特定的需求下才进行修改调整。

```
virtual_mcu_ble_client.c x virtual_mcu_ble_client.h x
2 #define __VIRTUAL_MCU_BLE_CLIENT_H__
3
4 #include "virtual_mcu_main.h"
5
6 //搜索类型
7 #define SET_TYPE_SCAN_PASSIVE 0
8 #define SET_TYPE_SCAN_ACTIVE 1
9
10 //搜索 周期大小
11 #define MCU_BLE_CLIENT_SET_SCAN_INTERVAL MCU_ADV_SCAN_MS(50) // unit: 0.625ms
12 //搜索 窗口大小
13 #define MCU_BLE_CLIENT_SET_SCAN_WINDOW MCU_ADV_SCAN_MS(20) // unit: 0.625ms, <= MCU_BLE_CLIENT_SET_SCAN_INTERVAL
14
15 //连接周期
16 #define MCU_BLE_CLIENT_BASE_INTERVAL_MIN (6)//最小的interval
17 #define MCU_BLE_CLIENT_SET_CONN_INTERVAL (MCU_BLE_CLIENT_BASE_INTERVAL_MIN*8) //(unit:1.25ms)
18 //连接latency
19 #define MCU_BLE_CLIENT_SET_CONN_LATENCY 0 //(unit:conn_interval)
20 //连接超时
21 #define MCU_BLE_CLIENT_SET_CONN_TIMEOUT 400 //(unit:10ms)
22
```

10.3 BLE 主机解析 SCAN 扫描到的广播包数据

SDK 里已做好 BLE 主机解析协议栈回调的广播包数据，具体可以查看该 `ble_client_resolve_adv_report()` 函数的内容实现。

10.4 BLE 主机数据传输函数的使用

在 `virtual_mcu_ble_client.c` 文件里，已经做好 BLE 主机功能的流程，`virtual_mcu_ble_client.h` 头文件里，已经封装好了可能会使用到的函数，可供外部调用

```

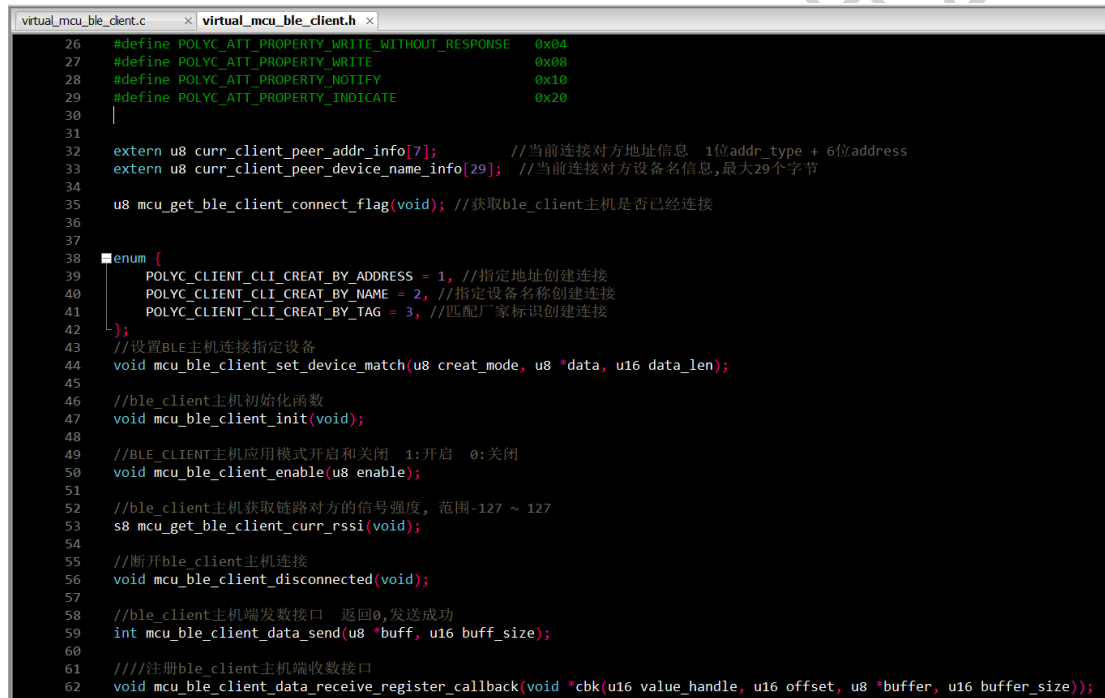
//BLE_CLIENT 主机应用模式开启和关闭 1:开启 0:关闭
void mcu_ble_client_enable(u8 enable);

//设置 BLE 主机连接指定设备
void mcu_ble_client_set_device_match(u8 creat_mode, u8 *data, u16
data_len);

//ble_client 主机端发数接口 返回 0, 发送成功
int mcu_ble_client_data_send(u8 *buff, u16 buff_size);

/*ble_client 主机接收到从机 server 端的数据回调函数
void mcu_ble_client_report_gatt_data_callback(polyc_att_data_report_t
*report_data)

```



```

virtual_mcu_ble_client.c x virtual_mcu_ble_client.h x
26 #define POLYC_ATT_PROPERTY_WRITE_WITHOUT_RESPONSE 0x04
27 #define POLYC_ATT_PROPERTY_WRITE 0x08
28 #define POLYC_ATT_PROPERTY_NOTIFY 0x10
29 #define POLYC_ATT_PROPERTY_INDICATE 0x20
30
31
32 extern u8 curr_client_peer_addr_info[7]; //当前连接对方地址信息 1位addr_type + 6位address
33 extern u8 curr_client_peer_device_name_info[29]; //当前连接对方设备名信息,最大29个字节
34
35 u8 mcu_get_ble_client_connect_flag(void); //获取ble_client主机是否已经连接
36
37
38 enum {
39     POLYC_CLIENT_CLI_CREAT_BY_ADDRESS = 1, //指定地址创建连接
40     POLYC_CLIENT_CLI_CREAT_BY_NAME = 2, //指定设备名称创建连接
41     POLYC_CLIENT_CLI_CREAT_BY_TAG = 3, //匹配厂家标识创建连接
42 };
43 //设置BLE主机连接指定设备
44 void mcu_ble_client_set_device_match(u8 creat_mode, u8 *data, u16 data_len);
45
46 //ble_client主机初始化函数
47 void mcu_ble_client_init(void);
48
49 //BLE_CLIENT主机应用模式开启和关闭 1:开启 0:关闭
50 void mcu_ble_client_enable(u8 enable);
51
52 //ble_client主机获取链路对方的信号强度, 范围-127 ~ 127
53 s8 mcu_get_ble_client_curr_rssi(void);
54
55 //断开ble_client主机连接
56 void mcu_ble_client_disconnected(void);
57
58 //ble_client主机端发数接口 返回0, 发送成功
59 int mcu_ble_client_data_send(u8 *buff, u16 buff_size);
60
61 //注册ble_client主机端收数接口
62 void mcu_ble_client_data_receive_register_callback(void *cbk(u16 value_handle, u16 offset, u8 *buffer, u16 buffer_size));
63

```

11. 基于非连接 BLE 广播数据收发应用

11.1 非连接 BLE 主机搜索周期和 BLE 从机广播周期配置

一般不要修改，该参数会影响到广播收发数的周期和时间间隔。

```
virtual_mcu_ble_nonconn_rx_tx.c  virtual_mcu_ble_nonconn_rx_tx.h
1  #ifndef VIRTUAL_MCU_BLE_NONCONN_RX_TX_H
2  #define VIRTUAL_MCU_BLE_NONCONN_RX_TX_H
3
4  #include "virtual_mcu_main.h"
5
6
7  //搜索类型
8  #define BLE_NONCONN_RX_SCAN_PASSIVE  0
9  #define BLE_NONCONN_RX_SCAN_ACTIVE   1
10
11 #define BLE_NONCONN_ADV_TYPE          3  /*Non connectable undirected advertising*/
12
13
14 //搜索 周期大小
15 #define BLE_NONCONN_RX_SCAN_INTERVAL  MCU_ADV_SCAN_MS(200) // unit: 0.625ms
16 //搜索 窗口大小
17 #define BLE_NONCONN_RX_SCAN_WINDOW    MCU_ADV_SCAN_MS(200) // unit: 0.625ms, <= BLE_NONCONN_RX_SCAN_INTERVAL
18
19
20 //广播周期
21 #define BLE_NONCONN_TX_ADV_INTERVAL_MIN MCU_ADV_SCAN_MS(20) /*毫秒 转换到 0.625ms 单位*/
22
```

11.2 非连接 BLE 广播数据收发函数的使用

在 virtual_mcu_ble_nonconn_rx_tx.c 文件里，已经做好非连接 BLE 广播数据收发的流程，virtual_mcu_ble_nonconn_rx_tx.h 头文件里，已经封装好了可能会使用到的函数，可供外部调用

//非连接 BLE 广播收发功能开启和关闭 1:开启 0:关闭

void virtual_mcu_ble_nonconn_rx_tx_enable(u8 enable);

//非连接 BLE 广播数据发送接口

u8 mcu_ble_nonconn_adv_data_tx(u8 *adv, u8 adv_size, u8 tx_interval);

//非连接 BLE 广播数据收数接口回调函数

static void mcu_ble_nonconn_rx_report_callback(polyc_adv_report_t *report_pt, u16 len)

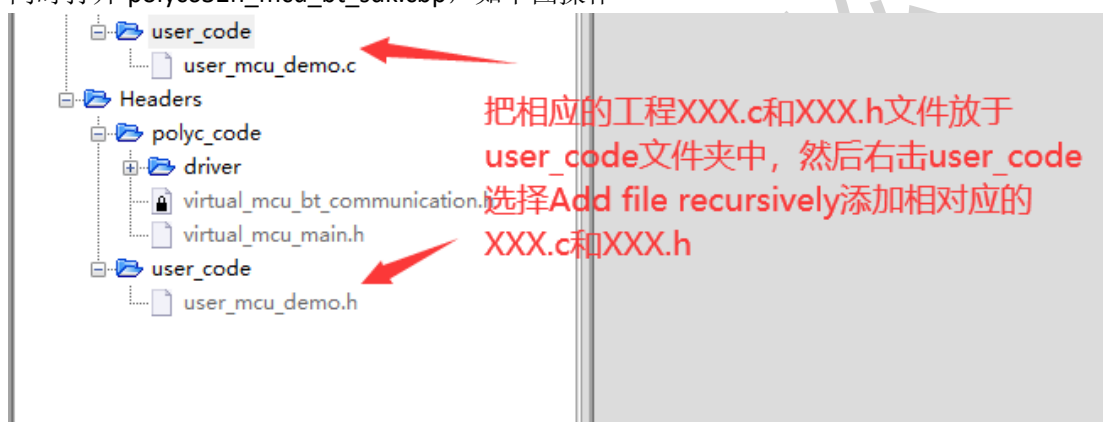
```
virtual_mcu_ble_nonconn_rx_tx.c  virtual_mcu_ble_nonconn_rx_tx.h
21 #define BLE_NONCONN_TX_ADV_INTERVAL_MIN MCU_ADV_SCAN_MS(20) /*毫秒 转换到 0.625ms 单位*/
22
23 //BLE非连接广播收发功能开启和关闭 1:开启 0:关闭
24 void virtual_mcu_ble_nonconn_rx_tx_enable(u8 enable);
25
26 /*-----*/
27 /*
28 @brief BLE非连接广播数据发送接口
29 @param *adv: 发送广播数据的指针
30 @param adv_size: 发送广播数据的长度,最大长度为31
31 @param tx_interval: 广播该数据持续的时间(单位:10ms)
32 @return 返回值: 0 表示失败
33 非0 表示成功,当前发送数据的长度
34
35 @note
36
37 */
38 /*-----*/
39 u8 mcu_ble_nonconn_adv_data_tx(u8 *adv, u8 adv_size, u8 tx_interval);
40
41 //注册BLE非连接广播数据收数接口
42 void mcu_ble_nonconn_adv_rx_register_callback(void *cbk(polyc_adv_report_t *rx_report_pt, u16 len));
```

12. 自定义函数程序 XXX.c 和 XXX.h 的添加说明

自定义添加的 XXX.c 和 XXX.h 程序，请放在如下路径
polyc632n_mcu_bt_sdk\apps\user_code，如下程序

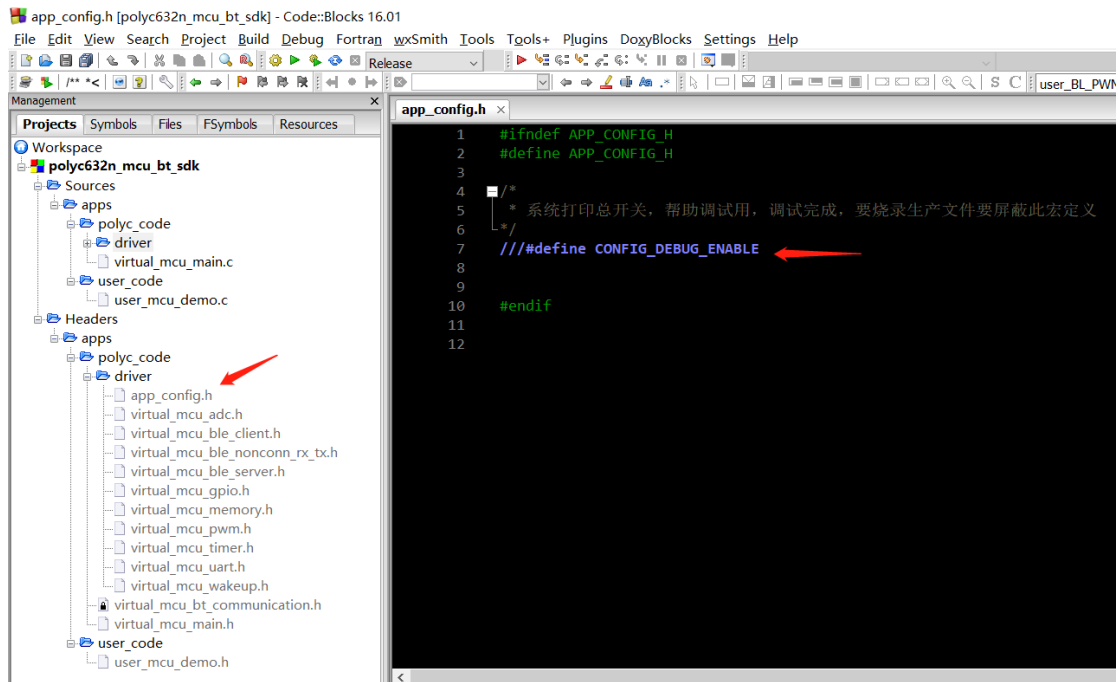
| polyc632n_mcu_bt_sdk > apps > user_code | | | |
|---|-----------------|------|-------|
| 名称 | 修改日期 | 类型 | 大小 |
| user_mcu_demo.c | 2023/5/24 17:14 | C 文件 | 11 KB |
| user_mcu_demo.h | 2023/5/24 15:46 | H 文件 | 1 KB |

同时打开 polyc632n_mcu_bt_sdk.cbp，如下图操作



13. 底层串口打印辅助开发

13.1 在 app_config.h 中打开打印开关 CONFIG_DEBUG_ENABLE，如下图



15.2 必须在 polyc_config_bt_system() 中设置相应的打印口和波特率

```
void polyc_config_bt_system(void)
{
    //-----使能调试打印口和打印波特率-----
    virtual_mcu_debug_tx_pin = IO_PORTA_01; //调试用的串口打印接口
    virtual_mcu_debug_baudrate = 460800; //调试用的串口波特率
}
```

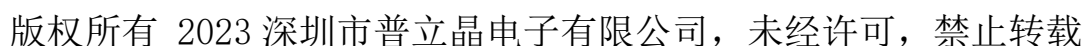
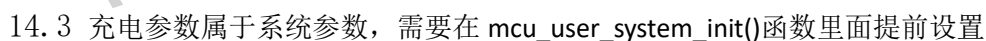
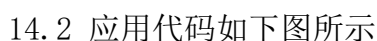
15.3 打印信息通过如下程序设置

```
printf("checksum error\n");
//打印字符串结果为，例 checksum error

printf("adc value5 = %d\n",value);
//打印变量值结果为，例 adc value5 = 1023

printf_buf(buff,buff_size);
//打印数组长度为 3 的结果为，例 01 02 03
```

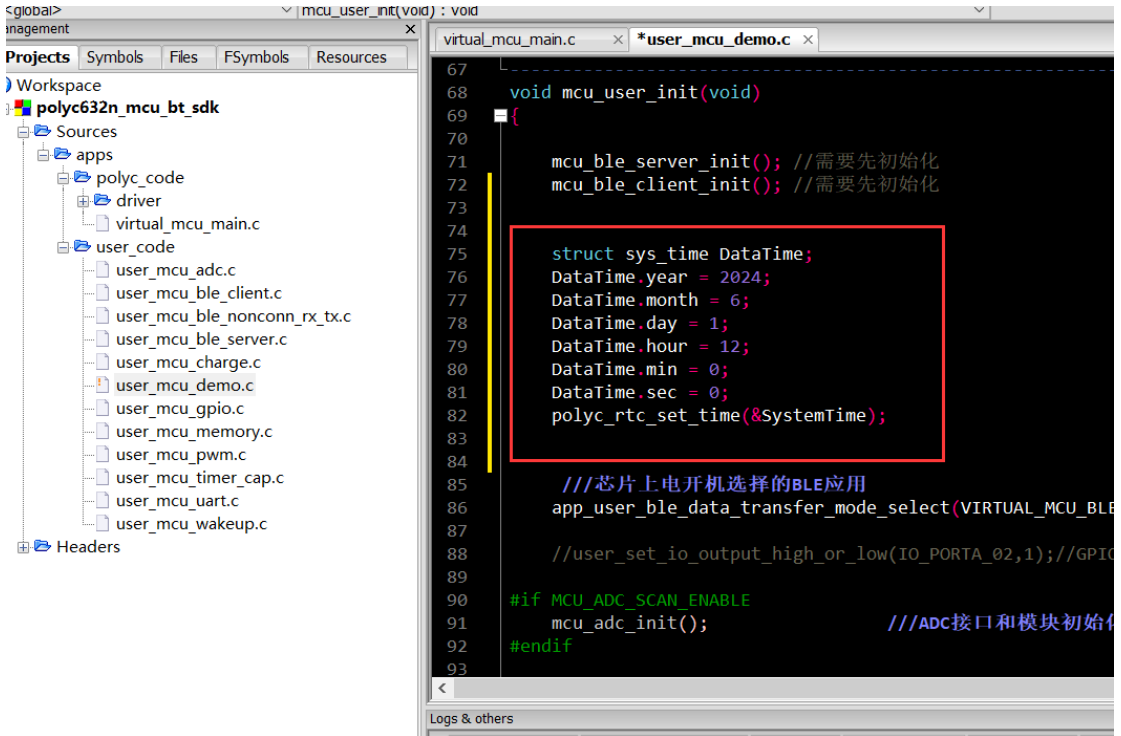
14.1 内置充电使能、充电电流、满电电压、满电电流等参数配置；如下图文件所示



15. RTC 时钟获取与设置

15.1 rtc 时钟设置

在 mcu_user_init 初始化 rtc 时钟

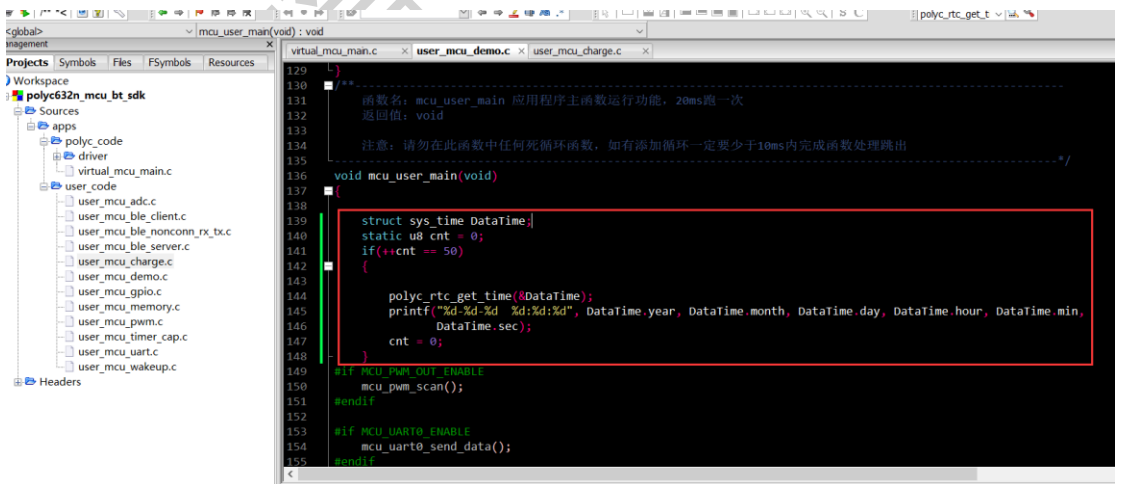


The screenshot shows the IDE interface with the project 'polyc632n_mcu_bt_sdk' open. The file explorer on the left shows the project structure, including the 'user_code' directory. The main editor window displays the 'mcu_user_init(void)' function in 'virtual_mcu_main.c'. The function is highlighted with a red box, showing the initialization of the 'DataTime' struct and the call to 'polyc_rtc_set_time(&SystemTime);'.

```
67  
68 void mcu_user_init(void)  
69 {  
70  
71     mcu_ble_server_init(); //需要先初始化  
72     mcu_ble_client_init(); //需要先初始化  
73  
74  
75     struct sys_time DateTime;  
76     DateTime.year = 2024;  
77     DateTime.month = 6;  
78     DateTime.day = 1;  
79     DateTime.hour = 12;  
80     DateTime.min = 0;  
81     DateTime.sec = 0;  
82     polyc_rtc_set_time(&SystemTime);  
83  
84  
85     ///芯片上电开机选择的BLE应用  
86     app_user_ble_data_transfer_mode_select(VIRTUAL_MCU_BLE  
87  
88     //user_set_io_output_high_or_low(IO_PORTA_02,1); //GPIO  
89  
90     #if MCU_ADC_SCAN_ENABLE  
91     mcu_adc_init(); //ADC接口和模块初始  
92     #endif  
93
```

15.2 rtc 时钟获取

自定义一个结构体变量 polyc_time，调用 polyc_rtc_get_time（）接口获取时钟，并将其打印出来。



The screenshot shows the IDE interface with the project 'polyc632n_mcu_bt_sdk' open. The file explorer on the left shows the project structure, including the 'user_code' directory. The main editor window displays the 'mcu_user_main(void)' function in 'virtual_mcu_main.c'. The function is highlighted with a red box, showing the retrieval of the 'DataTime' struct and the call to 'polyc_rtc_get_time(&DateTime);' followed by a printf statement to print the time.

```
129  
130  
131  
132  
133  
134  
135  
136  
137  
138  
139  
140  
141  
142  
143  
144  
145  
146  
147  
148  
149  
150  
151  
152  
153  
154  
155
```