遥控器应用入门指南

发布 ***1.2.0***

2019 年 **04** 月 **11** 日

目录

CHAPTER 1

概述

1.1 文档目的

方便开发人员快速入门，熟悉语音遥控器应用的开发。

1.2 术语说明

表 1.1: 术语说明

术语	说明
BLE	蓝牙低功耗技术
RMC	remote controller，遥控器

1.3 参考文档

- https://www.bluetooth.com/specifications/gatt

1.4 版本历史

表 1.2: 版本历史

日期	版本	注释	作者
2018-08-30	1.0	初始版本	ZS110A 项目组
2019-02-26	1.1	hid 语音遥控器 cache 版本	ZS110A 项目组
2019-04-10	1.2	hid ota 说明及红外自学功能介绍	ZS110A 项目组

CHAPTER 2

语音遥控器应用简介

语音遥控器应用是基于 BLE HID profile 开发的语音遥控器应用。

2.1 遥控器工程目录结构介绍

- 工程路径在：\samples\voice_rcu\
- 工程目录结构，如图所示

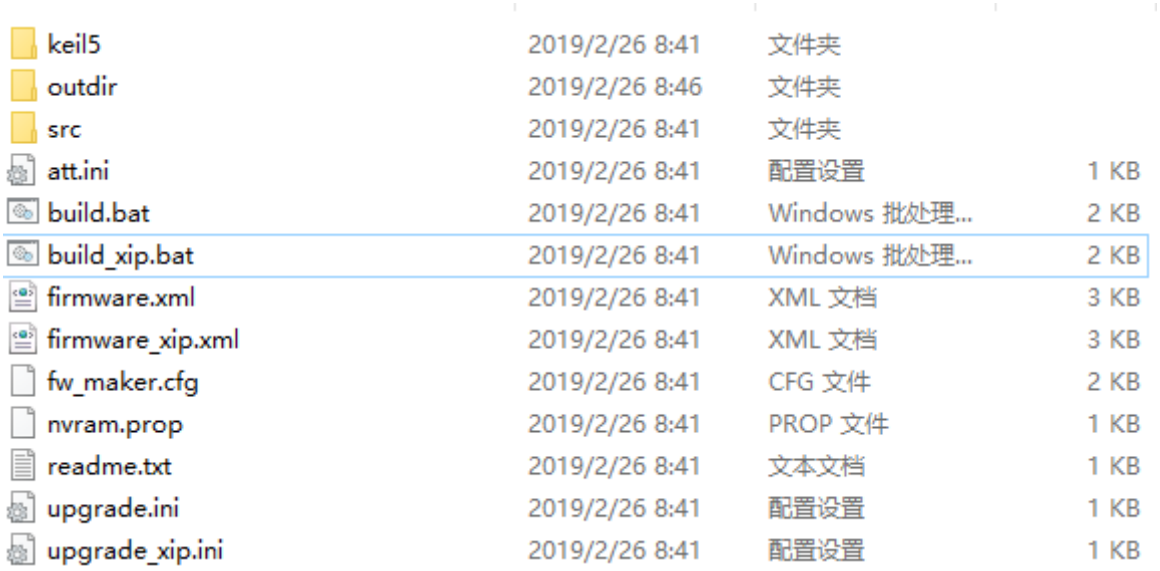


图 2.1: 工程目录结构图

1. keil 目录主要包含 3 个 keil 工程
2. outdir 目录主要包含工程产生的固件及 OTA 升级包
3. src 目录主要包含工程的源码
4. voirce_rcu 目录下的脚本主要用于生成固件和 OTA 升级包

2.2 遥控器工程概况

工程主要由 loader 工程、peripheral_rmc 工程和 rf_test 工程构成，如图所示:

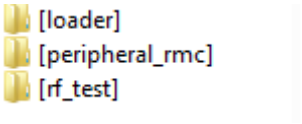


图 2.2: 工程概况图

- peripheral_rmc 是遥控器应用工程，主要实现遥控器的主要功能
- loader 是应用引导工程，根据不同的需求，引导遥控器应用工程或者 ble 频偏测试工程，默认情况，是引导应用工程，如需要测试频偏，可以在遥控器应用场景，通过组合键进入频偏测试模式，进而引导 ble 频偏测试工程。
- rf_test 是 ble 频偏测试工程。工厂生产组装好遥控器后需要对遥控器成品进行频偏确认时使用。

CHAPTER 3

loader 工程简介

引导和加载应用工程。

更多与 loader 相关的信息，请参考 <<ZS110A-loader 示例 >> <<ZS110A-SDK 架构介绍 >>

CHAPTER 4

遥控器工程简介

4.1 rmc project 目录介绍

- rmc project 目录结构，如图所示

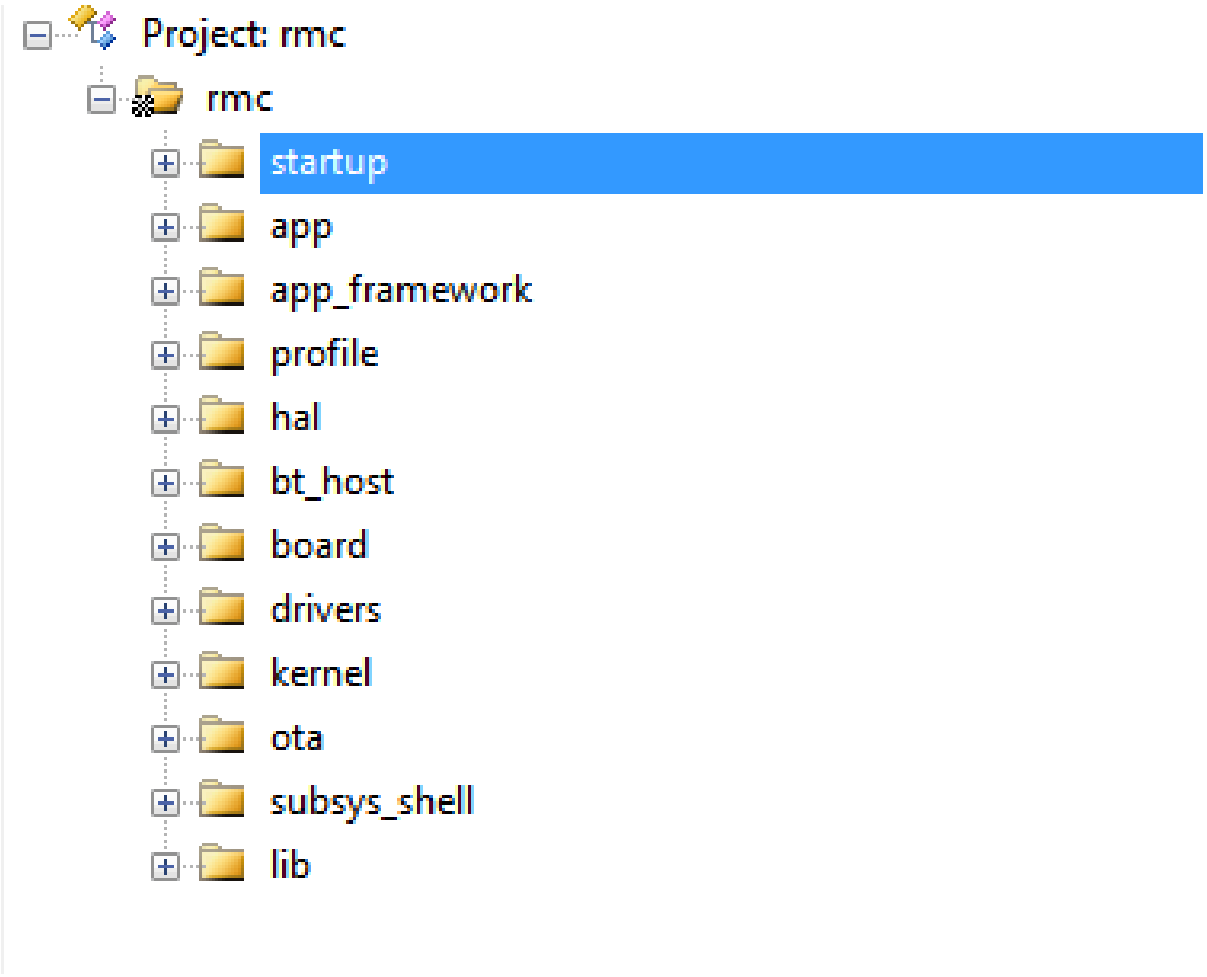
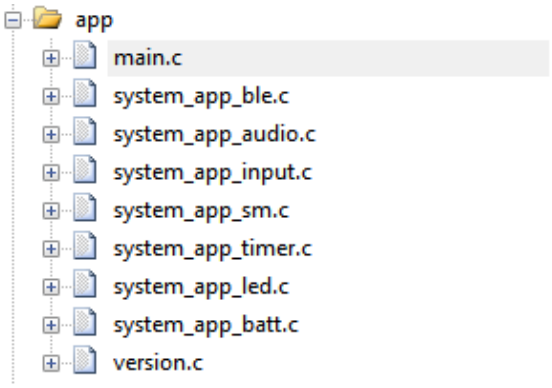


图 4.1: rmc project 目录结构图

- startup 系统启动相关功能代码
- app 应用层代码
- app_framework 应用层框架代码
- profile Gatt profile 的代码
- hal 硬件抽象层代码
- bt_host bt host 层代码
- board 板级相关初始化代码
- drivers 驱动相关代码
- kernel Zephyr 内核相关代码
- ota 相关代码
- subsys_shell shell 相关代码
- lib 语音编码库

4.2 应用层代码介绍

- 应用层结构如图所示



4.2.1 main.c 介绍

- 应用层相关功能模块初始化

```
/**
 * @brief main module initialization of the application.
 *
 * @details Initialize msg/led/input/sm etc.
 *
 * @return No return value.
 */
__init_once_text void system_app_init(void)
{
        /* init message manager */
        msg_manager_init();

        /*led manager initialization */
        led_manager_init();

        /* input manager initialization */
        system_input_handle_init();

        /* led handle initialization */
        system_led_handle_init();

        /* state machine of rmc initialization */
        rmc_sm_init();

        /* app timer of rmc initialization */
        rmc_timer_init();

        /* audio module initialization */
```

(continues on next page)

(续上页)

```
        audio_init();

        /* batt manager initialization */
        batt_manager_init();
}
```

- 进入等待消息处理循环

```
/**
 * @brief the entry function of the application.
 *
 * @details Initialize app module and enter into msg
↪mainloop.
 *
 * @return No return value.
 */
void app_main(void)
{
        int err;
        struct app_msg msg = {0};
        int result = 0;

        /* main module initialization */
        system_app_init();

        /* Initialize the Bluetooth Subsystem */
        app_get_wake_lock();
        err = bt_enable(bt_ready);
        if (err) {
                app_release_wake_lock();
                SYS_LOG_ERR("Bluetooth init failed (err %d)",
↪ err);
                return;
        }

        /* start the timer of No action */
        rmc_timer_start(ID_NO_ACT_TIMEOUT);

        /* enter into message processing main loop */
        while (1) {
                if (receive_msg(&msg, K_FOREVER)) {
                        switch (msg.type) {
                        case MSG_KEY_INPUT:
```

4.2.2 system_app_ble.c 介绍

- ble 相关配置

– 连接参数配置

```
struct bt_le_conn_param hid_app_update_cfg = {
        .interval_min = (10),
        .interval_max = (10),
        .latency = (44),
        .timeout = (400),
};
```

– 广播参数配置

```
#define BT_LE_ADV_CONN_UNDIRECT BT_LE_ADV_PARAM(
                BT_LE_ADV_OPT_CONNECTABLE |
                BT_LE_ADV_OPT_ONE_TIME,
                BT_GAP_ADV_FAST_INT_MIN_1,
                BT_GAP_ADV_FAST_INT_MAX_1)
```

– 广播内容配置

```
static struct bt_data ad[] = {
        BT_DATA_BYTES(BT_DATA_FLAGS, (BT_LE_AD_GENERAL | BT_
↪LE_AD_NO_BREDR)),
        BT_DATA_BYTES(BT_DATA_UUID16_ALL,
                      0x12, 0x18, /* HID Service */
                      0x0f, 0x18), /* Battery Service */
        BT_DATA_BYTES(BT_DATA_GAP_APPEARANCE, 0x80, 0x01),
        BT_DATA(BT_DATA_NAME_COMPLETE, DEVICE_NAME, DEVICE_
↪NAME_LEN),
};
```

注解:

– 对于 BLE HID 应用，广播包需要添加 HID service、Battery Service UUID、APPEARANCE 以及广播名称

– 广播应答包内容配置

```
static struct bt_data sd[] = {
        BT_DATA_BYTES(BT_DATA_MANUFACTURER_DATA, 0xe0, 0x03),
};
```

– Hid profile 相关配置，如 report 描述符配置等

```
const u8_t hid_report_map[] = {
0x05, 0x0c, /*  Usage Page (Consumer Devices) */
0x09, 0x01, /*  Usage (Consumer Control) */
0xa1, 0x01, /*  Collection (Application) */
0x85, HIDAPP_REMOTE_REPORT_ID, /*       report ID (0x01) */
0x19, 0x00, /*  USAGE_MINIMUM (0) */
```

(continues on next page)

(续上页)

```
0x2a, 0x9c, 0x02, /*    USAGE_MINIMUM (0x29c) */
0x15, 0x00, /*  Logical Minimum (0) */
0x26, 0x9c, 0x02, /*    Logical Maximum (0x29c) */
0x95, 0x01, /*  Report Count (1) */
0x75, 0x10, /*  Report Size (16) */
0x81, 0x00, /*  Input (Data, array, Absolute) */
0x09, 0x02, /*  Usage (Numeric Key Pad ) */
0xa1, 0x02, /*  Collection (Application) */
0x05, 0x09, /*  Usage Page (button) */
0x19, 0x01, /*  Usage Minimum (1) */
0x29, 0x0a, /*  Usage Maximum (10) */
0x15, 0x01, /*  Logical Minimum (1) */
0x25, 0x0a, /*  Logical Maximum (10) */
```

注解:

- hid 的具体配置方法，请参考 https://www.bluetooth.com/specifications/gatt 中 HID Service 及 HID over GATT Profile

- ble 回调函数处理

```
static struct bt_conn_cb conn_callbacks = {
.connected = connected,
.disconnected = disconnected,
.identity_resolved = le_identity_resolved,
.security_changed = security_changed,
.le_param_updated = le_param_updated,
};
```

  - connected 蓝牙连接事件的回调
  - disconnected 蓝牙断开事件的回调
  - le_identity_resolved 蓝牙加密过程中私有地址解析的回调
  - security_changed 蓝牙加密后的回调

- ble 事件处理
  - 通过 BLE 发送普通遥控器键值的接口

```
/**
 * @brief function of sending remote key event
 *
 * @param remote key code.
 *
 * @return true if invoked succsess.
 * @return false if invoked failed.
 */
```

(continues on next page)

(续上页)

```
int hid_app_remote_report_event(u16_t button);
```

- 通过 BLE 发送语音数据的接口

```
/**
 * @brief function of sending voice data event
 *
 * @param the buffer which contain audio data.
 *
 * @return true if invoked succsess.
 * @return false if invoked failed.
 */

int hid_app_voice_report_event(u8_t *buffer);
```

4.2.3 system_app_audio.c 介绍

- 语音的编码配置

```
#if (defined(USE_AL_ENCODE_1) || defined(USE_AL_ENCODE_2_8_
↪1) || defined(USE_AL_ENCODE_2_8_1_2))
        #define CONFIG_AUDIO_SAMPLE_RATE 16
        #define PCM_BUF_LEN  640
#else
        #define CONFIG_AUDIO_SAMPLE_RATE 8
        #define PCM_BUF_LEN  320
#endif
#define PCM_BUF_NUM  4
```

其中，

- PCM_BUF_LEN 是编码帧大小，PCM_BUF_NUM ADC 采集缓冲的 buffer 个数
- 如果配置编码算法为压缩比 8:1 及以上算法，ADC 自适应调整 16K—16bit 采样
- 默认 16：1 算法，ADC 16K_16bit 采样。

- 语音的编码

```
/**
 * @brief function of audio coding
 *
 * @param in_buffer audio data buffer before encoding.
 * @param out_buffer encoded audio data buffer.
 *
 * @return encoded audio buffer size.
```

(continues on next page)

(续上页)

```
 */

u16_t audio_encode(void *in_buffer, u8_t *out_buffer);
```

- 语音的发送

```
void system_audio_event_handle(void *ptr)
{
        u8_t *out_buffer = encode_out_buffer;
        u16_t i, out_len = 0;

        if (rmc_sm_state_is_activing())
                return;

        /* send start flag */
        audio_send_start_flag();

        if (audio_need_drop_buffer())
                return;

        out_len = audio_encode((void *)ptr, out_buffer);
        count += out_len;
        SYS_LOG_DBG("send (%d) data", count);

        for (i = 0; i < out_len; i = i+20)
                hid_app_voice_report_event(&encode_out_
↪buffer[i]);

        audio_send_stop_flag();
}
```

  - 语音发送之前需要发送开始标识
  - 语音发送结束后需要结束标识
  - 通过 ble 发送前，会丢掉 80ms 质量较差的语音.

4.2.4 system_app_input.c 介绍

- 按键转化
  - 将按键转化为红外键值
  - 将按键转化为蓝牙 HID 键值
- 按键发送
    - 红外发送

```
/**
 * @brief send irc keycode using tx channel
 *
 * @param irc device handle
 * @param irc key code
 * @param key_state which is key_down or key_up
 *
 */

void irc_device_send_key(void *handle, u32_t key_code, u8_t␣
↪key_state);
```

– 蓝牙发送

```
void send_ble_keycode(u32_t key_value)
```

4.2.5 system_app_sm.c 介绍

通过应用的各种事件触发，进遥控器进入不同的工作状态

- 触发事件，主要包含按键事件，蓝牙事件，定时器事件等

```
/* RMC event handler messages for state machine */
enum {
        /* messages from input */
        RMC_MSG_INPUT_KEY,
        /* messages from BLE */
        RMC_MSG_BLE_CONNECTED,
        RMC_MSG_BLE_SEC_CONNECTED,
        RMC_MSG_BLE_INPUT_ENABLED,
        RMC_MSG_BLE_USER_DISCONNECTED,
        RMC_MSG_BLE_NON_USER_DISCONNECTED,

        /* messages from appTimer */
        RMC_MSG_TIMER_ADV_TIMEOUT,
        RMC_MSG_TIMER_DIRECT_ADV_TIMEOUT,
        RMC_MSG_TIMER_NO_ACT_TIMEOUT,
        RMC_MSG_TIMER_SEC_TIMEOUT,

        /* message from others */
        RMC_MSG_RECONN_FAIL,
        RMC_MSG_POWER_UP,
        RMC_MSG_CLEAR_PIAR_KEY,
        RMC_MSG_LOW_POWER,
        RMC_NUM_MSGS
};
```

- 工作状态，主要包含 IDLE 状态、正激活状态、工作状态

```
/*Rmc State machine states */
enum {
        RMC_STATES_IDLE,
        RMC_STATES_ADVERTISING,
        RMC_STATES_CONNECTED,
        RMC_STATES_NUM,
};
```

- 状态转化执行过程

  主要是根据当前状态及正在发送的事件，进而转化成不同的状态，详细过程如下面函数所示

```
void rmc_sm_execute(u8_t evt)
{
        rmc_act_t       act_func;
        u8_t            action;
        u8_t            event;

        SYS_LOG_INF("rmc_sm_execute event=%d state=%d", evt,␣
↪g_rmc_cb.rmc_status);

        /* get the event */
        event = evt;

        /* get action */
        action = rmc_State_table[g_rmc_cb.rmc_
↪status][event][RMC_ACTION];

        /* set next state */
        g_rmc_cb.rmc_status = rmc_State_table[g_rmc_cb.rmc_
↪status][event][RMC_NEXT_STATE];

        /* look up action set */
        act_func = rmc_act_set[action];

        /* if action set present */
        if (act_func != NULL) {
                /* execute action function in action set */
                act_func(event, NULL);
        } else {
                /* no action */
                rmc_sm_act_none(event, NULL);
        }
}
```

4.2.6 system_app_timer.c 介绍

主要是通过 workqueue 实现几个应用需求的 timer

- timer 的功能

```
enum {
        ID_ADV_TIMEOUT,
        ID_DIRECT_ADV_TIMEOUT,
        ID_NO_ACT_TIMEOUT,
        ID_ADC_BATTERY_TIMEOUT,
        ID_PAIR_COMB_KEY_TIMEOUT,
        ID_HCI_MODE_COMB_KEY_TIMEOUT,
        ID_NUM_TIMEOUTS,
};
```

  - 非定向广播超时时间
  - 定向广播超时时间
  - 系统无操作超时时间
  - 电池上报时间
  - 进入配对模式的组合键生效时间
  - 进入 ble 频偏测试模式的组合键生效时间

- timer 的时间配置

```
#define CONFIG_RMC_ADV_TIMEOUT              30
#define CONFIG_RMC_DIRECT_ADV_TIMEOUT       2
#define CONFIG_RMC_NO_ACT_TIMEOUT           (5*60)
#define CONFIG_RMC_ADC_BATTERY_TIMEOUT      (3*60)
#define CONFIG_RMC_PAIR_COMB_KEY_TIMEOUT    5
#define CONFIG_RMC_HCI_MODE_COMB_KEY_TIMEOUT  8
#define CONFIG_RMC_SEC_TIMEOUT    6
```

  用户可以根据需求自行配置这些时间.

4.2.7 system_app_led.c 介绍

实现应用的各种场景灯显示

- 灯闪烁功能实现有 2 个方式
  - GPIO 方式
  - PWM 方式
- 遥控器各场景使用的灯的周期配置

```
/*low power indicator */
#define LOW_POWER_IND_BLINK_PERIOD 100 /* 100 ms*/
#define LOW_POWER_IND_TIMEOUT 3000 /* 3000 ms*/

#define IR_BTN_IND_BLINK_PERIOD 200 /* 200 ms*/
```

(continues on next page)

(续上页)

```
#define IR_BTN_IND_TIMEOUT 400 /* 400 ms*/

/* ble paring indicator */
#define PAIRING_IND_BLINK_PERIOD 400 /* 400 ms*/

/* ble pair success indicator */
#define PAIRED_IND_BLINK_PERIOD 1000 /* 1000 ms*/
#define PAIRED_IND_TIMEOUT 2000 /* 2000 ms*/

/* led indication event */
#define BTN_DOWN        LED_ON
#define BTN_UP          LED_OFF
```

用户可以根据需求自行配置这些时间.

4.2.8 system_app_batt.c 介绍

实现应用的电池管理

- 电池管理配置

```
#define VERY_LOW_BATT_VAL       1800    /* mv */
#define LOW_BATT_VAL            2000    /* mv */
#define FULL_BATT_VAL           3250    /* mv */
```

用户自定义遥控器，可以自行修改这些参数达到想要的电池管理效果。

4.2.9 version.c 介绍

应用版本维护

```
char sys_version[] = "2.0.00.18101609";
```

4.3 startup 启动代码介绍

关于 startup 启动相关代码，详细请参考 <<ZS110A-SDK 架构介绍 >>

4.4 app_framework 应用层框架代码

- input_mamager 按键输入管理
  - 驱动层消息的转发

  - 驱动层消息的过滤，如是否转发 repeat key 到应用，应用未初始化完成锁住消息，不允许消息上报

- led_manager led 灯资源的管理

```
const struct led_config led_configs[] = {
LED_PIN_CONFIG
};
```

  详细配置，请参考 <<ZS110A-方案配置指南 >>

- msg_manager 消息的发送和接收
  - 同步消息发送
  - 异常消息发送
  - 消息 buffer 的配置

```
NET_BUF_POOL_DEFINE(msg_buffer_pool_async, 8, sizeof(struct␣
↪app_msg), 4, NULL);
NET_BUF_POOL_DEFINE(msg_buffer_pool_sync, 8, sizeof(struct␣
↪app_msg), 4, NULL);
```

  注解:
  - 同步消息个数，建议根据应用需求配置足够多，但同时注意避免浪费
  - 异步消息个数，建议保证应用的正常需求个数，如按键消息，如果主线程处理不过来，从设计上允许丢按键，保证系统不卡住。

4.5 profile Gatt profile 的代码

HID profile 实现了 Hid profile spec 要求的必须实现的 3 个 service，如下所示

- hid service
- battery service
- discovery serivce

具体实现规格，请参考

https://www.bluetooth.com/specifications/gatt 中

HID Service 和 HID over GATT Profile

4.6 hal 硬件抽象层代码

主要系统驱动层的简单抽象，让应用程序开发者，不需要关注底层实现，更注重功能接口。更多功能接口，请参考 <<ZS110A API 指南 >>

4.6.1 irc hal 介绍

- 主要接口介绍

1. 打开红外设备

   主要是打开红外驱动，返回设备 handle，便于后续操作设备。

```
/**
 * @brief open irc device
 *
 * @param irc device name which will be open
 *
 */
void *irc_device_open(char *dev_name);
```

2. 使能红外接收通道

   主要是注册接收红外键值回调函数以及使能驱动的红外接收功能 (配置控制器及使能时钟等)

```
/**
 * @brief enable irc rx channel
 *
 * @param irc device handle
 * @param input_notify_t be used for notify irc rx␣
↪channel keycode
 *
 */
void irc_device_enable_rx(void *handle, input_notify_t␣
↪cb);
```

   注解: 由于红外驱动中断没有唤醒功能，使能红外接收通道后，系统将不会休眠，如不需要接收功能时，注意调用禁掉红外接收通道接口，以节省功耗。

3. 禁掉红外接收通道

   主要是关掉驱动的红外接收功能

```
/**
 * @brief disable irc rx channel
 *
```

(continues on next page)

(续上页)

```
 * @param irc device handle
 *
 */
void irc_device_disable_rx(void *handle);
```

4. 硬件红外发送接口

   通过红外硬件控制器发送红外码

```
/**
 * @brief send irc keycode using tx channel
 *
 * @param irc device handle
 * @param irc key code
 * @param key_state which is key_down or key_up
 *
 */
void irc_device_send_key(void *handle, u32_t key_code, u8_
↪t key_state);
```

> **注解:** 硬件控制器仅支持 9012、NEC、RC5、RC6 红外协议

5. 软件红外发送接口

   通过软件方式发送红外码

```
/**
 * @brief send irc keycode using tx channel
 *
 * @param irc device handle
 * @param irc key code
 * @param key_state which is key_down or key_up
 *
 */
void irc_device_send_learn_key(void *handle, u32_t key_
↪code, u8_t key_state);
```

> **注解:** 当前软件发送接口仅支持自学习区按键的红外发送

6. 退出自学习模式接口

   当前设置了 2 种退出学习模式的方式:

   – 设置一个超时时间，超时时间到，退出自学习模式

   – 按学习区按键以外的任意键，退出自学习模式

```
void irc_device_exit_learn_mode(void *handle);
```

- 软件自学习事件处理

```
void irc_hal_event_handle(struct device *dev, struct input_
↪value *ir_val)
```

将软件红外驱动解调出来的脉冲序列进行分析校验，是否学习成功的校验策略如下：

- 根据预设的协议，如果波形符合预设的协议，认为自学习是成功的，告诉用户学习成功
- 根据预设的协议，如果波形不符合预设的协议，认为自学习是失败的，告诉用户学习失败
- 对于非预设的协议，认为自学习是失败的，告诉用户学习失败，但是学习的波形仍然储存，供用户自己 try，能用则学习成功

该策略的优缺点如下：

- 优点：对已有的协议，能进行有效判断，减少虚警/漏报
- 缺点：无法对未知协议做有效判断，增加多种协议判断，将会增加比较多的代码

如何将该策略更加完善，主要是增加更多先验协议知识，让校验策略更加精准，增加协议判断的方法如下：

1. 在 check_irc_wave 增加先验协议判断函数如 is_nec_wave(pulse_tab, tab_len)

```
u8_t check_irc_wave(u16_t *pulse_tab, u16_t tab_
↪len)
{
        if (is_nec_wave(pulse_tab, tab_len))
                return IR_LEARN_SUCCESS;
        else if (is_9012_protocol(pulse_tab, tab_
↪len))
                return IR_LEARN_SUCCESS;
        else if (is_rc5_protocol(pulse_tab, tab_
↪len))
                return IR_LEARN_SUCCESS;
        else if (is_rc6_protocol(pulse_tab, tab_
↪len))
                return IR_LEARN_SUCCESS;
        else if (is_7461_protocol(pulse_tab, tab_
↪len))
                return IR_LEARN_SUCCESS;
        else if (is_50560_protocol(pulse_tab, tab_
↪len))
                return IR_LEARN_SUCCESS;
        else if (is_50462_protocol(pulse_tab, tab_
↪len))
```

(continues on next page)

(续上页)

```
                return IR_LEARN_SUCCESS;

        return IR_LEARN_FAILED;
}
```

2. 实现先验协议判断函数

   主要包含红外码流长度判断，引导码判断，码值判断等。

```
u8_t is_nec_wave(u16_t *pulse_tab, u16_t tab_len)
{
        u8_t i, match;
        u32_t data   = 0;

        /* check length */
        if(tab_len != (NEC_CODE_NUM * 2 + 2 + 1))
                return false;

        /* check LDC0 */
        match = (pulse_tab[0] > VAL_ESP_L(NEC_LDC_
↪0)) && (pulse_tab[0] < VAL_ESP_R(NEC_LDC_0));
        if(!match)
                return false;

        /* check LDC1 */
        match = (pulse_tab[1] > VAL_ESP_L(NEC_LDC_
↪1)) && (pulse_tab[1] < VAL_ESP_R(NEC_LDC_1));
        if(!match)
                return false;

        /* check cmd */
        for (i = 0; i < (NEC_CODE_NUM * 2); i =␣
↪i+2) {
                match = ((pulse_tab[i+2] > VAL_
↪ESP_L(NEC_COM)) && (pulse_tab[i+2] < VAL_ESP_
↪R(NEC_COM)));
                if (!match)
                        return false;

                if((pulse_tab[i+3] > VAL_ESP_
↪L(NEC_LOGIC_0)) && (pulse_tab[i+3] < VAL_ESP_
↪R(NEC_LOGIC_0)))
                        data = (data << 1) | 0;
                else if ((pulse_tab[i+3] > VAL_
↪ESP_L(NEC_LOGIC_1)) && (pulse_tab[i+3] < VAL_
↪ESP_R(NEC_LOGIC_1)))
                        data = (data << 1) | 1;
                else
                        return false;
```

(continues on next page)

(续上页)

```
        }

        generate_nec_wave(pulse_tab, tab_len,␣
↪data, NEC_CODE_NUM);

        printk("nec code : 0x%x\n", data);
        return true;
}
```

注解： 不同的协议有一定容错范围，当前设置的容错范围为 #define ESP (20)，开发者可以自行调整

3. 先验码流的波形校正

   如果判断好是已经协议，可以将波形脉宽进行校正，以防止误差累积，达到最好的后续发射效果

```
void generate_nec_wave(u16_t *pulse_tab, u16_t␣
↪tab_len, u64_t data, u16_t bits_of_data)
{
        u32_t mask;
        u8_t i = 0;

        /* header */
        pulse_tab[i++] = NEC_LDC_0;
        pulse_tab[i++] = NEC_LDC_1;

        /* code */
        for (mask = (1UL << (bits_of_data - 1)); ␣
↪mask;  mask >>= 1) {
                if (data & mask) {
                        pulse_tab[i++] = NEC_COM;
                        pulse_tab[i++] = NEC_
↪LOGIC_1;
                } else {
                        pulse_tab[i++] = NEC_COM;
                        pulse_tab[i++] = NEC_
↪LOGIC_0;
                }
        }

        /* stop burst */
        pulse_tab[i++] = NEC_COM;

        printk("generate_nec_wave\n");
}
```

4.7 bt_host bt host 层代码

zephyr bt_host 相关配置及 patch 文件详细的配置，请参考 <<ZS110A-方案配置指南 >>

4.8 drivers 驱动相关代码

驱动实现相关源码，详情请参考 <<ZS110A 驱动用户指南 >>

4.9 ota 相关代码

使用 OTA 进行升级的源码，详情请参考 <<ZS110A-OTA 指南 >>

CHAPTER 5

rf_test 工程简介

5.1 频偏测试

遥控器进入频偏测试模式后自动在指定频点 (默认为 19 通道，频点为 2440MHz) 上发送 single tone。

将遥控器靠近频谱仪的接收天线，从频谱仪上得到实际频率和频偏值后可以判断遥控器的频偏是否在合理范围内。

5.2 如何修改频点

下面代码中 0x13(十六进制) 表示 19(十进制) 通道。修改该值就可以改变信号频点。

```
u8_t BUF_CONTINUOUS_TX[] = {0x0F, 0xFE, 0x03, 0x13, 0x0, 0x00};
```

CHAPTER 6

固件更新

6.1 固件更新方式简介

主要有 3 种更新方式，分别是

- ATT 工具更新
- keil download 更新
- OTA 升级更新

> **注解：** OTA 升级，小机中必须有固件

6.2 keil 固件更新

- 编译 3 个工程，
- 编译成功后，在 Keil 工具界面点击”Load”即可完成固件更新。
  - rf_test 是非必须工程，可以不下载，其他两个工程必须下载
  - 编译时，注意板型的选择

```
//#include "rmc_atb1103_yt.h"
//#include "rmc_atb1103_yt_v2.h"
#include "rmc_atb1103_yt_v21.h"
//#include "rmc_atb110x_dvb_v10.h"
//#include "rmc_atb1103_atv.h"
```

6.3 ATT 固件更新

- 编译 3 个工程
- 点击\samples\voice_rcu\build_xip.bat 脚本，生成\samples\voice_rcu\outdir\zs110a_atf.fw
- 通过 ATT 工具更新固件，详细操作过程，请参考 <<ZS110A 自动化测试工具 (BLE) 使用说明 >>

6.4 OTA 升级更新

- 编译 3 个工程
- 点击\samples\voice_rcu\build_xip.bat 脚本，生成\samples\voice_rcu\zs110a_*_ota.zip
- 通过 OTA APK 升级到样机，详细参考 OTA 升级相关文档

CHAPTER 7

语音遥控器功能使用简介

- 使用 Actions 自定义 profile 遥控器功能演示
- 使用 HID profile 遥控器功能演示
- non-hid 遥控器 ota 升级演示
- hid 遥控器 ota 升级演示

7.1 使用 Actions 自定义 profile 遥控器功能演示

7.1.1 软硬件资源准备

- 13/16 key 遥控器或者 EVB 开发板
- android 设备

7.1.2 功能演示

以 16key 遥控器为例

- 遥控器工程使用非 hid 遥控器工程

> **注解:** 工程路径 \samples\voice_rcu_non_hid

- 编译并更新固件到遥控器
- 安装 Actions VoiceBleTest_V*.*.*.apk 到 android 设备，并打开

- 按遥控器的配对组合键 (“HOME”键 + “BACK”键) 进入配对模式，看到灯闪
- 在 APK 中，点击“START SCAN”开始搜索，然后点击“CONNECT”连接蓝牙名字为“BLE_YT_RMC”的 BLE 设备

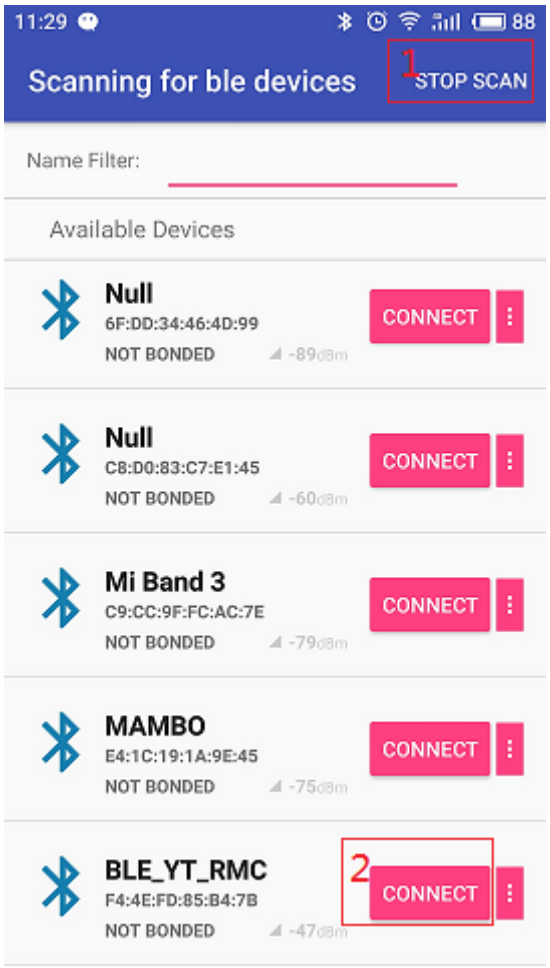


图 7.1: APK 主界面图

- 按遥控器的非语音键，界面会显示相应的键值，如图所示

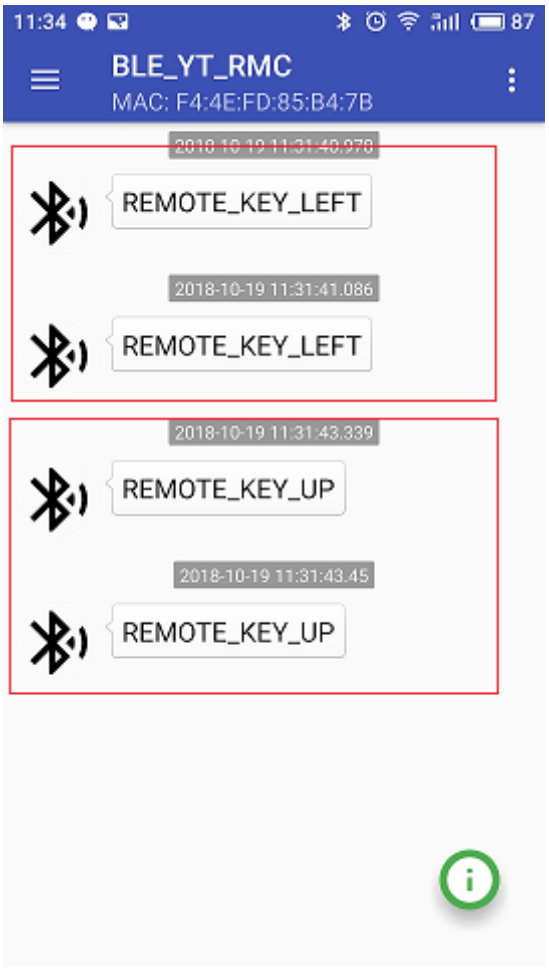


图 7.2: 按键显示效果图

- 按遥控器的语音键，界面会显示一段“语音”标识，如下图所示

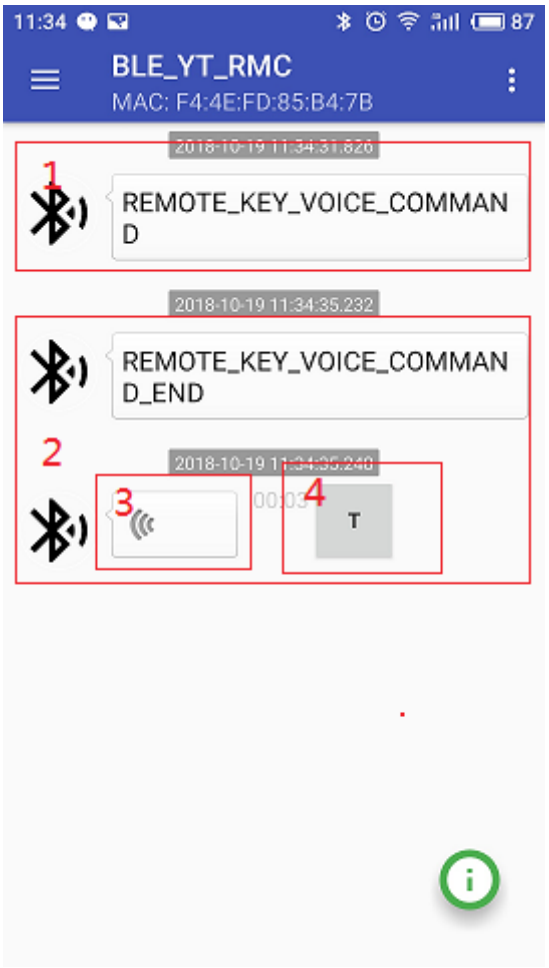


图 7.3: 语音显示效果图

- 点击“语音”标识，会播放录制的语音片段

> **注解:** 录制的语音片段保存在应用的 \com.actions.voicebletest\pcm\ 目录，可以导出到 pc 查看

- 点击“T”标识，会将语音片段翻译成文字，如下图所示

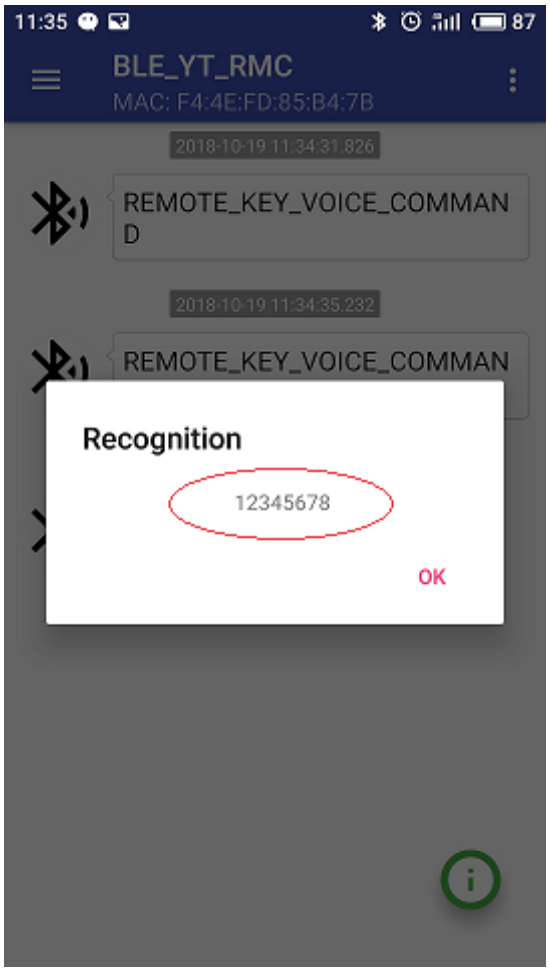


图 7.4: 语音翻译效果图

注解: 语音识别需要联网，请确保 android 设备网络畅通.

7.2 使用 HID profile 遥控器功能演示

7.2.1 软硬件资源准备

- 13/16 key 遥控器或者 EVB 开发板
- android 设备

7.2.2 功能演示

以 16key 遥控器为例

- 遥控器工程使用 hid 遥控器工程

> **注解：** 工程路径 \samples\voice_rcu

- 编译并更新固件到遥控器
- 按遥控器的配对组合键 (“HOME”键 + “BACK”键) 进入配对模式，看到灯闪
- 在 android 设备的 settings 中搜索并连接蓝牙名字为“BLE_YT_RMC”的 BLE 设备，如图所示

图 7.5: android settings 界面

- 按遥控器的音量键，就会弹出 android 设备的音量调节对话框，如图所示

图 7.6: 操作音量界面

注解:

- 语音按键在没有适配过语音通路的设备，是没有效果的

7.3 non-hid 遥控器 ota 升级演示

7.3.1 软硬件资源准备

- 13/16 key 遥控器或者 EVB 开发板
- android 设备

7.3.2 功能演示

以 16key 遥控器为例

- 编译并更新固件到遥控器

- 将 \samples\voice_rcu_non_hid\outdir\ 路径生成的 ota 包 zs110a_***_ota.zip 拷贝到手机 sdcard 卡
- 安装 Actions VoiceBleTest_V*.*.*.apk 到 android 设备，并打开
- 按遥控器的配对组合键 (“HOME”键 + “BACK”键) 进入配对模式，看到灯闪
- 在 APK 中搜索并连接蓝牙名字为“BLE_YT_RMC”的 BLE 设备，进入主界面，点击左上角的设置菜单，如图所示

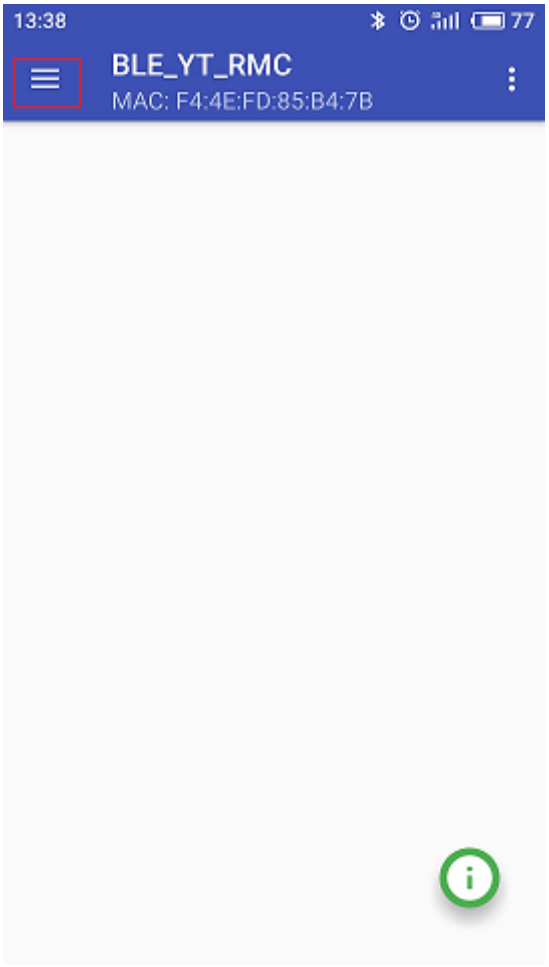


图 7.7: 语音操作界面

- 点击左上角的设置菜单后，进入功能显示选择页，如图所示

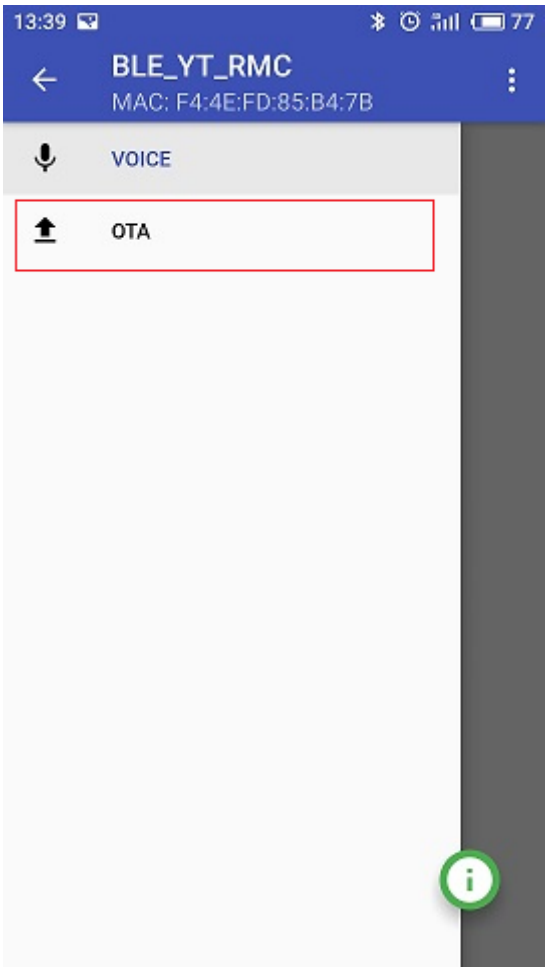
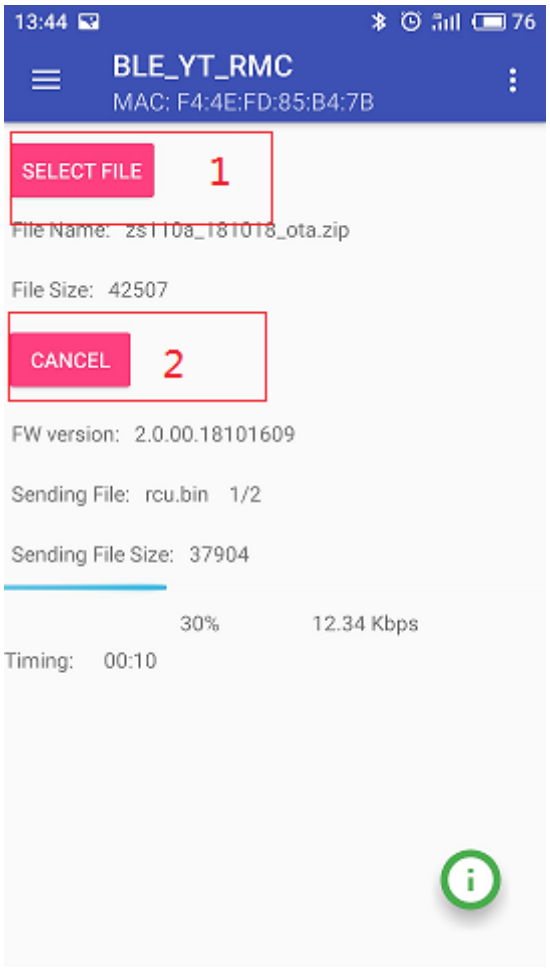


图 7.8: 功能切换选择界面

- 在功能选择页，选择 OTA，进入 OTA 功能页面，如下面所示



- OTA 页面，首先点击“SELECT FILE”选择 OTA 升级包，然后点击“UPLOAD”，就可以开始 OTA 升级了
- OTA 升级完成后，遥控器会重启

7.4 hid 遥控器 ota 升级演示

7.4.1 软硬件资源准备

- 13/16 key 遥控器或者 EVB 开发板
- android 设备

7.4.2 功能演示

以 16key 遥控器为例

- 编译并更新固件到遥控器
- 将 \samples\voice_rcu\outdir\ 路径生成的 ota 包 zs110a_***_ota.zip 拷贝到手机 sdcard 卡

- 安装 Actions VoiceBleOTA_V*.*.*.apk 到 android 设备，并打开
- 按遥控器的配对组合键 (“HOME”键 + “BACK”键) 进入配对模式，看到灯闪
- 在 android settings 中进行配对连接

图 7.9: settings 操作界面

- 在 APK 中搜索并连接蓝牙名字为“BLE_YT_RMC”的 BLE 设备，进入 OTA 功能界面界面，如图所示

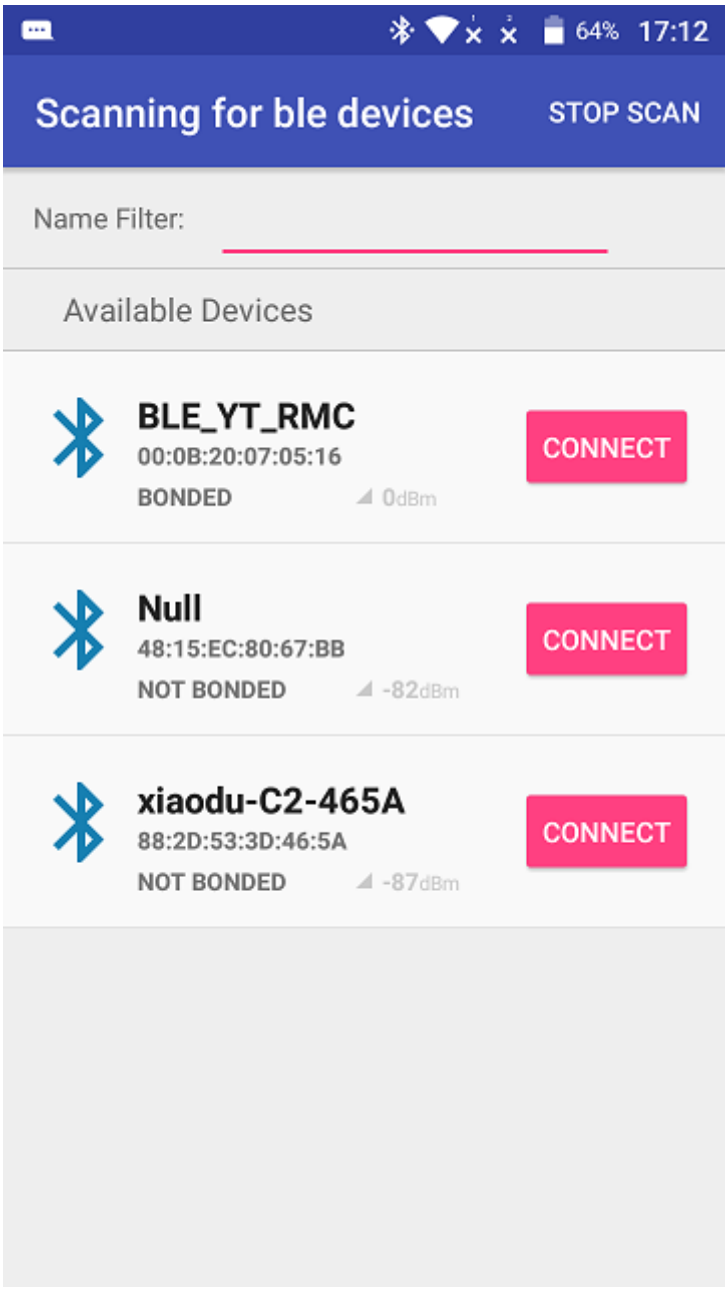


图 7.10: APK 操作界面

- OTA 页面，首先点击“SELECT FILE”选择 OTA 升级包，然后点击“UPLOAD”，就可以开始 OTA 升级了

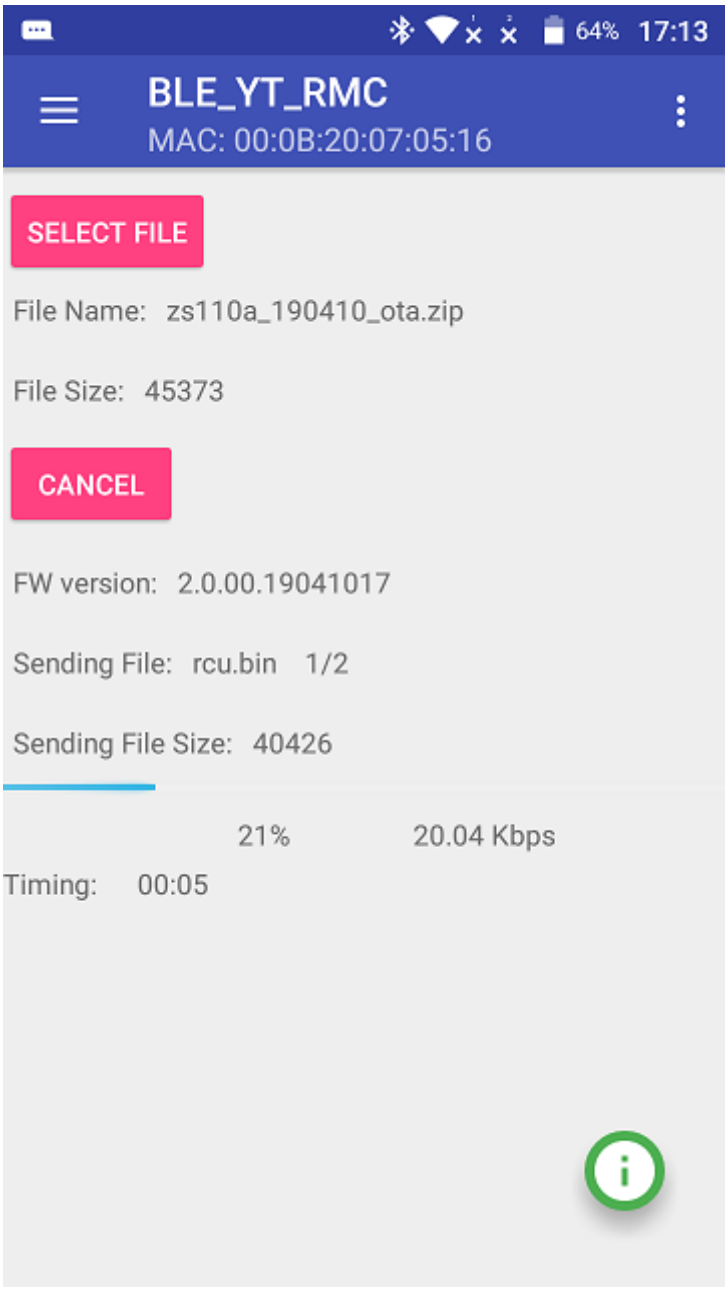


图 7.11: OTA 升级界面

- OTA 升级完成后，遥控器会重启

List of Figures

List of Tables