珠海市杰理科技股份有限公司
ZhuHai JieLi Technology Co.,Ltd

AC695N_soundbox_SDK_介绍

AC695N_soundbox_SDK_介绍 用户手册

Rev 2.0 —— 2020 年 10 月 26 日

目录

修改日志

版本	日期	描述
2.0	2020 / 10 / 26	AC695N 用户手册
更新：	● 建立初始版本 ● 定义文档格式	

Chapter 1 SDK 功能说明

1.1 编写目的

该文档主要描述 AC695x_SDK 整体功能介绍，开发人员可通过此文档快速了解 sdk 功能

1.2 模式介绍

1.AC695x_SDK 支持模式有：

（1）蓝牙模式

（2）音乐模式

（3）FM 模式

（4）PC 模式

（5）LINEIN 模式

（6）录音模式（暂支持单独 mic 录音）

（7）RTC 模式 （非真正的硬件 RTC，上电会重置）

2. 支持模式开关

```
24 //*********************************************************************************//
25 //                                 app 配置                                        //
26 //*********************************************************************************//
27 #define TCFG_APP_BT_EN                      1
28 #define TCFG_APP_MUSIC_EN                   1
29 #define TCFG_APP_LINEIN_EN                  1
30 #define TCFG_APP_FM_EN                      1
31 #define TCFG_APP_PC_EN                      1
32 #define TCFG_APP_RTC_EN                     1
33 #define TCFG_APP_RECORD_EN                  1
34 #define TCFG_APP_SPDIF_EN                   0
35 //*********************************************************************************//
36 //                              PCM_DEBUG调试配置                                  //
37 //*********************************************************************************//
38
39 //#define AUDIO_PCM_DEBUG                       //PCM串口调试，写卡通话数据
40
41 //*********************************************************************************//
42 //                                 UART配置                                        //
43 //*********************************************************************************//
44 #define TCFG_UART0_ENABLE                   ENABLE_THIS_MOUDLE                  //串口打印模块使能
45 #define TCFG_UART0_RX_PORT                  NO_CONFIG_PORT                      //串口接收脚配置（用于
46 #define TCFG_UART0_TX_PORT                  IO_PORTA_05                         //串口发送脚配置
47 #define TCFG_UART0_BAUDRATE                 1000000                             //串口波特率配置
48
49 //*********************************************************************************//
50 //                                 IIC配置                                         //
NORMAL  master  apps/soundbox/board/br23/board_ac695x_demo/board_ac695x_demo_cfg.h
```

1.3 细分功能介绍

蓝牙	高级音频	常规操作
		音量同步
		歌词 id 信息
		歌曲时间
	通话	回拨
		接听拒接
		声卡切换
		三方通话
	spp	
	hid	hid 触摸
	ble	弹窗、透传
		可支持开发 app
	tws 对箱	蓝牙播歌通话（通话主机 dac 出声音）
		解码（mp3、wma、wav）
		linein、pc、fm 等 pcm
		混响（内置 fm 不支持）
	emitter 发射器	蓝牙播歌
		解码（mp3、wma、wav）
		linein、pc、fm 等 pcm
		发射接收切换
	后台	fm、蓝牙共存
解码	格式	CVSD\MSBC\SBC\AAC\G729\MP3\WMA\FLAC\ APE\M4A\AMR\DTS\G726\MIDI
	加密播放	
音频播放	按路径播放	
	音频位置信息	文件夹总数、文件数、

		当前文件在当前文件夹序
	播放模式	单曲、循环、随机……
	文件系统	fat12、fat16、fat32、extfat
编码	格式	CVSD、MSBC\G726\MP3\ADPCM\SBC
音源输入	iis	暂不支持
	hdmi	暂不支持
	spdif	暂不支持
	linein	模拟、数字
	pc	
音效	混响	
	eq、drc\在线调试	支持 20 段
	音效、在线调试	魔音、啸叫抑制、变速、闪避、消人声 变调、高低音、电音……
fm	内置 fm 发射	蓝牙音频
		解码
		linein、pc 等
	内置 fm 收音	立体声、单声道、50k、100k 步进搜台
rtc	时钟、闹钟	非真正的硬件时钟，重新上电会重置，暂不能进低功耗
外设	双卡	
	usb	pc（massstorage、speaker、mic、hid）
	flash	支持分区、文件系统、pc 出盘符、提示音
提示音	格式	MP3、wtg\msbc\sbc\sin\mty

按键	iokey	
	adkey	
	irkey	红外
	touchkey	触摸
	rdeckey	旋码器
	slidekey	滑动电阻按键
audio	out	dac
		iis
		fm
		hdmi
		sdpif
		bt
	channel	MONO_L\左声道
		MONO_R\右声道
		LR\立体声
		MONO_LR_DIFF\单声道差分输出
		DUAL_LR_DIFF\双声道差分
		FRONT_LR_REAR_L\ 三声道单端输出 前 L+前 R+后 L（不可设置 vcmo 公共端
		FRONT_LR_REAR_R\ 三声道单端输出 前 L+前 R+后 R（可设置 vcmo 公共端）
		FRONT_LR_REAR_LR\四声道单端输出
显示	led	
	断码 lcd	
升级	U 盘	支持 uboot ui、io 保持、ufw 文件只识别后缀
	sd 卡	

	串口	
	测试盒	
	烧写器	
其他	无线充电	只在做充电仓支持
	内置充电	
	系统时钟	最高 240M
	不可屏蔽中断优先级	优先级 6 以上不开屏蔽
	dac	4 路 dac 组合
	adc	同时支持 3 路 ad

1.4 是否使用配置文件的参数配置

默认 sdk 是从配置文件读取配置来设置灯状态和提示音，如果不想从配置文件读取配置，只需修改宏 USE_CONFIG_STATUS_SETTING 为 0。

```
180 #define USE_CONFIG_BIN_FILE                 0
181
182 #define USE_CONFIG_STATUS_SETTING           1                   //状态设置，包括灯状态和提示音
183 #define USE_CONFIG_AUDIO_SETTING            USE_CONFIG_BIN_FILE //音频设置
184 #define USE_CONFIG_CHARGE_SETTING           USE_CONFIG_BIN_FILE //充电设置
185 #define USE_CONFIG_KEY_SETTING              USE_CONFIG_BIN_FILE //按键消息设置
186 #define USE_CONFIG_MIC_TYPE_SETTING         USE_CONFIG_BIN_FILE //MIC类型设置
187 #define USE_CONFIG_LOWPOWER_V_SETTING       USE_CONFIG_BIN_FILE //低电提示设置
188 #define USE_CONFIG_AUTO_OFF_SETTING         USE_CONFIG_BIN_FILE //自动关机时间设置
189
```

然后默认配置在每个 board.c 里面设置：

```
42 /*各个状态下默认的闪灯方式和提示音设置，如果USER_CFG中设置了USE_CONFIG_STATUS_SETTING为1,
43   则会从配置文件读取对应的配置来填充改结构体*/
44 STATUS_CONFIG status_config = {
45     //提示音设置
46     .tone = {
47         .charge_start  = IDEX_TONE_NONE,
48         .charge_full   = IDEX_TONE_NONE,
49         .power_on      = IDEX_TONE_POWER_ON,
50         .power_off     = IDEX_TONE_POWER_OFF,
51         .lowpower      = IDEX_TONE_LOW_POWER,
52         .max_vol       = IDEX_TONE_MAX_VOL,
53         .phone_in      = IDEX_TONE_NONE,
54         .phone_out     = IDEX_TONE_NONE,
55         .phone_activ   = IDEX_TONE_NONE,
56         .bt_init_ok    = IDEX_TONE_BT_MODE,
57         .bt_connect_ok = IDEX_TONE_BT_CONN,
58         .bt_disconnect = IDEX_TONE_BT_DISCONN,
59         .tws_connect_ok    = IDEX_TONE_TWS_CONN,
60         .tws_disconnect    = IDEX_TONE_TWS_DISCONN,
61     }
62 };
63
64 #define __this (&status_config)
```

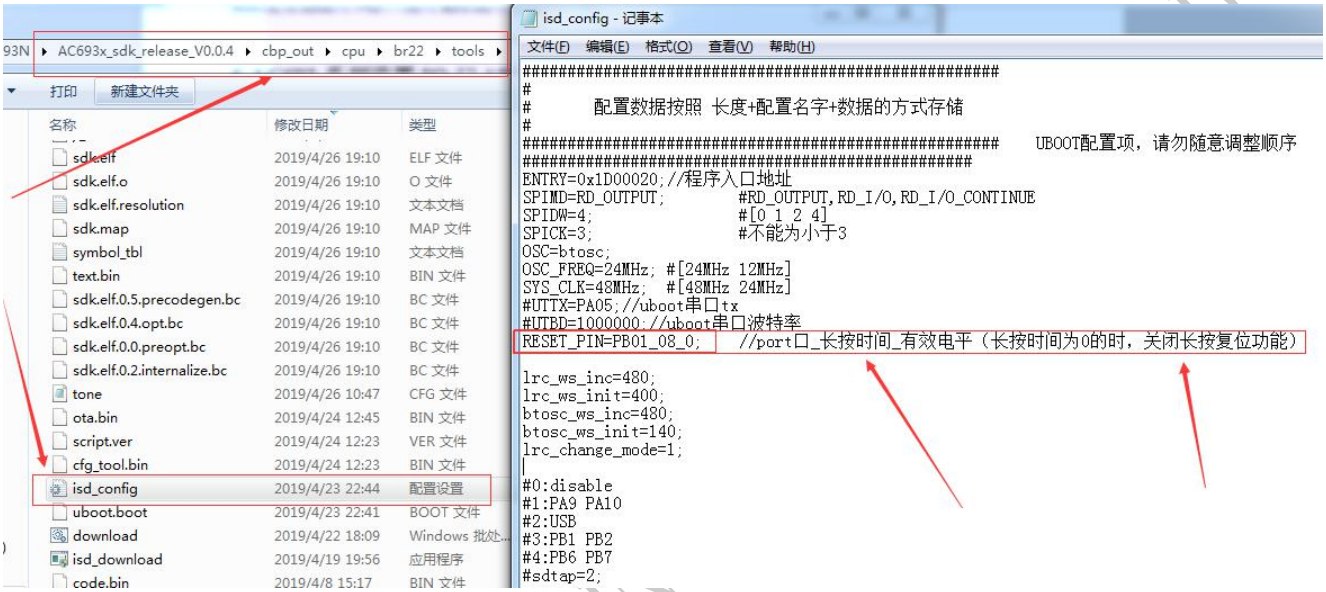
注意：如果想生成的 fw 文件能单独配置某些选项，要把对应的宏打开，如果想全部打开则设置

#define USE_CONFIG_BIN_FILE 1

即可。

1.5 长按复位设置

设置在如下路径...\cbp_out\cpu\br23\tools 的 isd_config.ini 文件设置



长按时间有 00、04、08 可选，其他值默认选 04。

1.6 功能选择注意事项

AC695x_SDK 功能比较全面独立，开发人员可以按照需求组合功能，但是仍有一些功能，因为资源和速度的限制，SDK 做了一些编译限制，例如：

```
//*********************************************************************************//
//                                 编译警告                                        //
//*********************************************************************************//
#if ((ANCS_CLIENT_EN || TRANS_DATA_EN || ((TCFG_ONLINE_TX_PORT == IO_PORT_DP)\
                && TCFG_ONLINE_ENABLE)) && (TCFG_PC_ENABLE || TCFG_UDISK_ENABLE || TCFG_SD0_PORTS == 'E'))
#error "eq online adjust enable, plaease close usb marco  and sdcard port not e!!!"
#endif// ((TRANS_DATA_EN || TCFG_ONLINE_ENABLE) && (TCFG_PC_ENABLE || TCFG_UDISK_ENABLE))

#if TCFG_UI_ENABLE
#if ((TCFG_SPI_LCD_ENABLE &&  TCFG_CODE_FLASH_ENABLE) && (TCFG_FLASH_DEV_SPI_HW_NUM == TCFG_TFT_LCD_DEV_SPI_HW_NUM))
#error "flash spi port == lcd spi port, please close one !!!"
#endif//((TCFG_SPI_LCD_ENABLE &&  TCFG_CODE_FLASH_ENABLE) && (TCFG_FLASH_DEV_SPI_HW_NUM == TCFG_TFT_LCD_DEV_SPI_HW_NUM))
#endif//TCFG_UI_ENABLE

#if((TRANS_DATA_EN + DUEROS_DMA_EN + ANCS_CLIENT_EN) > 1)
#error "they can not enable at the same time,just select one!!!"
#endif//(TRANS_DATA_EN && DUEROS_DMA_EN)

#if (TCFG_DEC2TWS_ENABLE && (TCFG_MIC_EFFECT_ENABLE ||TCFG_APP_RECORD_EN || TCFG_APP_RTC_EN ||TCFG_DRC_ENABLE))
#error "对箱支持音源转发，请关闭混响录音等功能 !!!"
#endif// (TCFG_DEC2TWS_ENABLE && (TCFG_MIC_EFFECT_ENABLE ||TCFG_APP_RECORD_EN || TCFG_APP_RTC_EN ||TCFG_DRC_ENABLE))

#if (TCFG_MIC_EFFECT_ENABLE && (TCFG_DEC_APE_ENABLE || TCFG_DEC_FLAC_ENABLE || TCFG_DEC_DTS_ENABLE))
#error "无损格式+混响暂时不支持同时打开 !!!"
#endif//(TCFG_MIC_EFFECT_ENABLE && (TCFG_DEC_APE_ENABLE || TCFG_DEC_FLAC_ENABLE || TCFG_DEC_DTS_ENABLE))

#if (TCFG_REVERB_DODGE_EN && (USER_DIGITAL_VOLUME_ADJUST_ENABLE == 0))
#error "使用闪避功能，打开自定义数字音量调节 !!!"
#endif

#if ((TCFG_NORFLASH_DEV_ENABLE || TCFG_NOR_FS_ENABLE) &&  TCFG_UI_ENABLE)
#error "引脚复用问题，使用norflash需要关闭UI ！ ！ ！"
#endif

#if ((TCFG_APP_RECORD_EN) && (TCFG_USER_TWS_ENABLE))
#error "TWS 暂不支持录音功能"
#endif

#if ((AUDIO_EQUALLOUDNESS_CONFIG) && (SYS_VOL_TYPE == VOL_TYPE_ANALOG))
#error "开启等响度 需要使用数字音量"
#endif

#if AUDIO_EQUALLOUDNESS_CONFIG
#if (TCFG_EQ_ENABLE == 0)
#error "开启等响度 需要打开EQ总使能"
#endif
#if (TCFG_EQ_ENABLE && TCFG_BT_MUSIC_EQ_ENABLE)
#error "开启等响度 需要打开EQ总使能，关闭其他模式的eq,例如关闭蓝牙播歌eq"
#endif
#endif
///<<<<所有宏定义不要在编译警告后面定义！！！！！！！！！！！！！！！！！！！！！！！！！！！
///<<<<所有宏定义不要在编译警告后面定义！！！！！！！！！！！！！！！！！！！！！！！！！！！
```

Chapter 2 VM 接口的使用

2.1 编写目的

该文档主要描述 AC695x_SDK VM 使用方法及开发中注意的一些问题，为用户进行二次开发提供参考。

2.2 详细说明

Vm 读写接口：

读接口：

int syscfg_read(u16 item_id, void *buf, u16 len);

写接口：

int syscfg_write(u16 item_id, void *buf, u16 len);

参数说明：

item_id：vm id 范围 1 ~ 60

*buf：读写指针

Len：长度

比如保存 USB MIC GAIN 和读 USB MIC GAIN 的 vm 操作：

1.确定 vm id 号

```
 4 //=============================================================================//
 5 //                          与APP CASE相关配置项[1 ~ 60]                       //
 6 //=============================================================================//
 7 #define     VM_FM_EMITTER_FREQ              1
 8 // #define  VM_FM_EMITTER_DIG_VOL           2
 9 // #define  VM_MUSIC_LAST_DEV               3
10 // #define  VM_WAKEUP_PND                   4
11 #define     VM_USB_MIC_GAIN                 5
12 #define     VM_ALARM_0                      6
13 #define     VM_ALARM_1                      7
14 #define     VM_ALARM_2                      8
15 #define     VM_ALARM_3                      9
16 #define     VM_ALARM_4                      10
17 #define     VM_ALARM_MASK                   11
18 #define     VM_ALARM_NAME_0                 12
19 #define     VM_ALARM_NAME_1                 13
20 #define     VM_ALARM_NAME_2                 14
21 #define     VM_ALARM_NAME_3                 15
22 #define     VM_ALARM_NAME_4                 16
23 #define     VM_FM_INFO                      17
24 // #define      VM_RTC_TRIM                      18
25
26 #define     CFG_RCSP_ADV_HIGH_LOW_VOL       19
27 #define     CFG_RCSP_ADV_EQ_MODE_SETTING    20
28 #define     CFG_RCSP_ADV_EQ_DATA_SETTING    21
29 #define     ADV_SEQ_RAND                    22
30 #define     CFG_RCSP_ADV_TIME_STAMP         23
31 #define     CFG_RCSP_ADV_WORK_SETTING       24
32 #define     CFG_RCSP_ADV_MIC_SETTING        25
33 #define     CFG_RCSP_ADV_LED_SETTING        26
34 #define     CFG_RCSP_ADV_KEY_SETTING        27
35 #define     CFG_RCSP_MISC_DRC_SETTING       28
36 #define     CFG_RCSP_MISC_REVERB_ON_OFF     29
37 #define     VM_ALARM_RING_NAME_0            30
38 #define     VM_ALARM_RING_NAME_1            31
39 #define     VM_ALARM_RING_NAME_2            32
NORMAL  master  ~/work/master/SDK/apps/soundbox/include/user_cfg_id.h
```

2.写 USB MIC GAIN

```
287     if (++usb_mic_gain_save_cnt >= 5) {
288         sys_hi_timer_del(usb_mic_gain_save_timer);
289         usb_mic_gain_save_timer = 0;
290         usb_mic_gain_save_cnt = 0;
291         local_irq_enable();
292         printf("USB_GAIN_SAVE\n");
293         syscfg_write(VM_USB_MIC_GAIN, &app_var.usb_mic_gain, 1);
294         return;
```

3.读 USB MIC GAIN

```
319         //// pc usb mic gain
320         s8 usb_mic_gain = -1;
321         ret = syscfg_read(VM_USB_MIC_GAIN, &usb_mic_gain, 1);
322         if (ret < 0) {
323             usb_mic_gain  = 7;
324         }
325         app_var.usb_mic_gain = usb_mic_gain;
326         log_info("usb_mic_gain: %d\n", app_var.usb_mic_gain);
```


Chapter 3 APP 模式管理

3.1 编写目的

该文档主要描述 AC695x_SDK APP 模式使用方法及开发中注意的一些问题，为用户进行二次开发提供参考。

3.2 模式任务

1. 模式任务 app_core 创建，所有的 APP 模式在此任务里执行处理

```
int main()
{
    wdt_close();

    os_init();

    setup_arch();

    board_early_init();

    task_create(app_task_handler, NULL, "app_core");

    os_start();

    local_irq_enable();

    while (1) {
        asm("idle");
    }

    return 0;
}
```

RMAL master apps/soundbox/common/init.c

2. APP 模式处理函数 app_task_loop()

```c
void app_task_loop()
{
    while (1) {
        switch (app_curr_task) {
        case APP_POWERON_TASK:
            log_info("APP_POWERON_TASK \n");
            app_poweron_task();
            break;
        case APP_POWEROFF_TASK:
            log_info("APP_POWEROFF_TASK \n");
            app_poweroff_task();
            break;
        case APP_BT_TASK:
            log_info("APP_BT_TASK \n");
            app_bt_task();
            break;
        case APP_MUSIC_TASK:
            log_info("APP_MUSIC_TASK \n");
            app_music_task();
            break;
        case APP_FM_TASK:
            log_info("APP_FM_TASK \n");
            app_fm_task();
            break;
        case APP_RECORD_TASK:
            log_info("APP_RECORD_TASK \n");
            app_record_task();
            break;
        case APP_LINEIN_TASK:
            log_info("APP_LINEIN_TASK \n");
            app_linein_task();
            break;
        case APP_RTC_TASK:
            log_info("APP_RTC_TASK \n");
            app_rtc_task();
            break;
        case APP_PC_TASK:
            log_info("APP_PC_TASK \n");
            app_pc_task();
            break;
        case APP_SPDIF_TASK:
            log_info("APP_SPDIF_TASK \n");
            app_spdif_task();
            break;
        case APP_IDLE_TASK:
            log_info("APP_IDLE_TASK \n");
            app_idle_task();
            break;
```

RMAL master apps/soundbox/app_main.c

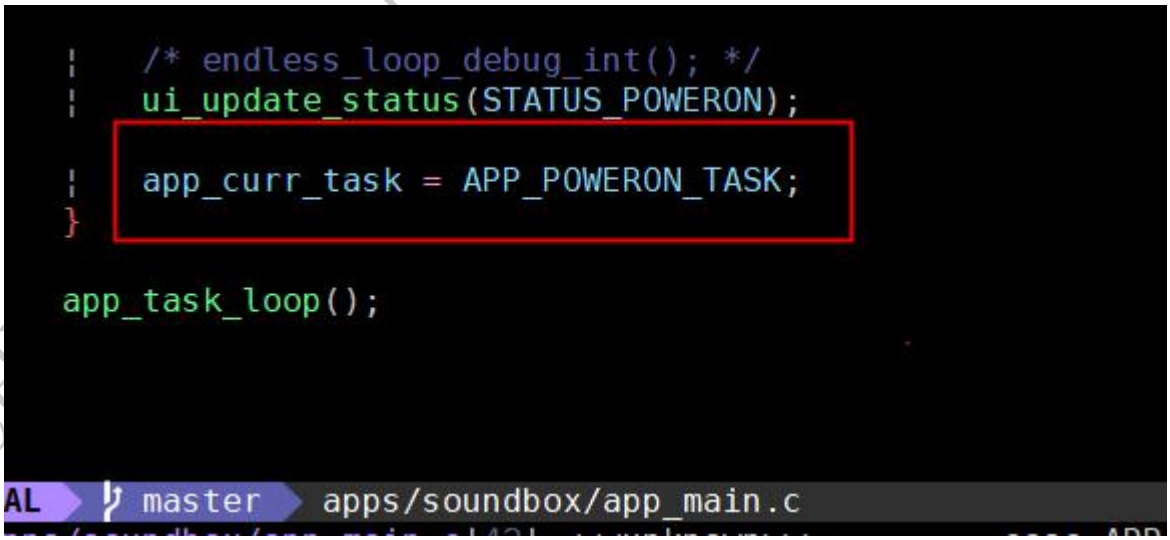
3. APP 模式值

```
enum {
    APP_POWERON_TASK  = 1,
    APP_POWEROFF_TASK = 2,
    APP_BT_TASK       = 3,
    APP_MUSIC_TASK    = 4,
    APP_FM_TASK       = 5,
    APP_RECORD_TASK   = 6,
    APP_LINEIN_TASK   = 7,
    APP_RTC_TASK      = 8,
    APP_PC_TASK       = 9,
    APP_SPDIF_TASK    = 10,
    APP_IDLE_TASK     = 11,
    APP_TASK_MAX_INDEX,
};
```

3.3 模式流程的详细介绍（以上电进入 powerno 模式为例）

1.APP 模式设置

上电设置 app 模式值



2.Poweron 模式进入

```
void app_task_loop()
{
    while (1) {
        switch (app_curr_task) {
        case APP_POWERON_TASK:
            log_info("APP_POWERON_TASK \n");
            app_poweron_task();
            break;
        case APP_POWEROFF_TASK:
            log_info("APP_POWEROFF_TASK \n");
            app_poweroff_task();
            break;
        case APP_BT_TASK:
            log_info("APP_BT_TASK \n");
            app_bt_task();
            break;
        case APP_MUSIC_TASK:
            log_info("APP_MUSIC_TASK \n");
```

3.APP 模式切换

（1）设置下一模式值

```
static int power_on_init(void)
{
    ///有些需要在开机提示完成之后再初始化的东西，  可以在这里初始化
#if (TCFG_SPI_LCD_ENABLE)
    lcd_ui_power_on();//由ui决定切换的模式
    return 0;
#endif

    printf("------>%s, %d\n", __FUNCTION__, __LINE__);
#if TCFG_APP_BT_EN
    app_task_switch_to(APP_BT_TASK);
#else
    app_task_switch_to(APP_MUSIC_TASK);
#endif

    return 0;
}
```

（2）推出 APP_MSG_SWITCH_TASK 切换消息

```
int app_task_switch_to(u8 app_task)
{
    if (app_curr_task == app_task) {
        return false;
    }

    if (!app_task_switch_check(app_task)) {
        return false;
    }

    if (!app_task_switch_exit_check(app_curr_task)) {
        return false;
    }

    printf("cur --- %x \n", app_curr_task);
    printf("new +++ %x \n", app_task);

    /* if(app_next_task) */
    /* printf("app_task_switch_to busy \n"); */
#if (defined SMART_BOX_EN) && (SMART_BOX_EN)
    extern void function_change_inform(u8 app_mode, u8 ret);
    function_change_inform(app_task, TRUE);
#endif

    app_prev_task = app_curr_task;
    app_next_task = app_task;
    app_task_put_usr_msg(APP_MSG_SWITCH_TASK, 0);

    return TRUE;

```

4.poweron 模式退出，跳出 while（1）循环

```
while (1) {
    app_task_get_msg(msg, ARRAY_SIZE(msg), 1);
    switch (msg[0]) {
    case APP_MSG_SYS_EVENT:
        if (poweron_sys_event_handler((struct sys_event *)(msg + 1)) == false) {
            app_default_event_deal((struct sys_event *)(&msg[1]));    //由common统一处理
        }
        break;
    default:
        break;
    }

    if (app_task_exitting()) {
        power_on_unint();
        return;
    }
}

master  apps/soundbox/task_manager/power_on/power_on.c
```

3.4 APP 模式管理函数接口介绍

APP 模式管理函数集中在 task_manager/app_task_switch.c

1.APP 模式配置表，这里可以配置切换模式的顺序，方案根据需求定义

```
///模式配置表，这里可以配置切换模式的顺序，方案根据需求定义
static const u8 app_task_list[] = {
#if TCFG_APP_BT_EN
    APP_BT_TASK,
#endif
#if TCFG_APP_MUSIC_EN
    APP_MUSIC_TASK,
#endif
#if TCFG_APP_FM_EN
    APP_FM_TASK,
#endif
#if TCFG_APP_RECORD_EN
    APP_RECORD_TASK,
#endif
#if TCFG_APP_LINEIN_EN
    APP_LINEIN_TASK,
#endif
#if TCFG_APP_RTC_EN
    APP_RTC_TASK,
#endif
#if TCFG_APP_PC_EN
    APP_PC_TASK,
#endif
#if TCFG_APP_SPDIF_EN
    APP_SPDIF_TASK,
#endif
};
```

2.相关函数介绍

```
//*----------------------------------------------------------------------------*/
/**@brief    模式退出检查
   @param    curr_task:当前模式
   @return   TRUE可以退出，  FALSE不可以退出
   @note
*/
/*----------------------------------------------------------------------------*/
static int app_task_switch_exit_check(u8 curr_task)
{
    int ret = false;
```

```
//*----------------------------------------------------------------------------*/
/**@brief    模式进入检查
   @param    app_task:目标模式
   @return   TRUE可以进入，  FALSE不可以进入
   @note     例如一些需要设备在线的任务(music/record/linein等),
             如果设备在线可以进入，  没有设备在线不进入可以在这里处理
*/
/*----------------------------------------------------------------------------*/
static int app_task_switch_check(u8 app_task)
{
    int ret = false;
    switch (app_task) {
#if TCFG_APP_MUSIC_EN
    case APP_MUSIC_TASK:
```

```
//*----------------------------------------------------------------------------*/
/**@brief    切换到上一个模式
   @param
   @return
   @note
*/
/*----------------------------------------------------------------------------*/
void app_task_switch_prev()
{
    int i = 0;
    int cur_index = 0;

    if (app_next_task) {
```

```
//*----------------------------------------------------------------------------*/
/**@brief    切换到下一个模式
   @param
   @return
   @note
*/
/*----------------------------------------------------------------------------*/
void app_task_switch_next()
{
    int i = 0;
    int cur_index = 0;

    if (app_next_task) {
        printf("app_task_switch_next busy \n");
```

```c
//*----------------------------------------------------------------------------*/
/**@brief    切换到指定模式
   @param    app_task:指定模式
   @return
   @note
*/
/*----------------------------------------------------------------------------*/
int app_task_switch_to(u8 app_task)
{
    if (app_curr_task == app_task) {
        return false;
    }

    if (!app_task_switch_check(app_task)) {
        return false;
    }
```

```c
//*----------------------------------------------------------------------------*/
/**@brief    跳回到原来的模式
   @param
   @return
   @note
*/
/*----------------------------------------------------------------------------*/
int app_task_switch_back()
{
    if (app_prev_task == 0) {
        return -EINVAL;
    }
    return app_task_switch_to(app_prev_task);
}
```

```c
//*----------------------------------------------------------------------------*/
/**@brief    模式切换退出检测
   @param
   @return   1:响应退出模式,  0:不响应
   @note
*/
/*----------------------------------------------------------------------------*/
u8 app_task_exitting()//
{
    struct sys_event clear_key_event = {.type =  SYS_KEY_EVENT, .arg = (void *)DEVICE_EVENT_FROM_KEY};
    if (app_next_task != 0) {
        app_curr_task = app_next_task;
```

```c
//*----------------------------------------------------------------------------*/
/**@brief    获取当前模式
   @param
   @return   当前模式id
   @note
*/
/*----------------------------------------------------------------------------*/
u8 app_get_curr_task()
{
    return app_curr_task;
}
```

```
//*----------------------------------------------------------------------------*/
/**@brief    通过指定id检查是否是当前模式
   @param
   @return   true:是当前模式，false:不是当前模式
   @note
*/
/*----------------------------------------------------------------------------*/
u8 app_check_curr_task(u8 app)
{
    if (app == app_curr_task) {
        return true;
    }
    return false;
}
```

3.5 增加 APP 模式例子（以 music 为例）

1.在 app_task.h 中增加模式 id（以 music 为例）

```
enum {
    APP_POWERON_TASK  = 1,
    APP_POWEROFF_TASK = 2,
    APP_BT_TASK       = 3,
    APP_MUSIC_TASK    = 4,
    APP_FM_TASK       = 5,
    APP_RECORD_TASK   = 6,
    APP_LINEIN_TASK   = 7,
    APP_RTC_TASK      = 8,
    APP_PC_TASK       = 9,
    APP_SPDIF_TASK    = 10,
    APP_IDLE_TASK     = 11,
    APP_TASK_MAX_INDEX,
};
```

2.将新加的模式 id 加入到 app_task_switch.c 中模式配置表 app_task_list

```
///模式配置表，这里可以配置切换模式的顺序，方案根据需求定义
static const u8 app_task_list[] = {
#if TCFG_APP_BT_EN
    APP_BT_TASK,
#endif
#if TCFG_APP_MUSIC_EN
    APP_MUSIC_TASK,
#endif
#if TCFG_APP_FM_EN
    APP_FM_TASK,
#endif
#if TCFG_APP_RECORD_EN
    APP_RECORD_TASK,
#endif
#if TCFG_APP_LINEIN_EN
    APP_LINEIN_TASK,
#endif
#if TCFG_APP_RTC_EN
    APP_RTC_TASK,
#endif
#if TCFG_APP_PC_EN
    APP_PC_TASK,
#endif
#if TCFG_APP_SPDIF_EN
    APP_SPDIF_TASK,
#endif
};
```

3.参考 task_key.c 中参考添加模式按键转换表（ad、io、ir 等）

4.在 task_manager 中添加对应的模式目录（及对应的头文件目录）

5.实现模式相关接口（参考已有模式，以下以 music 为例进行说明）

① 实现以下基础必要接口：

void app_music_task()

int music_app_check(void)

static int music_sys_event_handler(struct sys_event *event)

static int music_key_event_opr(struct sys_event *event)

② 模式主循环内完成以下基础操作（app_music_task）

获取消息

响应消息及事件

响应模式内部消息及事件

响应公共消息及事件

```
void app_music_task()
{
    int res;
    int msg[32];
    music_task_start();

    int err =  tone_play_with_callback_by_name(tone_table[IDEX_TONE_MUSIC], 1, music_tone_play_end_callback, (void *)IDEX_TONE_MUSIC);
    if (err) {
        music_player_play_start();
    }


    while (1) {
        app_task_get_msg(msg, ARRAY_SIZE(msg), 1);
        switch (msg[0]) {
        case APP_MSG_SYS_EVENT:
            if (music_sys_event_handler((struct sys_event *)(&msg[1])) == false) {
                app_default_event_deal((struct sys_event *)(&msg[1]));
            }
            break;
        default:
            break;
        }
        if (app_task_exitting()) {
            music_task_close();
            return;
        }
    }
}
```

初始化，非必要，根据具体情景定义

播放模式提示音，非必要，如果要的话参考music.c的案例实现

模式主循环

获取消息

处理系统case消息

模式内部消息拦截，如果不拦截，给公共消息处理响应

必要，响应公共消息处理

必要，模式退出检测处理

模式退出内部释放处理，非必要，如果有初始化，就一定要注意释放操作

③ 在 app_main.c 中调用对应的模式主循环接口（app_music_task）

```
38 void app_task_loop()
39 {
40     while (1) {
41         switch (app_curr_task) {
42         case APP_POWERON_TASK:
43             log_info("APP_POWERON_TASK \n");
44             app_poweron_task();
45             break;
46         case APP_POWEROFF_TASK:
47             log_info("APP_POWEROFF_TASK \n");
48             app_poweroff_task();
49             break;
50         case APP_BT_TASK:
51             log_info("APP_BT_TASK \n");
52             app_bt_task();
53             break;
54         case APP_MUSIC_TASK:
55             log_info("APP_MUSIC_TASK \n");
56             app_music_task();
57             break;
58         case APP_FM_TASK:
59             log_info("APP_FM_TASK \n");
60             app_fm_task();
61             break;
62         case APP_RECORD_TASK:
63             log_info("APP_RECORD_TASK \n");
64             app_record_task();
65             break;
NORMAL  master  apps/soundbox/app_main.c
```

④ app_check 接口的实现（music 为例）

app_check 其实是在切换模式的时候，是否满足条件进入该模式，music 模式进入条件是判断是否有可以播放的设备在线，故接口实现如下：

```
634 int music_app_check(void)
635 {
636     if (dev_manager_get_total(1)) {
637         return true;
638     }
639     return false;
640 }
```

⑤ 在 app_task_switch_check 调用 app_check（music 为例）

```
107 static int app_task_switch_check(u8 app_task)
108 {
109     int ret = false;
110     switch (app_task) {
111 #if TCFG_APP_MUSIC_EN
112     case APP_MUSIC_TASK:
113         ret = music_app_check();
114         break;
115 #endif
116 #if TCFG_APP_LINEIN_EN
117     case APP_LINEIN_TASK:
118         ret = linein_app_check();
119         break;
120 #endif
```

Chapter 4　APP 消息管理

4.1 系统事件管理

1、系统事件的分类

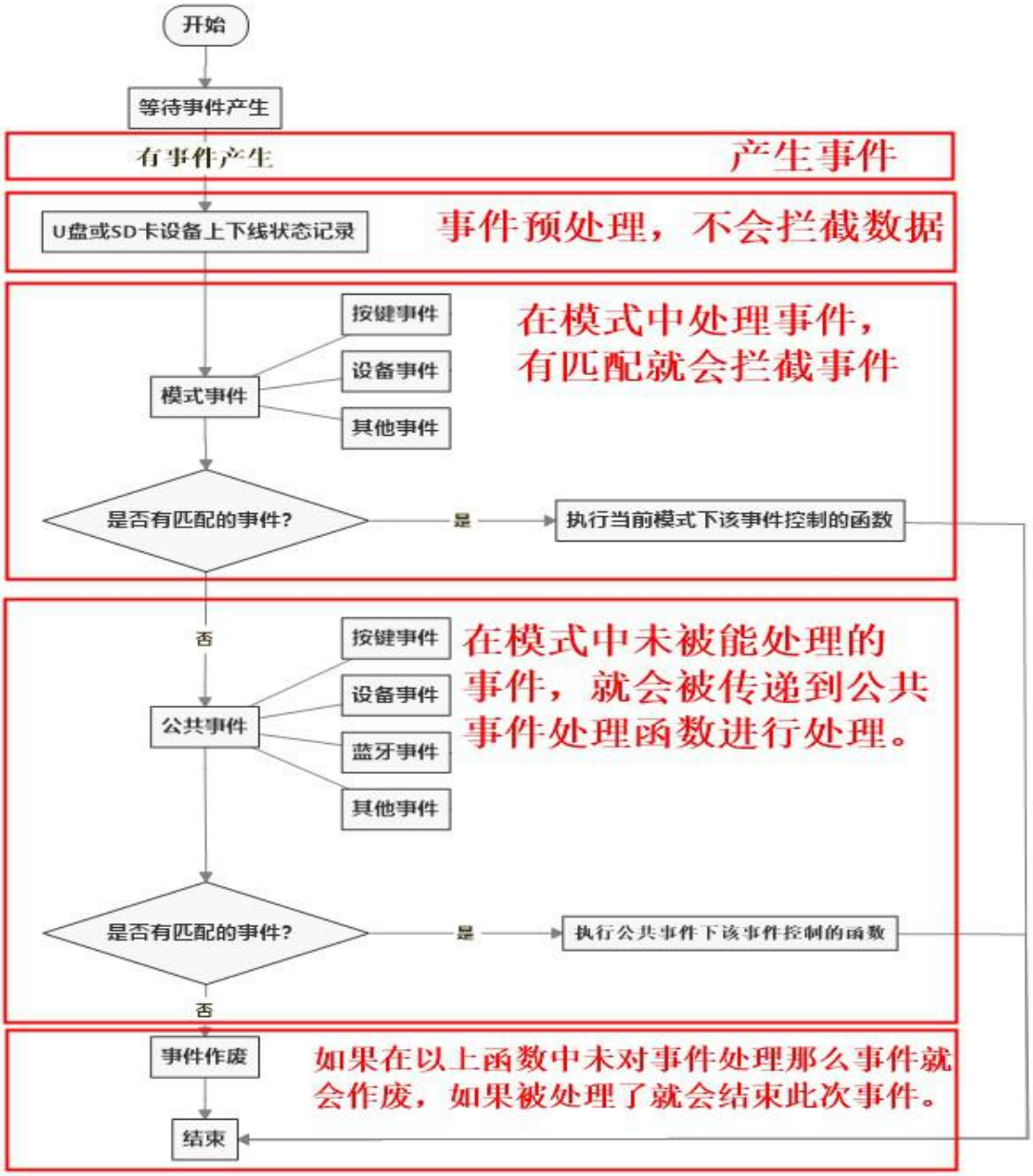
系统事件分为系统按键事件、系统设备事件以及系统蓝牙事件，为了更好管理，还把这三大事件分为模式里的系统事件类和每个模式都有的公共系统事件类。流程图如下：

特别说明：

①系统产生事件后，会先到模式里的系统事件查询是否有匹配的事件，如有，则执行相应的函数并且不再执行公共系统事件类，然后结束事件，如无，则到公共系统事件查询是否有匹配的事件，如有，则执行相应的函数，然后结束事件，如无，则视为事件作废，结束事件。

②蓝牙模式的设备事件如果不做处理，那么会把事件丢到公共事件中的公共蓝牙事件去处理，而不是丢到公共事件的公共设备事件去处理。

以下为系统事件的管理机制：



4.2 提供的 APP 消息处理函数说明

1.提供的消息函数接口

```
//app自定义消息发送接口
int app_task_put_usr_msg(int msg, int arg_num, ...);

//app消息获取接口(block参数为0表示内部pend，1直接返回)
void app_task_get_msg(int *msg, int msg_size, int block);

//app按键消息发送接口
int app_task_put_key_msg(int msg, int value);

//app清理按键消息接口
void app_task_clear_key_msg();
```

2.消息处理机制

（1） 获取消息

```
void app_poweron_task()
{
    int msg[32];

    UI_SHOW_MENU(MENU_POWER_UP, 0, 0, NULL);

    int err =  tone_play_with_callback_by_name(tone_table[IDEX_TONE_POWER_ON], 1, tone_play_end_callback, (void *)IDEX_TONE_POWER_ON);
    if (err) { //提示音没有,播放失败，直接init流程
        power_on_init();
    }


    while (1) {
        app_task_get_msg(msg, ARRAY_SIZE(msg), 1);
        switch (msg[0]) {
        case APP_MSG_SYS_EVENT:
            if (poweron_sys_event_handler((struct sys_event *)(msg + 1)) == false) {
                app_default_event_deal((struct sys_event *)(&msg[1]));    //由common统一处理
            }
            break;
        default:
            break;
        }

        if (app_task_exitting()) {
            power_on_unint();
            return;
        }
    }

}
```

（2） 推出 app 按键处理消息（非物理按键触发）

```
            break;
        }
        if (logo && (0 == memcmp(logo, evt_logo, str_len))) {
            ///相同的设备才响应
            if (music_player_get_playing_breakpoint(breakpoint, 1) == true) {
                breakpoint_vm_write(breakpoint, logo);
            }
            ///停止解码,防止设备掉线后还继续使用
            music_player_stop(1);
            ///重新选择活动设备播放
            app_task_put_key_msg(KEY_MUSIC_PLAYER_START, 0);//卸载了设备再执行
            log_i("KEY_MUSIC_PLAYER_START_AFTER_UMOUNT\n");
        }
    } else {
        ///新设备上线，先记录当前设备断点， 然后播放活动设备
        if (music_player_get_playing_breakpoint(breakpoint, 1) == true) {
            breakpoint_vm_write(breakpoint, logo);
        }
        ///停止解码，播放新活动设备
        music_player_stop(1);
        app_task_put_key_msg(KEY_MUSIC_PLAYER_START, 0);
        log_i("KEY_MUSIC_PLAYER_START_AFTER_MOUNT\n");
        //先挂载了设备再执行
    }
    break;
default://switch((u32)event->arg)
    break;
```

3. app 自定义消息发送接口

（1）消息值定义

```
enum {
    APP_MSG_SYS_EVENT = Q_EVENT,

    /* 用户自定义消息 */
    APP_MSG_SWITCH_TASK = Q_USER + 1,
    APP_MSG_USER        = Q_USER + 2,


    // 添加消息值
    //  eg:
    // APP_MSG_TEST         = Q_USER + 3,
    //


};


//切换到前一个有效模式
void app_task_switch_prev();
//切换到下一个有效模式
```

（2）消息处理

以蓝牙模式为例（如果消息其他模式都需要处理，就要在需要处理的模式都添加）

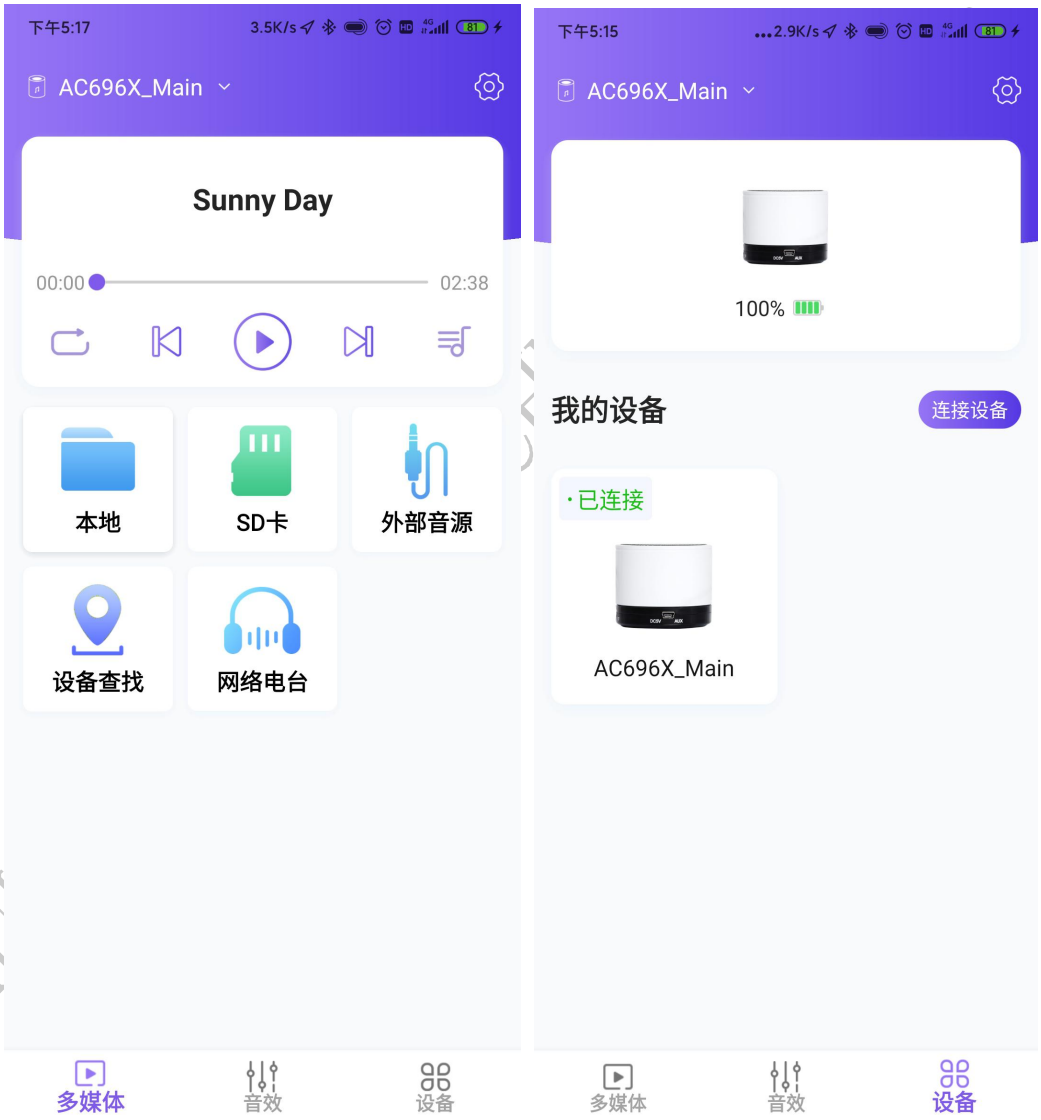
```
void app_bt_task()
{
    int res;
    int msg[32];
    ui_update_status(STATUS_EXIT_LOWPOWER);

    if (!__this->cmd_flag) { //蓝牙后台拉回蓝牙模式不播放提示音
    ¦   tone_play_by_path(tone_table[IDEX_TONE_BT_MODE], 1);
    }

    bt_task_start();

    while (1) {
    ¦   app_task_get_msg(msg, ARRAY_SIZE(msg), 1);

    ¦   switch (msg[0]) {
    ¦   case APP_MSG_SYS_EVENT:
    ¦   ¦   if (bt_sys_event_handler((struct sys_event *)(msg + 1)) == false) {
    ¦   ¦   ¦   app_default_event_deal((struct sys_event *)(&msg[1]));
    ¦   ¦   }
    ¦   ¦   break;

    ¦   /* case APP_MSG_TEST:            */
    ¦   /*     // deal */
    ¦   /*     break; */

    ¦   default:
    ¦   ¦   break;
    ¦   }

    ¦   if (app_task_exitting()) {
    ¦   ¦   bt_task_close();
    ¦   ¦   __this->wait_exit = 1;
    ¦   }

    ¦   if (__this->wait_exit) { //开始退出
    ¦   ¦   if (!__this->exiting) { ///等待蓝牙退出处理完成
    ¦   ¦   ¦   return;
    ¦   ¦   }
    ¦   }
    }
}
```


Chapter 5 杰理之家 APP

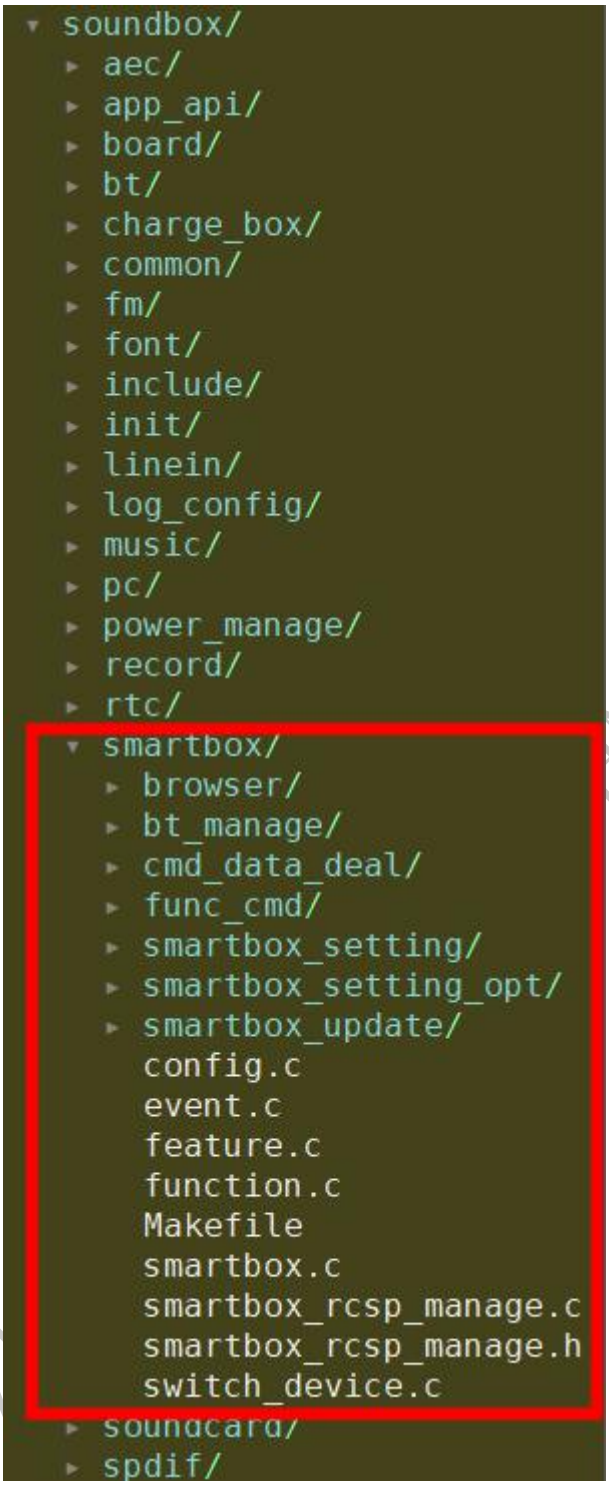
5.1 编写目的

该章节主要描述 AC695N 系列芯片杰理之家 APP 介绍，为用户进行二次开发提供参考

5.2 APP 介绍



5.3 文件目录说明



协议代码：soundbox/smartbox

目录说明：

-browser

目录浏览相关目录，存放目录浏览相关代码。

- bt_manage

透传接口目录，存放透传相关代码。

- cmd_data_deal

rcsp 协议目录，存放 rcsp 交互相关代码。

- func_cmd

各种模式（如蓝牙模式、音乐模式）功能目录。

- smartbox_setting

功能设置目录，存放各种功能具体实现的代码。

- smartbox_setting_opt

功能设置操作目录，存放公共功能设置相关的代码。

- smartbox_update

OTA 升级目录，存放升级相关的代码。

-其他

其他文件是 SMARTBOX 相关的公共操作。

5.4 功能开关、配置

在 soundbox/board/br23/board_ac695x_smartbox/board_ac695x_smartbox.h 里

#define CONFIG_APP_BT_ENABLE （功能总开关）

#define SMART_BOX_EN 1

```
//*********************************************************************************//
//                                  AI配置                                         //
//*********************************************************************************//
#define CONFIG_APP_BT_ENABLE // AI功能、流程总开关

#ifdef CONFIG_APP_BT_ENABLE
#define    TRANS_DATA_EN            0
#define    SMART_BOX_EN             1
#else
#define    TRANS_DATA_EN            0
#define    SMART_BOX_EN             0
#endif

#if (SMART_BOX_EN)                      //rcsp需要打开ble
#define CONFIG_DOUBLE_BANK_ENABLE    0
#define RCSP_UPDATE_EN               1     //是否支持rcsp升级
#define OTA_TWS_SAME_TIME_ENABLE     0     //是否支持TWS同步升级
#define UPDATE_MD5_ENABLE            1     //升级是否支持MD5校验
#define RCSP_FILE_OPT                1
#define JL_EARPHONE_APP_EN           1
#define TCFG_LFN_EN                  1
#define TCFG_BS_DEV_PATH_EN          1
#else
#define OTA_TWS_SAME_TIME_ENABLE     0
#define RCSP_UPDATE_EN               0
#define UPDATE_MD5_ENABLE            0     //升级是否支持MD5校验
#define RCSP_FILE_OPT                0
#define JL_EARPHONE_APP_EN           0
#endif

#if RCSP_UPDATE_EN
#define APP_UPDATE_EN                0    //需要使用APP升级的话要把该宏打开
#else
#define APP_UPDATE_EN                0    //客户如需要开发自己的app升级协议需要把这个宏打开,并提供升级需要的read\seek等接口,具体请参照说明文档
#endif
```

在 soundbox/board/br23/board_ac695x_smartbox/board_ac695x_smartbox.h 里

关于具体支持模式使能相关的宏设置

```
//*********************************************************************************//
//                                  app 配置                                       //
//*********************************************************************************//
#define TCFG_APP_BT_EN                      1
#define TCFG_APP_MUSIC_EN                   1
#define TCFG_APP_LINEIN_EN                  1
#define TCFG_APP_FM_EN                      0
#define TCFG_APP_PC_EN                      0
#define TCFG_APP_RTC_EN                     0
#define TCFG_APP_RECORD_EN                  0
#define TCFG_APP_SPDIF_EN                   0
#define TCFG_APP_RTC_SIMULATE_EN            0
```

关于 RCSP 功能具体模块设置使能相关的宏设置

在 soundbox/smartbox/bt_manage/bt_trans_data/le_smartbox_module.h 里

```
//rcsp功能模块使能
#define RCSP_ADV_NAME_SET_ENABLE        1
#define RCSP_ADV_KEY_SET_ENABLE         0
#define RCSP_ADV_LED_SET_ENABLE         1
#define RCSP_ADV_MIC_SET_ENABLE         0
#define RCSP_ADV_WORK_SET_ENABLE        0
#define RCSP_ADV_EQ_SET_ENABLE          1
#define RCSP_ADV_MUSIC_INFO_ENABLE      1
#define RCSP_ADV_HIGH_LOW_SET           1
#define RCSP_ADV_PRODUCT_MSG_ENABLE     1
#define RCSP_ADV_FIND_DEVICE_ENABLE     1
```

5.5 命令、数据 API 介绍

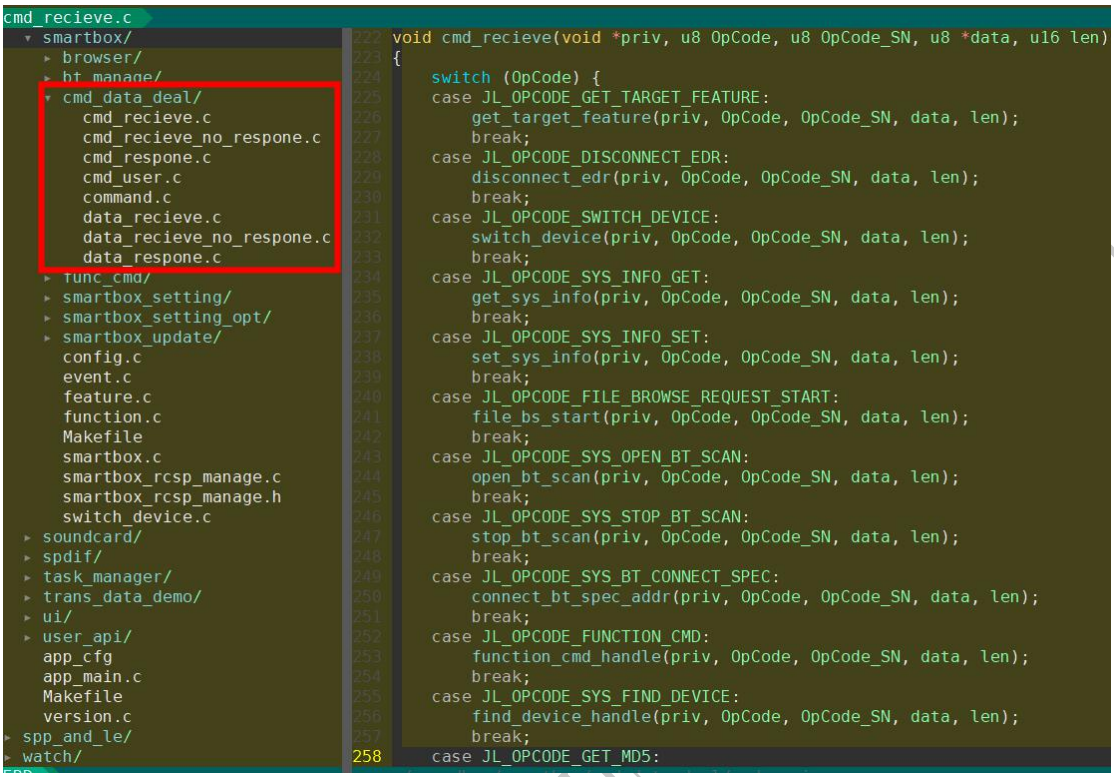
5.5.1 已有命令

```
/***************************************/
/*      注意：不能在这个枚举里加代码     */
/*      注意：不能在这个枚举里加代码     */
/*      注意：不能在这个枚举里加代码     */
/***************************************/
enum {
    JL_OPCODE_DATA = 0x01,
    JL_OPCODE_GET_TARGET_FEATURE_MAP = 0x02,
    JL_OPCODE_GET_TARGET_FEATURE = 0x03,
    // JL_OPCODE_SWITCH_DEVICE = 0x04,
    JL_OPCODE_DISCONNECT_EDR = 0x06,
    JL_OPCODE_SYS_INFO_GET = 0x07,
    JL_OPCODE_SYS_INFO_SET = 0x08,
    JL_OPCODE_SYS_INFO_AUTO_UPDATE = 0x09,
    JL_OPCODE_CALL_REQUEST = 0xa,
    JL_OPCODE_SWITCH_DEVICE = 0x0b,
    JL_OPCODE_FILE_BROWSE_REQUEST_START = 0x0C,
    JL_OPCODE_FILE_BROWSE_REQUEST_STOP = 0x0D,
    JL_OPCODE_FUNCTION_CMD = 0x0E,

    JL_OPCODE_SYS_OPEN_BT_SCAN = 0x12,
    JL_OPCODE_SYS_UPDATE_BT_STATUS = 0X13,
    JL_OPCODE_SYS_STOP_BT_SCAN = 0X14,
    JL_OPCODE_SYS_BT_CONNECT_SPEC = 0X15,
    JL_OPCODE_SYS_EMITTER_BT_CONNECT_STATUS = 0X20, // 原来0x19
    JL_OPCODE_SYS_FIND_DEVICE = 0X19,

    JL_OPCODE_GET_MD5 = 0xD4,
    JL_OPCODE_LOW_LATENCY_PARAM = 0xD5,

    JL_OPCODE_CUSTOMER_USER = 0xFF,
};
/***************************************/
/***************************************/
```

其中 JL_OPCODE_CUSTOMER_USER = 0xFF 用于自定义命令扩展

注意： 不能用其他非 0xFF 的命令号来自定义命令

5.5.2 小机和手机交互的处理



小机接收到的命令，统一在 cmd_recieve.c 的 cmd_recieve 函数中（对已有命令）进行处理。其中较为常用的有 JL_OPCODE_GET_TARGET_FEATURE 、 JL_OPCODE_SYS_INFO_GET 、 JL_OPCODE_SYS_INFO_SET 和 JL_OPCODE_FUNCTION_CMD 这几个命令。

JL_OPCODE_GET_TARGET_FEATURE 命令是用于设置 SMARTBOX 的特性。对应会调用到 soundbox/smartbox/feature.c 的 target_feature_mask_get_tab 列表中所有的函数。

```
static const attr_get_func target_feature_mask_get_tab[FEATURE_ATTR_TYPE_MAX] = {
    [FEATURE_ATTR_TYPE_PROTOCOL_VERSION  ] = target_feature_attr_protocol_version,
    [FEATURE_ATTR_TYPE_SYS_INFO          ] = target_feature_attr_sys_info,
    [FEATURE_ATTR_TYPE_EDR_ADDR          ] = target_feature_attr_edr_addr,
    [FEATURE_ATTR_TYPE_PLATFORM          ] = target_feature_attr_platform,
    [FEATURE_ATTR_TYPE_FUNCTION_INFO     ] = target_feature_function_info,
    [FEATURE_ATTR_TYPE_DEV_VERSION       ] = target_feature_dev_version,
    [FEATURE_ATTR_TYPE_SDK_TYPE          ] = target_feature_sdk_type,
    [FEATURE_ATTR_TYPE_UBOOT_VERSION     ] = target_feature_uboot_version,
    [FEATURE_ATTR_TYPE_DOUBLE_PARITION   ] = target_feature_ota_double_parition,
    [FEATURE_ATTR_TYPE_UPDATE_STATUS     ] = target_feature_ota_update_status,
    [FEATURE_ATTR_TYPE_DEV_VID_PID       ] = target_feature_pid_vid,
    [FEATURE_ATTR_TYPE_DEV_AUTHKEY       ] = target_feature_authkey,
    [FEATURE_ATTR_TYPE_DEV_PROCODE       ] = target_feature_procode,
    [FEATURE_ATTR_TYPE_DEV_MAX_MTU       ] = target_feature_mtu,
    [FEATURE_ATTR_TYPE_CONNECT_BLE_ONLY  ] = target_feature_ble_only,
    [FEATURE_ATTR_TYPE_BT_EMITTER_INFO   ] = target_feature_bt_emitter_info,
    [FEATURE_ATTR_TYPE_MD5_GAME_SUPPORT  ] = target_feature_md5_game_support,
};

u32 target_feature_parse_packet(void *priv, u8 *buf, u16 buf_size, u32 mask)
{
    printf("target_feature_parse_packet, mask = %x\n", mask);
    return attr_get(priv, buf, buf_size, target_feature_mask_get_tab, FEATURE_ATTR_TYPE_MAX, mask);
}
#endif//SMART_BOX_EN

NORMAL  master  ~/BR22_SDK/SDK/apps/soundbox/smartbox/feature.c
```

JL_OPCODE_SYS_INFO_GET 命令是用于获取 SMARTBOX 系统信息的命令。对应会调用到 soundbox/smartbox/function.c 的 target_common_function_get_tab 列表中的所有函数。

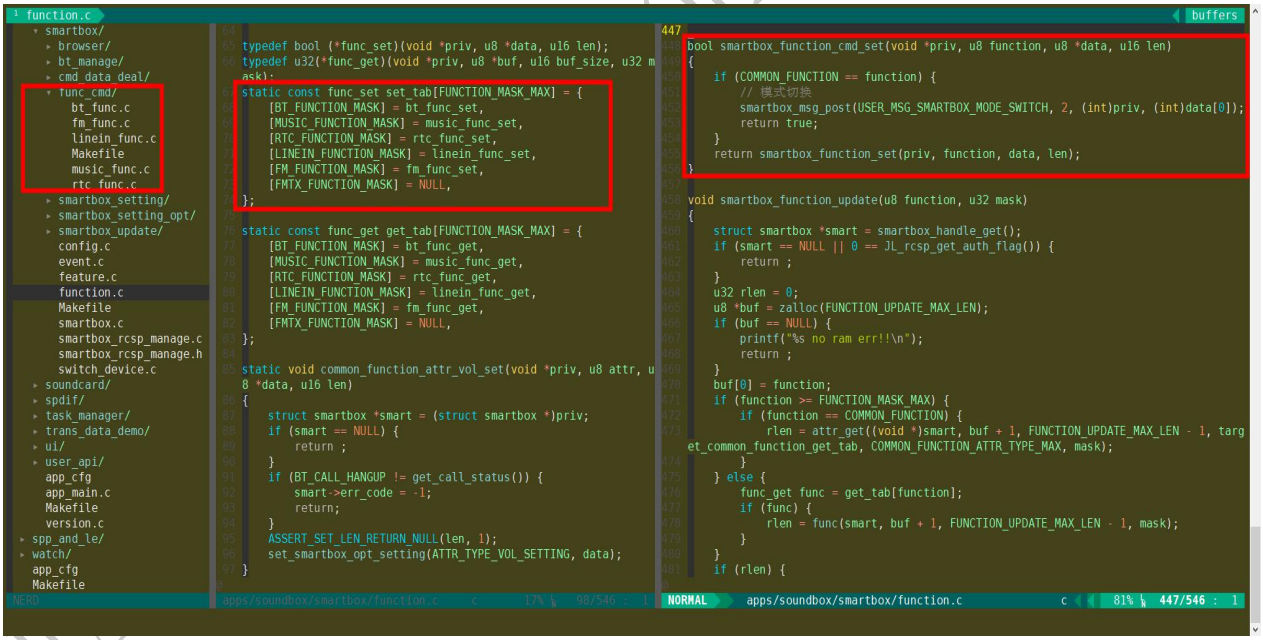
```
static const attr_get_func target_common_function_get_tab[COMMON_FUNCTION_ATTR_TYPE_MAX] = {
    [COMMON_FUNCTION_ATTR_TYPE_BATTERY                    ] = common_function_attr_battery_get,
    [COMMON_FUNCTION_ATTR_TYPE_VOL                        ] = common_function_attr_vol_get,
    [COMMON_FUNCTION_ATTR_TYPE_DEV_INFO                   ] = common_function_attr_dev_info_get,
    [COMMON_FUNCTION_ATTR_TYPE_ERROR_STATS                ] = NULL,
    [COMMON_FUNCTION_ATTR_TYPE_EQ_INFO                    ] = common_function_attr_eq_param_get,
    [COMMON_FUNCTION_ATTR_TYPE_BS_FILE_TYPE               ] = common_function_attr_bs_file_type,
    [COMMON_FUNCTION_ATTR_TYPE_FUNCTION_MODE              ] = common_function_attr_function_mode_get,
    [COMMON_FUNCTION_ATTR_TYPE_FMTX_FREQ                  ] = common_function_attr_fmtx_freq_get,
    [COMMON_FUNCTION_ATTR_TYPE_BT_EMITTER_SW              ] = common_function_attr_bt_emitter_sw_get,
    [COMMON_FUNCTION_ATTR_TYPE_BT_EMITTER_CONNECT_STATES  ] = common_function_attr_bt_emitter_connect_state_get,
    [COMMON_FUNCTION_ATTR_TYPE_HIGH_LOW_SET               ] = common_function_attr_high_low_get,
    [COMMON_FUNCTION_ATTR_TYPE_PRE_FETCH_ALL_EQ_INFO      ] = common_function_attr_pre_fetch_all_eq_info_get,
    [COMMON_FUNCTION_ATTR_TYPE_PHONE_SCO_STATE_INFO       ] = common_function_attr_phone_sco_state_info_get,
};


static bool smartbox_common_function_set(void *priv, u8 *data, u16 len)
{
    printf("smartbox_common_function_set\n");
    struct smartbox *smart = (struct smartbox *)priv;
    if (smart == NULL) {
        return false;
    }
    put_buf(data, len);
    attr_set(priv, data, len, common_function_set_tab, COMMON_FUNCTION_ATTR_TYPE_MAX);

NORMAL  master  ~/BR22_SDK/SDK/apps/soundbox/smartbox/function.c                    c  utf
```

JL_OPCODE_SYS_INFO_SET 命令是用与设置 SMARTBOX 系统信息的命令。对应会调用到 soundbox/smartbox/function.c 的 common_function_set_tab 列表中的所有函数。

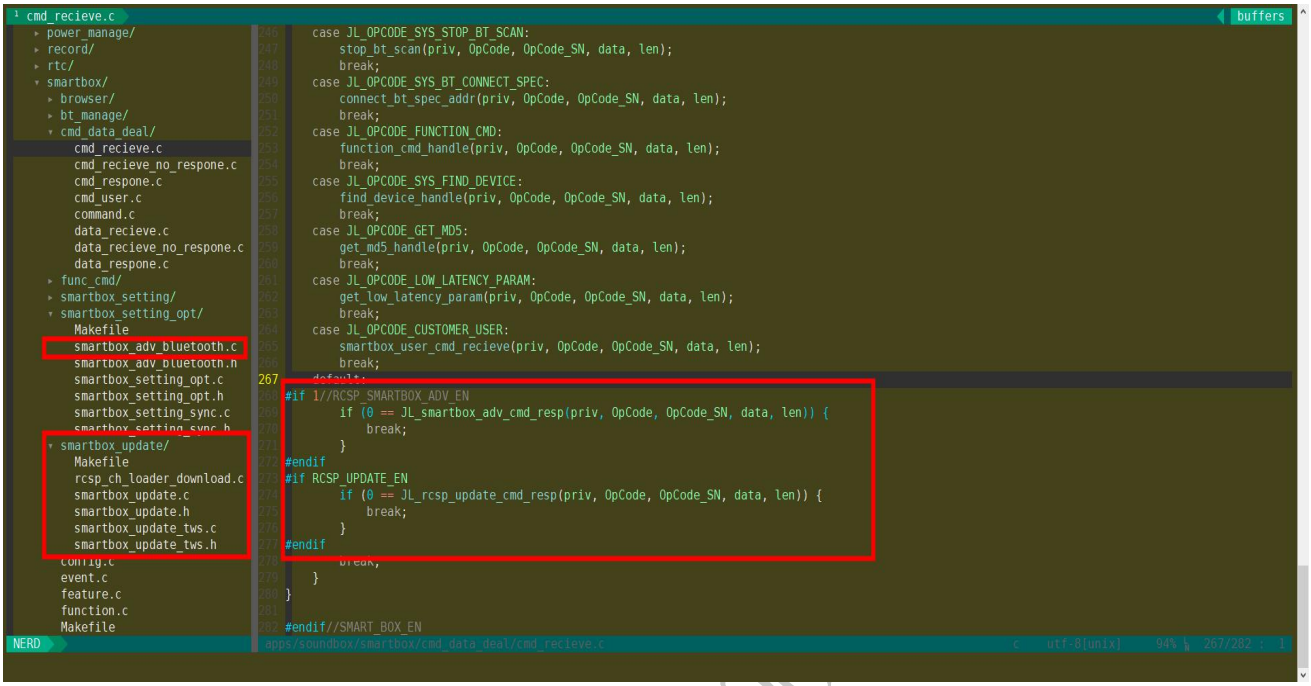
```c
static const attr_set_func common_function_set_tab[COMMON_FUNCTION_ATTR_TYPE_MAX] = {
    [COMMON_FUNCTION_ATTR_TYPE_BATTERY                      ] = NULL,
    [COMMON_FUNCTION_ATTR_TYPE_VOL                          ] = common_function_attr_vol_set,
    [COMMON_FUNCTION_ATTR_TYPE_DEV_INFO                     ] = NULL,
    [COMMON_FUNCTION_ATTR_TYPE_ERROR_STATS                  ] = NULL,
    [COMMON_FUNCTION_ATTR_TYPE_EQ_INFO                      ] = common_function_attr_eq_set,//NULL,
    [COMMON_FUNCTION_ATTR_TYPE_BS_FILE_TYPE                 ] = NULL,
    [COMMON_FUNCTION_ATTR_TYPE_FUNCTION_MODE                ] = NULL,
    [COMMON_FUNCTION_ATTR_TYPE_FMTX_FREQ                    ] = common_function_attr_fmtx_freq_set,
    [COMMON_FUNCTION_ATTR_TYPE_BT_EMITTER_SW                ] = common_function_attr_bt_emitter_sw_set,
    [COMMON_FUNCTION_ATTR_TYPE_BT_EMITTER_CONNECT_STATES    ] = common_function_attr_bt_emitter_connect_state_set,
    [COMMON_FUNCTION_ATTR_TYPE_HIGH_LOW_SET                 ] = common_function_attr_high_low_set,
    [COMMON_FUNCTION_ATTR_TYPE_PRE_FETCH_ALL_EQ_INFO        ] = NULL,
    [COMMON_FUNCTION_ATTR_TYPE_PHONE_SCO_STATE_INFO         ] = NULL,
};


static u32 common_function_attr_battery_get(void *priv, u8 attr, u8 *buf, u16 buf_size, u32 offset)
{
    u32 rlen = 0;
    extern u8 get_vbat_percent(void);
    u8 vbat = get_vbat_percent();
    rlen = add_one_attr(buf, buf_size, offset, attr, &vbat, sizeof(vbat));
    return rlen;
}
static u32 common_function_attr_vol_get(void *priv, u8 attr, u8 *buf, u16 buf_size, u32 offset)
NORMAL  master  ~/BR22_SDK/SDK/apps/soundbox/smartbox/function.c                    c  utf-8[unix
```

JL_OPCODE_FUNCTION_CMD 命令是用于切换模式和每个模式具体的设置行为交互的指令。

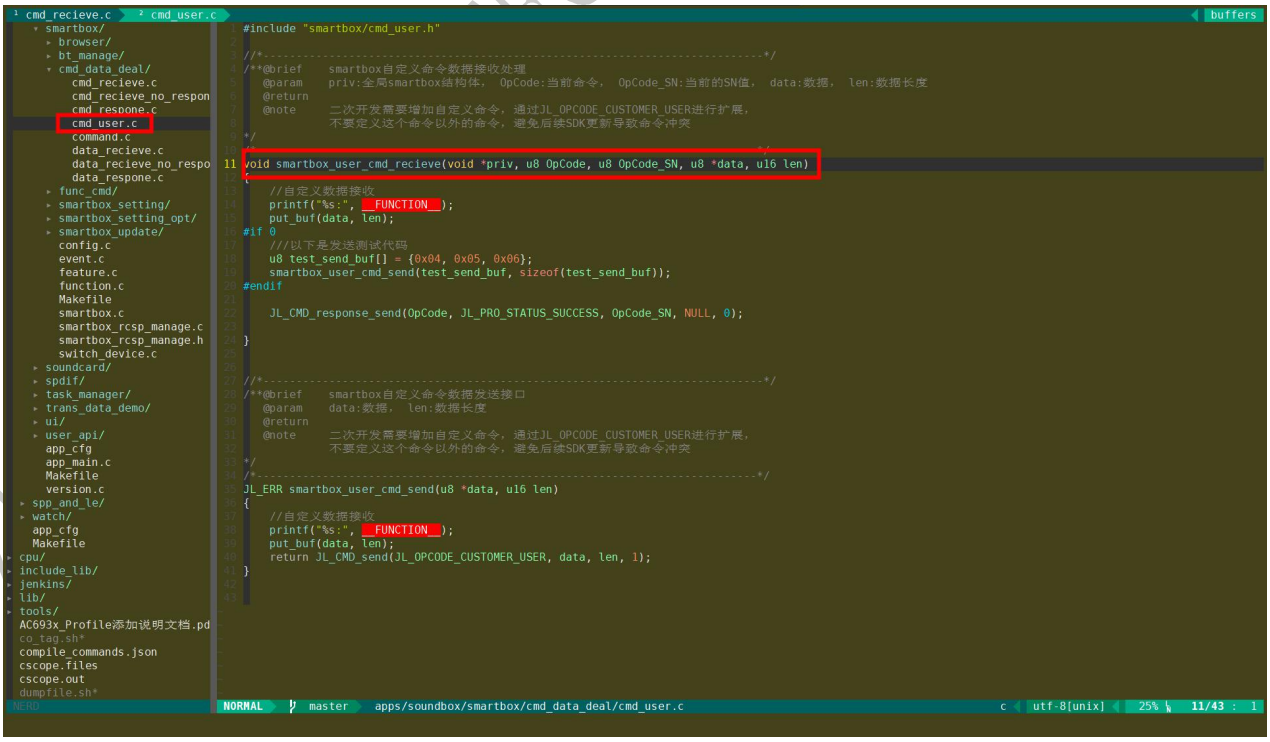
该命令分别调用到 soundbox/smartbox/function.c 中的 smartbox_function_cmd_set 函数进行模式切换和 set_tab 列表进行每个模式具体的设置。而每个模式具体的设置行为对应的代码就是存放在 func_cmd 目录中。



在 cmd_recieve 函数中，还会调用到关于弹出和升级相关的处理。弹窗对应的处理是 soundbox/smartbox/smartbox_setting_opt/smartbox_adv_bluetooth.c 文件夹，升级对应的处理是对应 smartbox_update 文件目录。



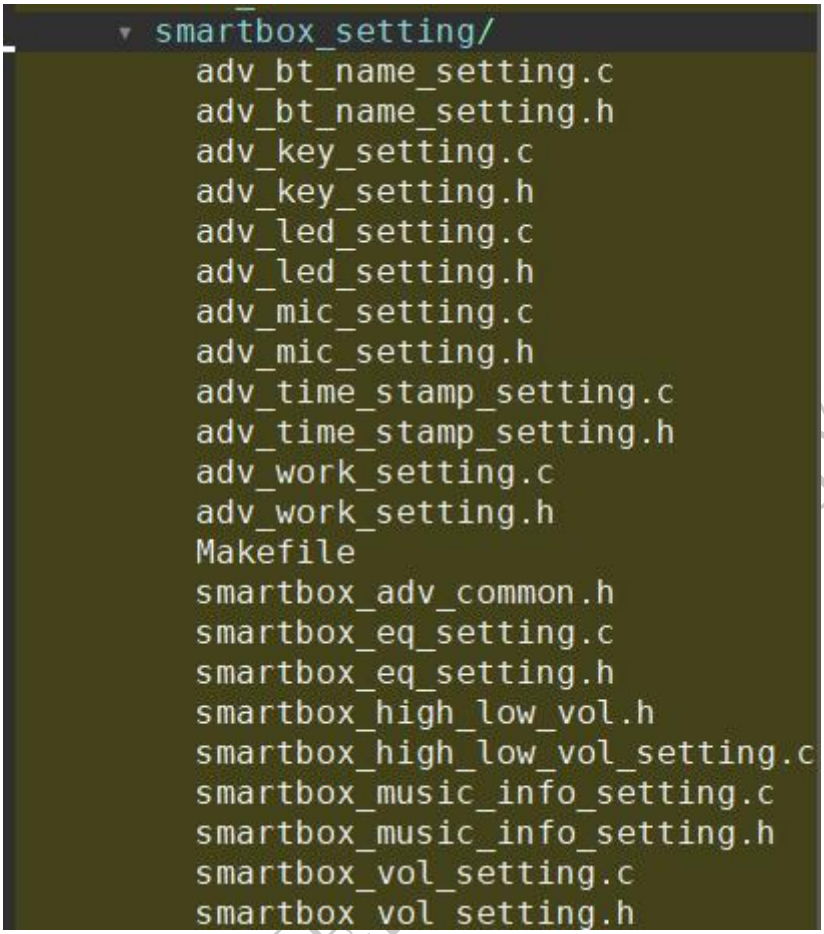
如需添加自定义命令，可在 soundbox/smartbox/cmd_data_deal/cmd_user.c 文件的 smartbox_user_cmd_recieve 函数分支进行添加自定义代码（该函数会在 cmd_recieve 函数的 JL_OPCODE_CUSTOMER_USER 分支被调用）。



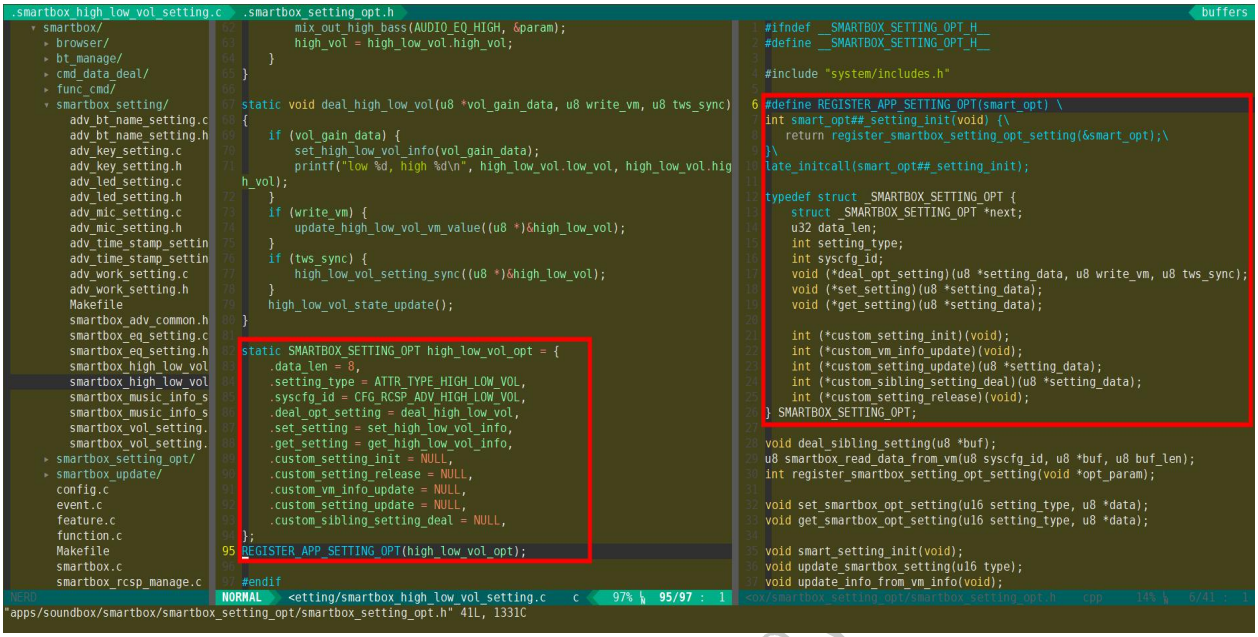
5.6 功能配置

5.6.1 功能配置

对于所有模式的具体设置的代码都存放在 soundbox/smartbox/smartbox_setting 目录下。



对于不需要读写 VM 的功能，可自己新建一个文件实现对应功能，例如 smartbox_music_info_setting.c，该文件就是蓝牙模式下实现 ID3 功能的具体设置。如果需要读写 VM，可以按照已有的格式来实现对应的功能。以 smartbox_high_low_vol_setting.c 为例子，如需按照这个格式来实现对应的功能，只需填充 SMARTBOX_SETTING_OPT 结构体，再调用 RESGISTER_APP_SETTING_OPT 把结构体放入链表即可。



如上图所示，SMARTBOX_SETTING_OPT 成员的含义：

data_len：有效数据的长度（必填）

setting_type：功能设置的类型（必填）

syscfg_id：系统配置的类型，用于 VM 的读写（必填）

deal_opt_setting：功能配置具体的实现函数（必填）

set_setting：有效数据的设置（必填）

get_setting：有效数据的获取（必填）

custom_setting_ini：自定义功能设置初始化函数（可选）

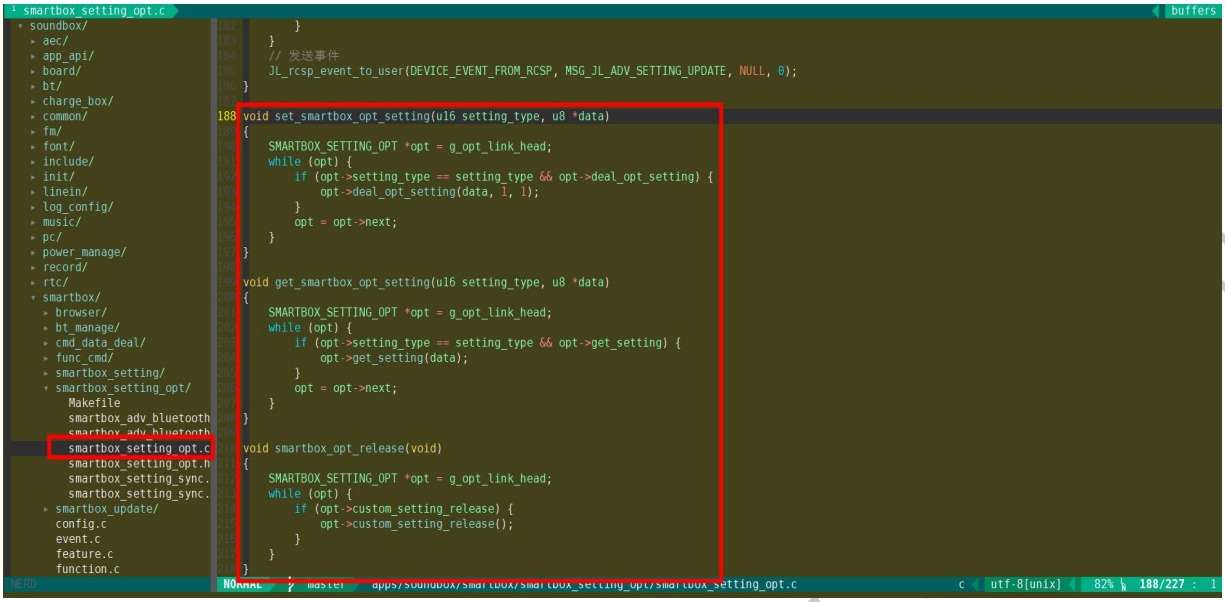
custom_setting_release：自定义功能设置释放函数（可选）

custom_vm_info_update：自定义功能设置函数，对端同步后触发，一般需要实现 VM 操作与状态更新即可（可选）

custom_setting_update：自定义设置更新函数，对端同步前数据的获取（可选）

custom_sibling_setting_deal：自定义同步动作，对端同步后数据的填充（可选）

SMARTBOX_SETTING_OPT 结构体的各个成员，都在 smartbox_smetting_opt.c 文件中被调用，具体的实现流程可查看该文件。



该文件对外提供了三个接口 set_smarbox_opt_setting 、get_smartbox_opt_setting 和 smartbox_opt_release，分别对应会调用到结构体中的 deal_opt_setting 、get_setting 和 custom_setting_release 三个成员，其中第一个参数就是对应结构体中 setting_type 的值，第二个参数是需要操作的数据。

对于 SMARTBOX_SETTING_OPT 结构体的 deal_opt_setting 成员对应函数的实现一般分为 4 个部分，数据的设置、VM 的写操作、对端同步操作和状态的更新操作，具体请参考 smartbox_high_low_vol_setting.c 的 deal_high_low_vol 函数。

对于 SMARTBOX_SETTING_OPT 结构体的 setting_type 成员的值最大不能超过 31。具体使用情况可查看 soundbox/smartbox/bt_manage/bt_trans_data/le_smartbox_module.h 文件。

```
#define ATTR_TYPE_BAT_VALUE         (0)
#define ATTR_TYPE_EDR_NAME          (1)
#define ATTR_TYPE_KEY_SETTING       (2)
#define ATTR_TYPE_LED_SETTING       (3)
#define ATTR_TYPE_MIC_SETTING       (4)
#define ATTR_TYPE_WORK_MODE         (5)
#define ATTR_TYPE_PRODUCT_MESSAGE   (6)
#define ATTR_TYPE_TIME_STAMP        (7)
#define ATTR_TYPE_EQ_SETTING        (8)
#define ATTR_TYPE_HIGH_LOW_VOL      (9)
#define ATTR_TYPE_VOL_SETTING       (10)
```

对于 SMARTBOX_SETTING_OPT 结构体的 syscfg_id 成员具体使用情况可查看

sounde/include/user_cfg_id.h 文件。

```
// #define      VM_RTC_TRIM                     18
#define     CFG_RCSP_ADV_HIGH_LOW_VOL       19
#define     CFG_RCSP_ADV_EQ_MODE_SETTING    20
#define     CFG_RCSP_ADV_EQ_DATA_SETTING    21
#define     ADV_SEQ_RAND                    22
#define     CFG_RCSP_ADV_TIME_STAMP         23
#define     CFG_RCSP_ADV_WORK_SETTING       24
#define     CFG_RCSP_ADV_MIC_SETTING        25
#define     CFG_RCSP_ADV_LED_SETTING        26
#define     CFG_RCSP_ADV_KEY_SETTING        27
#define     CFG_RCSP_ADV_VOL_SETTING        28
// #define  CFG_FLASH_BREAKPOINT            29
// #define  CFG_USB_BREAKPOINT              30
// #define  CFG_SD0_BREAKPOINT              31
// #define  CFG_SD1_BREAKPOINT              32
// #define  CFG_MUSIC_DEVICE                33
// #define  CFG_FM_RECEIVER_INFO            34
// #define  CFG_FM_TRANSMIT_INFO            35
// #define  CFG_AAP_MODE_INFO               36
// #define     CFG_UI_SYS_INFO                 37
```

对于 VM 的写可使用 syscfg_write 接口，例如：

```
static void update_high_low_vol_vm_value(u8 *vol_gain_param)
{
    syscfg_write(CFG_RCSP_ADV_HIGH_LOW_VOL, vol_gain_param, sizeof(struct _HIGH_LOW_VOL));
}
```

对于 VM 的读可使用 syscfg_read 或者封装过的 smartbox_read_data_from_vm 接口，例如：

```
u8 smartbox_read_data_from_vm(u8 syscfg_id, u8 *buf, u8 buf_len)
{
    int len = 0;
    u8 i = 0;

    len = syscfg_read(syscfg_id, buf, buf_len);

    if (len > 0) {
    ¦   for (i = 0; i < buf_len; i++) {
    ¦   ¦   if (buf[i] != 0xff) {
    ¦   ¦   ¦   return (buf_len == len);
    ¦   ¦   }
    ¦   }
    }

    return 0;
}
```

Chapter 6 APP 升级固件说明

6.1 编写目的

该章节主要描述 AC695N 系列芯片 OTA 升级功能，为用户进行二次开发提供参考

6.2 使用 sdk 自带协议进行 OTA 升级

6.2.1 在 board_ac695x_smartbox.h 中打开 CONFIG_APP_BT_ENABLE，如图

```
516
517 //*********************************************************************************//
518 //                                 AI配置                                          //
519 //*********************************************************************************//
520 #define CONFIG_APP_BT_ENABLE // AI功能、流程总开关
521
522 #ifdef CONFIG_APP_BT_ENABLE
523 #define     TRANS_DATA_EN               0
524 #define     SMART_BOX_EN                1
525 #else
526 #define     TRANS_DATA_EN               0
527 #define     SMART_BOX_EN                0
528 #endif
529
530 #if (SMART_BOX_EN)                          //rcsp需要打开ble
531 #define CONFIG_DOUBLE_BANK_ENABLE   0
532 #define RCSP_UPDATE_EN              1       //是否支持rcsp升级
533 #define OTA_TWS_SAME_TIME_ENABLE    0       //是否支持TWS同步升级
534 #define UPDATE_MD5_ENABLE           1       //升级是否支持MD5校验
535 #define RCSP_FILE_OPT               1
536 #define JL_EARPHONE_APP_EN          1
537 #define TCFG_LFN_EN                 1
538 #define TCFG_BS_DEV_PATH_EN         1
539 #else
540 #define OTA_TWS_SAME_TIME_ENABLE    0
541 #define RCSP_UPDATE_EN              0
542 #define UPDATE_MD5_ENABLE           0       //升级是否支持MD5校验
543 #define RCSP_FILE_OPT               0
544 #define JL_EARPHONE_APP_EN          0
545 #endif
```

其中 CONFIG_DOUBLE_BANK_ENABLE 为 1 是使用双备份升级，为 0 是使用单备份升级，在 Flash 空间足够的时候建议使用双备份升级（支持对耳同步进行升级）

6.2.2 从服务器进行升级相关配置

在对应的下载路径下，如 br23\tools\download_ai_double_bank 底下的 json.txt 配置文件中

```
{
    "name": "ver_info",
    "说明": "版本信息:VID(2byte),PID(2byte),VER(2byte)",
    "type":   "ARRAY",
    "data":   "0x00, 0x02, 0x00, 0x33, 0x00, 0x01"
},
{
    "name": "ver_info_ext",
    "说明": "额外信息(AuthKey,ProCode)",
    "type": "STRING",
    "data": "hE9yfseX6UdK7rFh,jl_sdk_ac697_publish"
},
```

其中 AuthKey 公司申请的用于标志公司信息的 Key, ProCode 为产品代码, VID、PID、VID 根据不同产品和版本号做修改。

6.2.3. MD5 校验说明

如使能 MD5 校验，则服务器在寻找升级文件时还会判断当前固件的 MD5 值和服务器上的 MD5 列表是否匹配。

```
#if (SMART_BOX_EN)                          //rcsp需要打开ble
#define CONFIG_DOUBLE_BANK_ENABLE    0
#define RCSP_UPDATE_EN               1      //是否支持rcsp升级
#define OTA_TWS_SAME_TIME_ENABLE     0      //是否支持TWS同步升级
#define UPDATE_MD5_ENABLE            1      //升级是否支持MD5校验
#define RCSP_FILE_OPT                1
#define JL_EARPHONE_APP_EN           1
#define TCFG_LFN_EN                  1
#define TCFG_BS_DEV_PATH_EN          1
#else
#define OTA_TWS_SAME_TIME_ENABLE     0
#define RCSP_UPDATE_EN               0
#define UPDATE_MD5_ENABLE            0      //升级是否支持MD5校验
#define RCSP_FILE_OPT                0
#define JL_EARPHONE_APP_EN           0
#endif
```

对应生成的 ufw 文件名后缀会加上改固件的 md5 值

名称	修改日期	类型	大小
jl_isd.bin	2020/6/22 14:32	BIN 文件	444 KB
jl_isd.fw	2020/6/22 14:32	FW 文件	570 KB
update_01b4e61cf1c16eee991274bccf0cbca3.ufw	2020/6/22 14:32	UFW 文件	1,011 KB
config.dat	2020/6/22 14:32	DAT 文件	1 KB

6.3 客户自带协议进行 OTA 升级

6.3.1 客户自定义升级需要打开宏定义 APP_UPDATE_EN，然后提供升级所需要的 read、seek 等接口注册到升级模块

```
#if RCSP_UPDATE_EN
#define APP_UPDATE_EN                1    //需要使用APP升级的话要把该宏打开
#else
#define APP_UPDATE_EN                0    //客户如需要开发自己的app升级协议需要把这个宏打开
#endif
```

6.3.2 相关文件：

```
update/
    rcsp_ch_loader_download.c
    update_tws.c
```

6.3.3、rcsp_ch_loader_download.c 主要实现协议提供给升级模块的接口，客户需要参考该文件**根据自己的协议实现相应的接口**，注册给升级模块（协议相关流程可以参考 thid_party_profile/JL_rcsp 目录的相关文件）

```
void rcsp_update_loader_download_init(int update_type, void (*result_cbk)(void *priv, u8 type, u8 cmd))
{
#if((OTA_TWS_SAME_TIME_ENABLE && (RCSP_ADV_EN || RCSP_BTMATE_EN)))
    if(get_tws_sibling_connect_state()) {
        extern int tws_ota_init(void);
        tws_ota_init();
        app_update_loader_downloader_init(
            update_type,
            result_cbk,
            NULL,
            &rcsp_tws_update_op);
    } else
#endif
    {
        app_update_loader_downloader_init(
            update_type,
            result_cbk,
            NULL,
            &rcsp_update_op);
    }
}
```

调用该接口将接口注册给升级模块

客户需要提供的接口有：

- File_open：

```
u16 rcsp_f_open(void)
{
    deg_puts(">>>rcsp_f_open\n");
    __this->file_offset = 0;
    __this->seek_type = BT_SEEK_SET;
    return 1;
}
```

该接口主要为初始化记录当前升级进度的 file_offset 变量。

- File_read:

```c
u16 rcsp_f_read(void *fp, u8 *buff, u16 len)
{
    //printf("===rcsp_read:%x %x\n", __this->file_offset, len);
    u8 retry_cnt = 0;

    __this->need_rx_len = len;
    __this->state = UPDATA_REV_DATA;
    __this->read_len = 0;
    __this->read_buf = buff;

#if((OTA_TWS_SAME_TIME_ENABLE && (RCSP_ADV_EN || RCSP_BTMATE_EN)))
    if((tws_ota_control(OTA_TYPE_GET) == OTA_TWS) && !get_tws_sibling_connect_state()) { //假如TWS一起升级，TWS断开了,返回失败
        r_printf("tws disconn, stop update");
        rcsp_f_stop(DEVICE_UPDATE_STA_FAIL);
        return (u16)-1;
    }
#endif

__RETRY:
    __this->data_send_hdl(fp, __this->file_offset, len);

    while (!((0 == __this->state) && (__this->read_len == len))) {
        if (__this->sleep_hdl && get_rcsp_connect_status()) {
            __this->sleep_hdl(NULL);
        } else {
            len = -1;
            break;
        }

        if (!((0 == __this->state) && (__this->read_len == len))) {
            if (retry_cnt++ > RETRY_TIMES) {
                len = (u16) - 1;
                break;
            } else {
                goto __RETRY;
            }
        }
    }

    if ((u16) - 1 != len) {
        __this->file_offset += len;
    }

    return len;
}
```

read 接口实现向主机获取升级数据，调用 data_send_hdl 向主机请求升级文件对应偏移的数据，发送请求后会 Pend 住直到获取到升级数据或者超时返回错误。

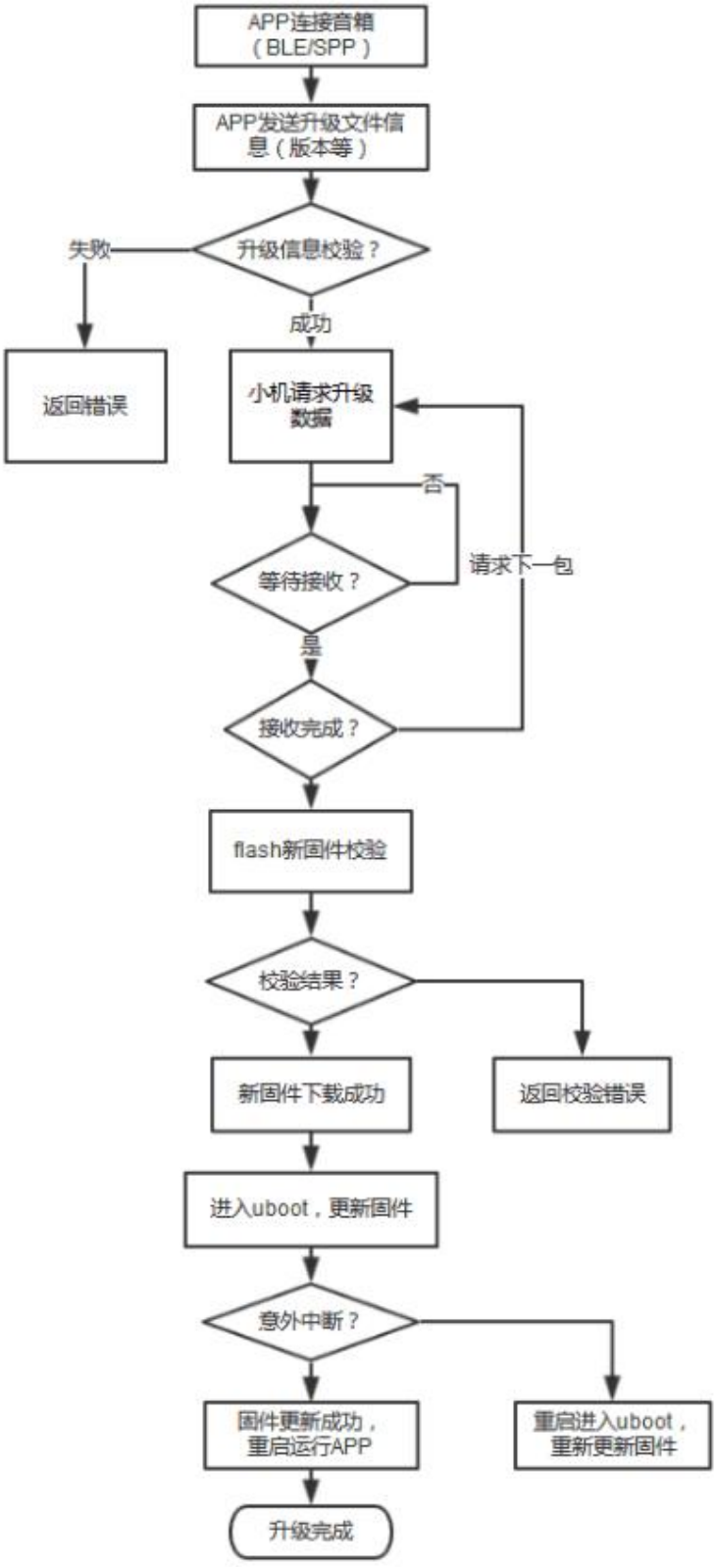
- File_seek:

```c
int rcsp_f_seek(void *fp, u8 type, u32 offset)
{
    if (type == SEEK_SET) {
        __this->file_offset = offset;
        __this->seek_type = BT_SEEK_SET;
    } else if (type == SEEK_CUR) {
        __this->file_offset += offset;
        __this->seek_type = BT_SEEK_CUR;
    }

    /* lib_printf("---------UPDATA_seek type %d,
    return 0;//FR_OK;
}
```

对 file_offset 参数进行设置

6.3.4、update_tws.c 实现 tws 同步升级的接口和流程，客户不需要修改

6.4 升级流程图



Chapter 7 SD、USB 复用 IO 介绍

7.1 编写目的

该章节主要描述 AC695N、AC696N 系列芯片 sd cmd 复用 linein ad 检测、sd clk 复用 iokey，sd data 与 usb dm 绑定复用的介绍说明。(sd cmd 复用 linein ad 检测、sd clk 复用 iokey 需要硬件电路支持,硬件电路的接法不在本文讨论，在制定方案时候请咨询硬件)

7.2 sd 复用 api 说明

函数原型	u8 sd_io_suspend(u8 sdx, u8 sd_io);
功能描述	强占 sd io 资源
参数说明	sdx 0:sd0 1:sd1 sd_io 0:cmd 1:clk 2:data
输出	0:强占成功 1：强占失败，需要 sd 当前正在使用，需要再次强占

例子	__try_t:
	if (sd_io_suspend(0, 0)) {
	os_time_dly(1);
	printf("wait %s\n", __FUNCTION__);
	goto __try_t;
	}

补充说明	cmd 和 data 使用了同一个控制器，如果有复用 data io 的应用，则不能使用 cmd 检测的方式

函数原型	u8 sd_io_resume(u8 sdx, u8 sd_io)
功能描述	释放强占 sd io 资源
参数说明	Sdx 0:sd0 1:sd1 sd_io 0:cmd 1:clk 2:data
输出	0:释放强占成功 1：释放强占失败
例子	sd_io_resume(0, 0);
补充说明	强占操作和释放操作必须成对出现，如果有强占没有释放会影响 sd 通信

7.3 sd cmd 复用 linein ad 检测

```
//*********************************************************************//
//                          linein配置                                 //
//*********************************************************************//
#define TCFG_LINEIN_ENABLE              TCFG_APP_LINEIN_EN  // linein使能
// #define TCFG_LINEIN_LADC_IDX         0                   // linein使用的ladc通道，对应ladc_list
#define TCFG_LINEIN_LR_CH               AUDIO_LIN0_LR
#define TCFG_LINEIN_CHECK_PORT          IO_PORTC_04         // linein检测IO
#define TCFG_LINEIN_PORT_UP_ENABLE      1                   // 检测IO上拉使能
#define TCFG_LINEIN_PORT_DOWN_ENABLE    0                   // 检测IO下拉使能
#define TCFG_LINEIN_AD_CHANNEL          AD_CH_PC4//NO_CONFIG_PORT       // 检测IO是否使用AD检测
#define TCFG_LINEIN_VOLTAGE             1000                // AD检测时的阀值
#define TCFG_LINEIN_INPUT_WAY           LINEIN_INPUT_WAY_ADC
#define TCFG_LINEIN_MULTIPLEX_WITH_SD   ENABLE              // linein 检测与 SD cmd 复用
#define TCFG_LINEIN_SD_PORT             1// 0:sd0 1:sd1     //选择复用的sd
```

一个典型的 linein 复用 sd cmd 配置如上图所示，下面对参与复用的配置项目进行描述。

TCFG_LINEIN_CHECK_PORT ：为 ad 的检测 io，需要填写为 sd cmd 的具体 io，根据具体方案来进行填写。

TCFG_LINEIN_AD_CHANNEL ：为 ad 的通道，头文件在 adc_api.h，用户可以根据具体方案填写，本 demo 为 AD_CH_PC4。

TCFG_LINEIN_VOLTAGE ：为 ad 检测的阈值，用户可以在 linein_sample_mult_sd 函数把 ad 值进行打印出来，判断阈值。SDK 判断当前 ad 值比阈值低时候，设备判断为插入。

TCFG_LINEIN_MULTIPLEX_WITH_SD ：为复用的开关。

TCFG_LINEIN_SD_PORT ：为 sd 的 port，需要根据具体方案进行调整，例如是 sd0 的 cmd 参与复用，则此处填写 0，反之如果是 sd1 的 cmd 参与复用，则此处是填写 1。

7.4 sd clk 复用 iokey 检测

```
49
50
51 #define TCFG_IO_MULTIPLEX_WITH_SD       ENABLE              // IO_KEY与 SD_CLK复用
52 #define TCFG_MULTIPLEX_PORT             TCFG_IOKEY_PREV_ONE_PORT // 复用的引脚
53
```

如图，在板卡中增加对应的配置即可支持 sd clk 复用 iokey 的检测，clk io 和按键之间需要串联一个小电阻，细节的硬件电路的接法不在本文讨论。

TCFG_IO_MULTIPLEX_WITH_SD ：为复用开关。

TCFG_MULTIPLEX_PORT ：为复用的引脚 io，这里填写具有方案的 clk 引脚。注意事项，因为具体应用方案可能是 sd0、sd1 的 clk io 参与了复用，因此在代码中需要注意修改参与复用的 sd。如下图函数在 io_key.c 的 sd_mult_io_detect 函数

```
134     JL_TIMER0->CON = BIT(14);
135 }
136 extern u8 sd_io_suspend(u8 sdx, u8 sd_io);
137 extern u8 sd_io_resume(u8 sdx, u8 sd_io);
138 void sd_mult_io_detect(void *arg)
139 {
140     static u32 cnt = 0;
141     if (sd_io_suspend(1, 1) == 0) {
142         gpio_set_direction(TCFG_MULTIPLEX_PORT, 0);
143         gpio_write(TCFG_MULTIPLEX_PORT, 0);
144         udelay(10);
145         gpio_set_die(TCFG_MULTIPLEX_PORT, 1);
146         gpio_set_pull_down(TCFG_MULTIPLEX_PORT, 0);
147         gpio_set_pull_up(TCFG_MULTIPLEX_PORT, 1);
148         gpio_set_direction(TCFG_MULTIPLEX_PORT, 1);
149         udelay(10);
150         mult_key_value = gpio_read(TCFG_MULTIPLEX_PORT);
151         sd_io_resume(1, 1);
152     }
153 }
154 #endif
155
```

修改对应sd

7.5 sd data 复用 usb dm

如下图，一个典型的 sd data 复用配置

```
//*********************************************************************************//
//                                 USB 配置                                        //
//*********************************************************************************//
#define TCFG_PC_ENABLE                      TCFG_APP_PC_EN//PC模块使能
#define TCFG_UDISK_ENABLE                   ENABLE_THIS_MOUDLE//U盘模块使能
#define TCFG_OTG_USB_DEV_EN                 BIT(0)//USB0 = BIT(0)  USB1 = BIT(1)

#include "usb_std_class_def.h"


#define TCFG_USB_PORT_CHARGE            DISABLE

#define TCFG_USB_DM_MULTIPLEX_WITH_SD_DAT0      1
#if TCFG_USB_DM_MULTIPLEX_WITH_SD_DAT0
//复用情况下，如果使用此USB口作为充电（即LDO5V_IN连接到此USB口），
//TCFG_OTG_MODE需要或上TCFG_OTG_MODE_CHARGE，用来把charge从host区
//分开；否则不需要，如果LDO5V_IN与其他IO绑定，则不能或上
#define TCFG_DM_MULTIPLEX_WITH_SD_PORT      0//0:sd0  1:sd1 //dm 参与复用的sd配置
#undef TCFG_OTG_MODE
#define TCFG_OTG_MODE                       (TCFG_OTG_MODE_HOST|TCFG_OTG_MODE_SLAVE|TCFG_OTG_MODE_CHARGE|OTG_DET_DP_ONLY)

#undef USB_DEVICE_CLASS_CONFIG
#if TCFG_SD0_SD1_USE_THE_SAME_HW //开启了双卡的可以使能读卡器存续设备
#define     USB_DEVICE_CLASS_CONFIG (MASSSTORAGE_CLASS|SPEAKER_CLASS|MIC_CLASS|HID_CLASS)
#else
#define     USB_DEVICE_CLASS_CONFIG (SPEAKER_CLASS|MIC_CLASS|HID_CLASS)
#endif

#undef TCFG_SD0_DET_MODE
#define TCFG_SD0_DET_MODE                   SD_CLK_DECT
#define TCFG_USB_SD_MULTIPLEX_IO            IO_PORTB_03

#endif
```

下面对参与 usb 和 sd 复用的配置项目进行说明：

TCFG_USB_DM_MULTIPLEX_WITH_SD_DAT0 ：复用的总开关。

TCFG_DM_MULTIPLEX_WITH_SD_PORT ：参与复用的 sd，根据具体的封装应用场景进行填写。0 为 sd0、1 为 sd1。

TCFG_OTG_MODE :复用情况下，如果使用此 USB 口作为充电（即 LDO5V_IN 连接到此 USB 口），TCFG_OTG_MODE 需要或上 TCFG_OTG_MODE_CHARGE，用来把 charge 从 host 区分开；否则不需要，如果 LDO5V_IN 与其他 IO 绑定，则不能或上。

USB_DEVICE_CLASS_CONFIG :从机模式支持的功能配置，如果只有单卡的场景，usb 和 sd 复用下，不支持读卡器功能。注意事项：在插入 usb 线场景下播放 sd 卡音乐，sd 卡播放可能有影响，这是无法解决的，干扰来源来着电脑端。用户可以选择性关闭 pc 模式。

TCFG_USB_SD_MULTIPLEX_IO ： 为 sd 卡 data io,用户需要根据具体的应用方案进行填写。

注意事项：如下图 usb 和 sd data 复用场景 sd 不能使用 cmd 检测，只是 sd 控制器决定，因此本 demo 默认重新修改了检测方式为 clk 检测。用户需要根据具体的方案应用选择 clk 检测或者 io 检测。特别注意，如果是 sd1 参与复用，下面的检测方式应该是修改为 sd1 的对应配置。

```
163 #endif
164
165 #undef TCFG_SD0_DET_MODE
166 #define TCFG_SD0_DET_MODE                SD_CLK_DECT
167 #define TCFG_USB_SD_MULTIPLEX_IO         IO_PORTB_03
168
```


Chapter 8 常用模块使用介绍

8.1 编写目的

该文档主要描述 AC695x_SDK 一些常用设备使用以及注意的一些问题，为用户进行二次开发提供参考

8.2 fft 使用

```
//same 是 data 跟 outdata 是不是同一片地址，是为 1，不是为 0
extern void REAL_FFT_FUN(int *data,int blockbit,int *outdata,int same);          //实数 fft
extern void REAL_IFFT_FUN(int *data,int blockbit,int *outdata,int same);         //实数 ifft

extern void COMPLEX_FFT_FUN(int *data,int blockbit,int *outdata,int same);       //复数 fft
extern void COMPLEX_IFFT_FUN(int *data,int blockbit,int *outdata,int same);      //复数 ifft
```

8.3 三角函数使用

此页主要说明，定点(Fix)和浮点(Float)的数学接口函数的调用方法及注意事项！

目前仅支持 sin、cos、arctan(y/x)、sqrt(x^2+y^2)、sinh、cosh、exp、arctanh(y/x)、sqrt(x^2-y^2)、ln(x)、sqrt(x)共 11 个常用数学函数调用

特别说明：1、各数学接口函数调用时的输入数据，应严格遵循表 1 中的限制；

2、由于受 int 型数据位宽限制，exp 函数的输入数据范围应限制在[-10，4.852]；

3、数据的精度损失详见 Reference relative error。

函数接口请查看 math.h 文件

Fix:	相应数学接口函数定义详见 MathFunc_fix.h
eg.1:	void sin_fix(int a, int aQ, int rQ, int *r) 计算定点数a的正弦值r， 其中：aQ为数a的实际值移位数； rQ为计算sin输出r的放大倍数位宽，可自定义。 注意：计算sin、cos的输入a值均是不包含pi！详见表1
eg.2:	void arctan_fix(int x, int xQ, int y, int yQ, int rQ, int *r) 计算定点数y/x的反正切值， 其中：xQ为输入定点数x的实际放大倍数位宽； yQ为输入定点数y的实际放大倍数位宽； rQ为计算arctan输出r的放大倍数位宽，可自定义。 注意：计算arctan时的输出r值是乘以了pi！详见表1

Float：	相应数学接口函数定义详见 MathFunc_float.h
eg.1：	void sin_float(float a, float *r)
	计算浮点数a的正弦值r，
	注意：计算sin、cos的输入a值均是不包含pi！详见表1
eg.2：	void arctan_float(float x, float y, float *r)
	计算浮点数y/x的反正切值r，
	注意：计算arctan时的输出r值是乘以了pi！详见表1

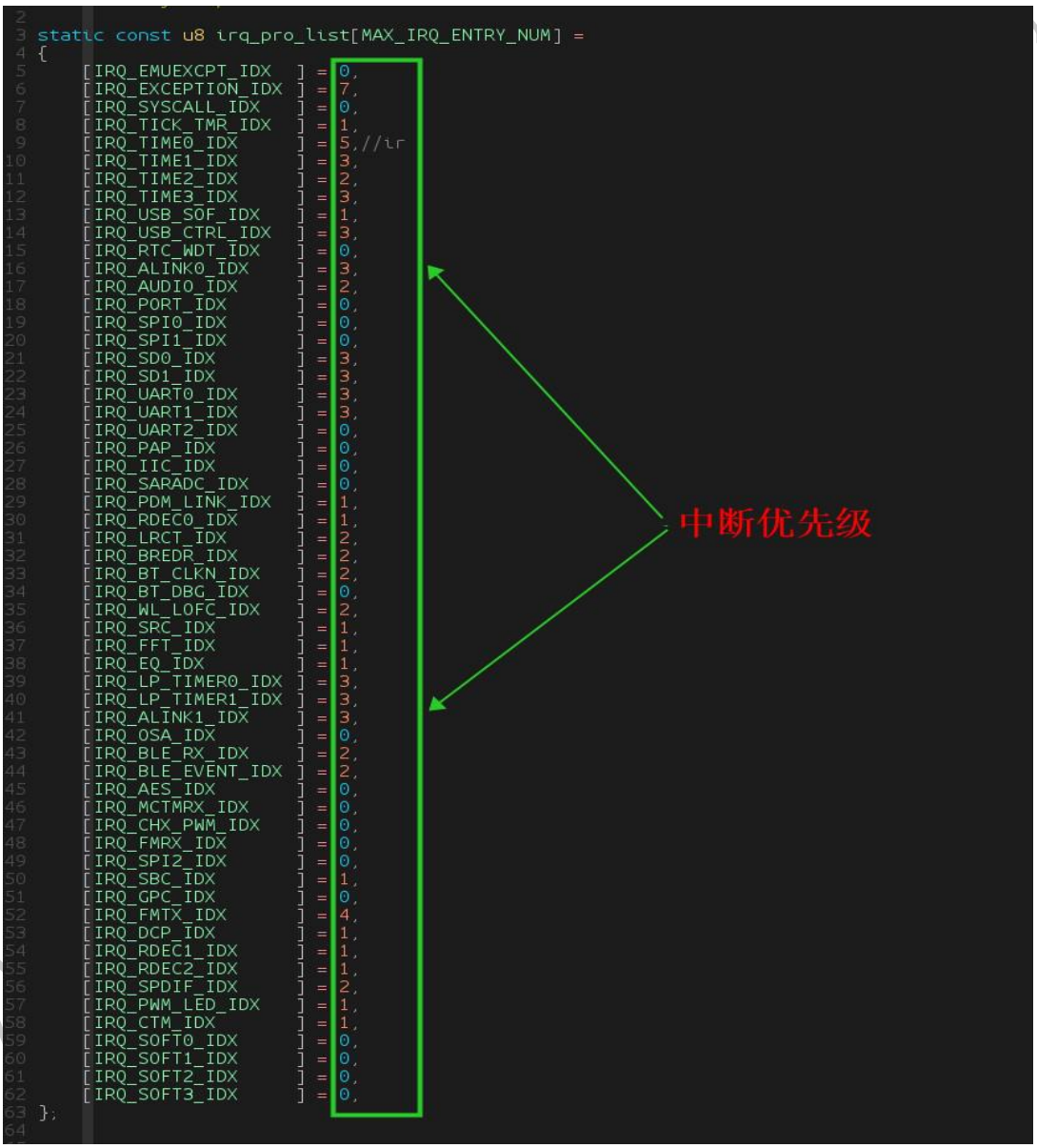
Function	Input Field	Output Field
$\sin(x)$	$x \in [-128\pi, 127\pi]$, $\pi=1\cdot 2^{24}$	$[-1,1]$, $1=1\cdot 2^{24}$
$\cos(x)$		
$\arctan(y/x)$	$x, y \in [-2^{29}, 2^{29})$, $1=1\cdot 2^{0}$	$[-\pi,\pi]$, $\pi=1\cdot 2^{24}$
$\sqrt{x^2+y^2}$		$[0,2^{30}]$, $1=1\cdot 2^{0}$
$\sinh(x)$	$[-1,1]$, $1=1\cdot 2^{24}$	$[-1.1752,1.1752]$, $1=1\cdot 2^{24}$
$\cosh(x)$		$[1,1.5431]$, $1=1\cdot 2^{24}$
e^x	$[-10,10]$, $1=1\cdot 2^{24}$	$[4.54\times10^{-5}, 2.2\times10^{4}]$, $1=1\cdot 2^{24}$
$\operatorname{arctanh}(y/x)$	$x, y \in (0,127]$, $\frac{1}{32} \le \frac{y}{x} \le \frac{4}{5}$	$[0.0313,1.0986]$, $1=1\cdot 2^{24}$
$\sqrt{x^2-y^2}$	$1=1\cdot 2^{24}$	$(0,127]$, $1=1\cdot 2^{24}$
$\ln(x\cdot 2^{y})$	$x \in (0,127]$, $y \in [-32,31]$	$(0,37.99)$, $1=1\cdot 2^{24}$
$\sqrt{x}$	$x \in [0, 2^{31})$, $1=1\cdot 2^{0}$	$[0, 2^{15}]$, $1=1\cdot 2^{15}$

Chapter 9　中断和线程优先级时钟

9.1 中断优先级设置

文件：irq_config.c

关键数组：static const u8 irq_pro_list[MAX_IRQ_ENTRY_NUM]内部参数说明：



说明：最低优先级：0；最高优先级：7；没有特殊要求不建议修改，避免引起系统崩溃。

9.2 线程优先级设置

文件：task_table.c

关键数组：const struct task_info task_info_table[]

```
/*任务列表 */
const struct task_info task_info_table[] = {
    {"app_core",        5,    1024,   512  },
    {"sys_event",       6,    256,    0    },
    {"btctrler",        4,    512,    384  },
    {"tws",             5,    512,    128  },
    {"btstack",         3,    768,    256  },
#if (defined(TCFG_DEV_PREPROCESSOR_ENABLE) && TCFG_DEV_PREPROCESSOR_ENABLE)
    {"dev_pre",         2,    256,    0    },
    {"audio_dec",       3,    768 + 32,   128  },
#else
    {"audio_dec",       3,    768 + 32,   128  },
#endif
    {"dev_mg",          3,    512,    512  },
    {"music_ply",       4,    1024,   128  },
    {"audio_enc",       3,    512,    128  },
    {"usb_msd",         2,    512,    128  },
    {"aec",             2,    768,    128  },
    {"aec_dbg",         3,    128,    128  },
    {"update",          1,    256,    0    },
    {"systimer",        6,    128,    0    },
    {"usb_audio",       5,    256,    256  },
    {"plnk_dbg",        5,    256,    256  },
    {"adc_linein",      2,    768,    128  },
    {"enc_write",       1,    768,    0    },
#if (GMA_EN)
    {"tm_gma",          4,    768,    256  },
#endif
    {"ui",              3,    384,    64   },
    {0, 0},
};
```

优先级

说明：最低优先级：0；最高优先级：6；没有特殊要求不建议修改，避免引起系统崩溃。

9.3 时钟设置

文件：clock_manager.c

关键数组：const struct clock_type　clock_enum[]

```
const struct clock_type  clock_enum[] = {

    { BT_IDLE_CLOCK          , (24), "   BT_IDLE_CLOCK      " },
    { MUSIC_IDLE_CLOCK       , (24), "   MUSIC_IDLE_CLOCK   " },
    { FM_IDLE_CLOCK          , (48), "   FM_IDLE_CLOCK      " },
    { LINEIN_IDLE_CLOCK      , (96), "   LINEIN_IDLE_CLOCK  " },
    { PC_IDLE_CLOCK          , (120), "  PC_IDLE_CLOCK      " },
    { REC_IDLE_CLOCK         , (12), "   REC_IDLE_CLOCK     " },
    { RTC_IDLE_CLOCK         , (12), "   RTC_IDLE_CLOCK     " },
    { SPDIF_IDLE_CLOCK       , (12), "   SPDIF_IDLE_CLOCK   " },
    { BOX_IDLE_CLOCK         , (24), "   BOX_IDLE_CLOCK   " },
    /*************************************************************

    { DEC_SBC_CLK    , (32), "DEC_SBC_CLK     " },
    { DEC_AAC_CLK    , (32), "DEC_AAC_CLK     " },
    { DEC_MSBC_CLK   , (12), "DEC_MSBC_CLK    " },
    { DEC_CVSD_CLK   , (8), "DEC_CVSD_CLK     " },

    { AEC8K_CLK , (48), "AEC8K_CLK       " },
    { AEC8K_ADV_CLK , (60), "AEC8K_ADV_CLK   " },
    { AEC16K_CLK     , (80), "AEC16K_CLK  " },
    { AEC16K_ADV_CLK, (120), "AEC16K_ADV_CLK " },

    { DEC_TONE_CLK   , (24 + 32), "DEC_TONE_CLK    " },
    { DEC_MP3_CLK    , (46), "DEC_MP3_CLK     " },
    { DEC_WAV_CLK    , (24), "DEC_WAV_CLK     " },
    { DEC_G729_CLK   , (24), "DEC_G729_CLK    " },
    { DEC_G726_CLK   , (24), "DEC_G726_CLK    " },
```

红色框为每个模式的 idle 时钟，在进入每一个模式会配置一次；

后面的时钟是每一种算法或者运行的操作需要添加的时钟值，在要运行对应算法的时候需要添加对应时钟，在结束对应处理的时候要把时钟删除。

把时钟设置加入到 ext 中，但是不是立刻设置时钟

需要最后调用 clock_set_cur 来最后设置时钟

用于连续地方设置时钟

用完后必须把时钟 remove

void clock_add(u32 type)

void clock_remove(u32 type)

void clock_set_cur(void)

把时钟设置加入到 ext 中，立刻设置时钟

需要立刻添加时钟

用完后必须把时钟 remove

void clock_add_set(u32 type)

void clock_remove_set(u32 type)

如果想要固定频率，可以设置下面宏定义，为非 0 就是固定频率值

```
////  如果clock_fix 为0 就按照配置设置时钟，如果有值就固定频率
#define CLOCK_FIX   0//192
```


Chapter 10 Audio 模块使用

10.1 支持的音效列表

序号	音效名	中文名	备注
1	EQ	均衡器	
2	DRC	动态范围压缩	
3	VBass	虚拟低音	
4	Voice Changer	变声	
5	CrossOver	分频器	
6	Dynamic EQ	动态 EQ	
7	Echo	回声	
8	Plate Reverb	混响	
9	Frequency Shift	啸叫抑制——移频	
10	Howling Suppress	啸叫抑制——陷波	
11	Surround Effect	环绕声	
12	Autotune	电音修音	
13	BassTreble	高低音调节	
14	EnergyDetect	能量检测	
15	PitchSpeed	变速变调	
16	Spectrum	频谱	
17	StereoMix	立体声混合	
18	Vocal Remover	人声消除	
19	NoiseGate	噪声门限	

10.2 在线调试配置

打开板级配置中的在线调试功能，根据需要可自行选择 USB/SPP/UART 通讯方式，下载之后打开调音工具，即可读取 bin 文件，进入在线调音。

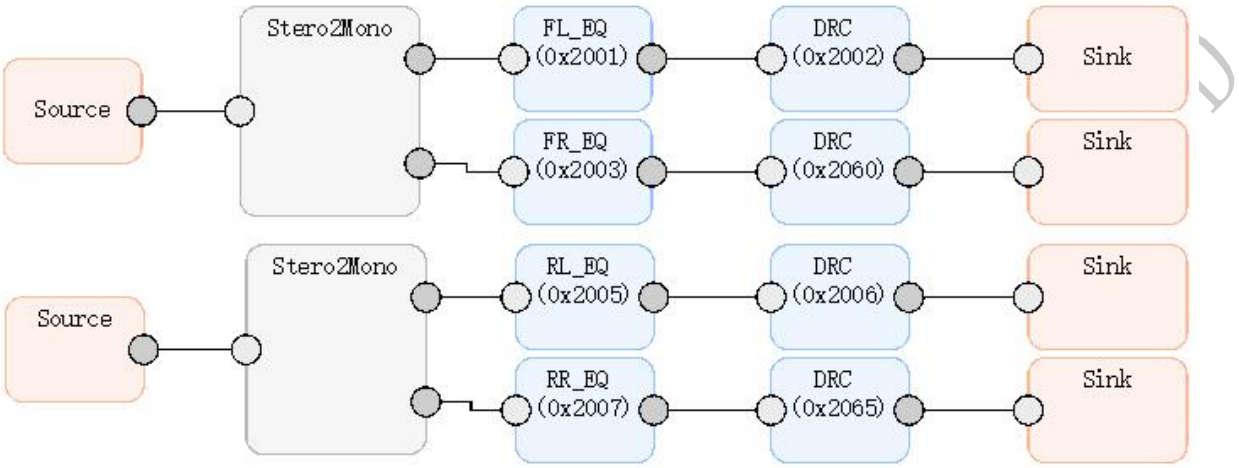
```
//*********************************************************************************//
//                          新音箱配置工具 && 调音工具                             //
//*********************************************************************************//
#define TCFG_SOUNDBOX_TOOL_ENABLE           DISABLE         //是否支持音箱在线配置工具
#define TCFG_EFFECT_TOOL_ENABLE             1 //DISABLE         //是否支持在线音效调试,使能该项还需使能EQ总使能TCFG_EQ_ENABL,
#define TCFG_NULL_COMM                      0               //不支持通信
#define TCFG_UART_COMM                      1               //串口通信
#define TCFG_USB_COMM                       2               //USB通信
#define TCFG_SPP_COMM                       3               //USB通信
#if (TCFG_SOUNDBOX_TOOL_ENABLE || TCFG_EFFECT_TOOL_ENABLE)
#define TCFG_COMM_TYPE                      TCFG_USB_COMM   //通信方式选择
#else
#define TCFG_COMM_TYPE                      TCFG_NULL_COMM
#endif
#define TCFG_TOOL_TX_PORT                   IO_PORT_DP      //UART模式调试TX口选择
#define TCFG_TOOL_RX_PORT                   IO_PORT_DM      //UART模式调试RX口选择
#define TCFG_ONLINE_ENABLE                  (TCFG_EFFECT_TOOL_ENABLE)     //是否支持音效在线调试功能
```

10.2.1 四声道 EQ 独立调试

使能**四声道输出、四声道 eq 独立**宏

```
/*
DAC 硬件上的连接方式,可选的配置:
    DAC_OUTPUT_MONO_L               左声道
    DAC_OUTPUT_MONO_R               右声道
    DAC_OUTPUT_LR                   立体声
    DAC_OUTPUT_MONO_LR_DIFF         单声道差分输出
    DAC_OUTPUT_FRONT_LR_REAR_LR     四声道输出
*/
#define TCFG_AUDIO_DAC_CONNECT_MODE     DAC_OUTPUT_FRONT_LR_REAR_LR

#define TCFG_EQ_DIVIDE_ENABLE      1  // 四声道 eq 是否独立  0 使用同个 eq 效果
```

下载后会默认使用 music_base_quad.bin 文件，调音界面如图所示：



10.3 第三代音效配置

第三代音效添加了 DRC_PRO 模块，分频器添加 MDRC_V2 版本可选。DRC PRO 增加压缩启动时间、压缩释放时间、检测时间，能够产生更好压制效果。

```
1. #define TCFG_DRC_ENABLE                    0     //DRC 总使能
2. #define TCFG_DRC_PRO_ENABLE                1     //DRC PRO 总使能
3. #define TCFG_AUDIO_MDRC_ENABLE             2     //多带 drc 使能  0:关闭多带 drc,  1:
   使能多带 drc  2：使能多带 drc 并且 多带 drc 后再做一次全带的 drc
4.
5. #define MDRC_V1                            1
6. #define MDRC_V2                            2     //MDRC_V2 目前仅支持两带，当读取 bin 文
   件中分频器为三带时，自动切回 MDRC_V1
7. #define TCFG_AUDIO_MDRC_VERSION            MDRC_V2
```

配置了 DRC_PRO 使能之后，会将流程中的 DRC 替换为 DRC_PRO，使用 music_drc_pro.bin（仅音乐通路，带有分频器），mic_drc_pro.bin（音乐+麦克风通路，没有分频器）。

注：1、由于 AC695N 资源限制，若使能麦克风音效，则会自动关闭分频器（MDRC）。

2、在线调试需要将工具升级到最新版本。

Chapter 11 客户常见问题反馈解答

Q1：

Q2：