

BK7251 RTOS SDK

API Reference



Better Life with Wireless
美好生活尽在无线

Version 3.0.0
Copyright © 2020



Release Notes

Version	Date	Description
V1.0	2019.04	First Release
V2.0	2019.08	1.Modified about bootloader 2.add qspi/jpeg module
V3.0	2020.03	1.Adjust document's structure 2.add BLE module

目录

Release Notes	2
1 ADC	18
1.1 ADC 简介	18
1.2 ADC Related API	18
1.2.1 adc 通用结构体说明	18
1.2.2 创建 adc 检测线程	18
1.2.3 配置 adc 检测通道及回调函数	19
1.2.4 启动 adc	19
1.2.5 关闭 adc	19
1.3 ADC 示例代码	19
2 PWM	22
2.1 PWM 简介	22
2.2 PWM Related API	22
2.2.1 pwm 枚举类型说明	22
2.2.2 初始化 pwm	22
2.2.3 启动 pwm 功能	23
2.2.4 停止 pwm 功能	23
2.3 PWM 示例代码	23
2.4 操作说明	24
2.4.1 打开配置	24
2.4.2 运行现象	24
2.5 注意事项	25
3 Audio	26
3.1 Audio 简介	26
3.2 Audio Related API	26
3.2.1 audio 通用结构体说明	26
3.2.2 audio 设备初始化	26
3.2.3 打开 mic	27
3.2.4 设置 mic 数据采集通道	27

3.2.5 设置 mic 采样率	27
3.2.6 采集 mic 声音数据	27
3.2.7 关闭 mic	28
3.2.8 打开 audio 设备	28
3.2.9 设置 dac 采样率	28
3.2.10 设置 dac 音量	28
3.2.11 获取 dac 数据, 播放音频	28
3.2.12 关闭 dac	29
3.3 Audio 示例代码	29
3.3.1 关键说明	29
3.3.2 示例代码	29
4 Button	34
4.1 Button 简介	34
4.2 Button Related API	34
4.2.1 button 初始化	34
4.2.2 配置回调函数	34
4.2.3 开始 button 工作	35
4.2.4 结束 button 工作	35
4.3 Button 示例代码	35
4.3.1 关键说明	35
4.3.2 示例代码	35
4.4 操作说明	38
4.4.1 打开配置	38
4.4.2 按键测试	38
5 I2C 总线	40
5.1 I2C 简介	40
5.2 模拟 I2C Related API	40
5.2.1 寻找总线获取设备句柄	40
5.2.2 对从设备的读写数据	40
5.3 模拟 I2C 总线示例代码	41
5.3.1 关键说明	41

5.3.2 对从设备的读写数据	41
5.4 硬件 I2C 总线示例代码	45
5.4.1 硬件 I2C 接口说明	45
5.4.2 对从设备的读写数据	45
6 I2S 总线	50
6.1 I2S 简介	50
6.2 I2S Related API	50
6.2.1 i2s 通用结构体说明	50
6.2.2 i2s 模块参数设置	50
6.2.3 i2s 主从设备发送/接收数据	51
6.3 I2S 示例代码	51
6.3.1 关键说明	51
6.3.2 示例代码	52
7 通用 SPI	57
7.1 通用 SPI 简介	57
7.2 通用 SPI Related API	57
7.2.1 spi 通用结构体说明	57
7.2.2 spi 模块配置	57
7.2.3 spi 发送数据	58
7.2.4 spi 接收数据	58
7.3 通用 SPI 示例代码	58
7.3.1 关键说明	59
7.3.2 示例代码	59
7.4 操作说明	62
7.4.1 打开配置	62
7.4.2 运行现象	62
7.5 注意事项	63
8 通用 SPI FLASH 设备	64
8.1 通用 SPI FLASH 简介	64
8.2 通用 SPI FLASH Related API	64
8.2.1 控制设备	64

8.3 通用 SPI FLASH 示例代码.....	64
8.3.1 关键说明	64
8.3.2 示例代码	64
8.4 操作说明	67
8.4.1 运行现象	67
8.5 注意事项	68
9 通用 SPI PSRAM 设备	69
9.1 通用 SPI PSRAM 简介	69
9.2 通用 SPI PSRAM related API	69
9.3 通用 SPI PSRAM 示例代码	69
9.3.1 关键说明	69
9.3.2 示例代码	69
9.4 操作说明	71
9.4.1 打开配置	71
9.4.2 运行现象	71
9.5 注意事项	71
10 高速 SPI 从设备	72
10.1 高速 SPI 从设备简介	72
10.2 高速 SPI 从设备 Related API	72
10.3 高速 SPI 从示例代码	72
10.3.1 关键说明	72
10.3.2 示例代码	72
10.4 操作说明	75
10.4.1 打开配置	75
10.4.2 运行现象	75
10.5 注意事项	75
11 GPIO	76
11.1 GPIO 简介	76
11.2 GPIO Related API	76
11.2.1 设置引脚模式	76
11.2.2 设置引脚电平	76

11.2.3 读取引脚电平	77
11.2.4 绑定引脚中断回调函数	77
11.2.5 使能引脚中断	77
11.2.6 脱离引脚中断回调函数	77
11.3 GPIO 示例代码	78
11.3.1 关键说明	78
11.3.2 示例代码	78
11.4 操作说明	79
11.4.1 打开配置	80
11.4.2 运行现象	80
12 UART	81
12.1 UART 简介	81
12.2 UART Related API	81
12.2.1 uart 通用结构体说明	81
12.2.2 控制串口设备	81
12.3 UART 示例代码	82
12.3.1 关键说明	82
12.3.2 示例代码	83
12.4 操作说明	85
12.4.1 打开配置	86
12.4.2 运行现象	86
12.5 注意事项	86
13 Player	87
13.1 Player 简介	87
13.2 Player Related API	87
13.2.1 初始化播放器	87
13.2.2 开始/恢复播放	87
13.2.3 停止播放	88
13.2.4 暂停播放	88
13.2.5 设置当前歌曲播放的位置	88
13.2.6 设置播放歌曲的 uri	88

13.2.7 获取当前歌曲的 uri.....	88
13.2.8 获取当前播放器的状态	89
13.2.9 准备开始播放用户音频数据流	89
13.2.10 写入用户音频数据到播放器	89
13.2.11 设置用户音频数据的总长度	89
13.2.12 设置播放层事件的回调函数	89
13.2.13 设置播放器音量	90
13.2.14 设置播放器音量	90
13.2.15 获取当前歌曲的持续时间	90
13.2.16 获取当前歌曲播放的位置	90
13.3 Player 示例代码.....	90
13.4 操作说明	98
13.4.1 运行现象	98
14 网络接口	100
14.1 网络接口简介	100
14.2 网络接口 Related API.....	100
14.2.1 启动网络	100
14.2.2 启动 STATION 快速连接	101
14.2.3 关闭网络	102
14.2.4 启动 scan	102
14.2.5 注册 scan 结束后的回调函数	102
14.2.6 scan 特定的网络	102
14.2.7 启动监听模式	102
14.2.8 关闭监听模式	103
14.2.9 注册监听回调函数	103
14.2.10 获取当前的网络状态	103
14.2.11 获取当前的连接状态	104
14.2.12 获取当前的信道	104
14.2.13 设置信道	105
14.3 网络接口使用示例	105
14.3.1 关键说明	105

14.3.2 代码示例	105
14.4 操作说明	115
14.4.1 启动 STATION 连接	115
14.4.2 启动 STATION 快速连接	116
14.4.3 STATION 模式获取状态	117
14.4.4 启动 AP	117
14.4.5 AP 模式获取状态	117
14.4.6 启动 SCAN	118
14.4.7 启动混杂包监听	119
15 RTOS 接口	120
15.1 RTOS 接口简介	120
15.2 RTOS Related APIs	120
15.2.1 RTOS 结构体说明	120
15.2.2 创建一个新的线程	121
15.2.3 删除一个使用结束的线程	122
15.2.4 使当前线程挂起，等待另一个线程终止	122
15.2.5 使一个线程挂起一段时间	122
15.2.6 初始化一个信号量	122
15.2.7 发出信号量	122
15.2.8 获取一个信号量，并提供超时机制	123
15.2.9 销毁一个信号量	123
15.2.10 初始化一个互斥锁	123
15.2.11 获得一个互斥锁	123
15.2.12 释放一个互斥锁	123
15.2.13 销毁一个互斥锁	124
15.2.14 初始化一个消息队列	124
15.2.15 将一个数据对象推入消息队列	124
15.2.16 从消息队列中取出一个数据对象	124
15.2.17 销毁一个消息队列	125
15.2.18 查询一个队列是否为空	125
15.2.19 查询一个队列是否已满	125

15.2.20	初始化一个时钟，并传入回调函数	125
15.2.21	启动一个时钟	126
15.2.22	停止一个时钟	126
15.2.23	重新加载一个过期的时钟	126
15.2.24	销毁一个时钟	126
15.2.25	获取一个时钟是否正在运行	126
15.3	RTOS 关键说明	127
16	OTA	128
16.1	OTA 简介	128
16.2	OTA Related API	128
16.2.1	fal 初始化	128
16.2.2	远程下载固件	128
16.3	OTA 示例代码	129
16.4	操作说明	130
16.4.1	生成 rbl 升级文件	130
16.4.2	搭建本地 HTTP Server 环境	130
16.4.3	运行现象	131
17	Bootloader	133
17.1	Bootloader 简介	133
17.2	分区表的设置	133
17.2.1	Bootloader 分区	133
17.2.2	App 分区	133
17.2.3	Download 分区	133
17.3	L_boot	134
17.4	UP_boot	134
17.5	获取 bootloader.bin 文件	134
17.6	生成 all.bin 文件	134
17.7	Bootloader 示例代码	135
17.7.1	2M 分区表信息配置文件 partition_audio_2M.json 示例	135
17.7.2	UP_boot 示例	136
17.7.3	生成 all.bin 的配置文件 config_sample.json 示例	139

18 低功耗.....	140
18.1 低功耗简介	140
18.2 低功耗 Related API.....	140
18.2.1 进入低功耗模式	140
18.2.2 deep_sleep 模式	140
18.3 低功耗示例代码	141
18.3.1 关键说明	141
18.3.2 示例代码	141
18.4 操作说明	144
18.4.1 连接万用表	144
18.4.2 运行现象	145
19 混音	146
19.1 混音简介	146
19.2 混音 Related API.....	146
19.2.1 混音初始化	146
19.2.2 暂停背景音播放	146
19.2.3 重新播放背景音	146
19.3 混音示例代码	147
19.3.1 关键说明	147
19.3.2 示例代码	147
19.4 操作说明	148
19.4.1 打开配置	148
19.4.2 运行现象	148
20 Airkiss 配网.....	149
20.1 Airkiss 简介	149
20.2 Airkiss Related API.....	149
20.2.1 开始 airkiss	149
20.2.2 获取 airkiss 状态	149
20.2.3 获取 airkiss 解码结果	149
20.3 Airkiss 示例代码.....	150
20.3.1 关键说明	150

20.3.2 示例代码	150
20.4 操作说明	152
20.4.1 扫描微信 airkiss 配网二维码或下载 Airkiss 调试工具	152
20.4.2 运行现象	152
20.5 注意事项	154
21 声波配网	155
21.1 声波配网简介	155
21.2 声波配网 Related API	155
21.2.1 声波配网开始	155
21.2.2 用户提前终止声波配网	156
21.2.3 获取版本号	156
21.3 声波配网示例代码	156
21.4 操作说明	161
21.4.1 打开配置	161
21.4.2 运行现象	161
22 Vad	164
22.1 Vad 自动语音检测简介	164
22.2 Vad Related API	164
22.2.1 进入 vad 检测模式	164
22.2.2 获取帧的长度	164
22.2.3 vad 入口函数	164
22.2.4 关闭 vad	165
22.3 Vad 示例代码	165
22.4 操作说明	168
22.4.1 打开配置	168
22.4.2 运行现象	168
23 AMR 编码器	170
23.1 AMR 编码器简介	170
23.2 AMR 编码器 Related API	170
23.2.1 AMR-NB 编码器初始化	170
23.2.2 AMR-NB 编码	170

23.2.3 释放 AMR-NB 编码	171
23.3 AMR 编码器示例代码	171
23.3.1 关键说明	171
23.3.2 示例代码	171
23.4 操作说明	184
23.4.1 下载 AMR Player 工具	184
23.4.2 网络调试助手设置	184
23.4.3 打开配置	184
23.4.4 运行现象	184
24 Opus 编码器	185
24.1 Opus 编码器简介	185
24.2 Opus 编码器 Related API	185
24.2.1 创建 opus 编码器	185
24.2.2 返回 opus 编码器所需内存的大小	185
24.2.3 修改 opus 编码器的复杂度	186
24.2.4 获取 opus 编码器的比特率	186
24.2.5 获取 opus 编码器的最终状态	186
24.2.6 opus 编码	186
24.2.7 释放 opus 编码器对象	187
24.3 Opus 编码器示例代码	187
24.3.1 关键说明	187
24.3.2 示例代码	187
24.4 操作说明	199
24.4.1 下载 Cool Edit Pro 工具	199
24.4.2 网络调试助手设置	200
24.4.3 打开配置	200
24.4.4 运行现象	200
25 EasyFlash	201
25.1 EasyFlash 简介	201
25.2 EasyFlash Related API	201
25.2.1 easyflash 初始化	201

25.2.2 获得 easyflash 环境变量	201
25.2.3 将数据写入到环境变量中	201
25.2.4 保存数据到 flash	202
25.3 EasyFlash 示例代码	202
25.3.1 关键说明	202
25.3.2 示例代码	202
25.4 操作说明	205
25.4.1 打开配置	205
25.4.2 运行现象	206
26 Voice Changer	207
26.1 Voice Changer 简介	207
26.2 Voice Changer Related API	207
26.2.1 voice changer 初始化	207
26.2.2 退出 voice changer	207
26.2.3 开始 voice changer	207
26.2.4 停止 voice changer	208
26.2.5 设置 voice changer 变声功能标志	208
26.2.6 voice changer 获取 mic 数据	208
26.2.7 设置消耗数据的长度	208
26.2.8 处理数据	208
26.3 Voice Changer 示例代码	209
26.3.1 关键说明	209
26.3.2 示例代码	209
26.4 操作说明	217
26.4.1 运行现象	217
27 图像传输	218
27.1 图像传输简介	218
27.2 图像传输 Related API	218
27.2.1 打开 video_transfer	219
27.2.2 关闭 video_transfer	219
27.2.3 设置摄像头的参数	219

27.2.4 打开获取 jpeg 帧的功能	220
27.2.5 关闭获取 jpeg 帧的功能	220
27.2.6 获取 jpeg 帧的数据	220
27.3 图像传输的示例代码	220
27.3.1 关键说明	220
27.3.2 示例代码	221
27.4 操作说明	225
27.4.1 下载 PC 调试工具	226
27.4.2 启动 softap	226
27.4.3 UDP 传输测试	226
27.4.4 TCP 传输测试	227
27.4.5 web camera 浏览器实时显示视频	227
28 Qspi Dcache 模式	229
28.1 Qspi Dcache 简介	229
28.2 Qspi Dcache Related API	229
28.2.1 初始化 qspi 为 dcache 模式	229
28.2.2 启动 qspi 功能	229
28.2.3 停止 qspi 功能	230
28.3 Qspi Dcache 示例代码	230
28.4 操作说明	237
28.4.1 运行现象	237
28.5 注意事项	237
29 BLE	238
29.1 BLE 简介	238
29.2 BLE Related API (通用)	238
29.2.1 启动 ble 协议栈	238
29.2.2 设置 write callback	238
29.2.3 设置 read callback	238
29.2.4 设置 event callback	238
29.3 BLE Related API (slave)	239
29.3.1 开始广播	239

29.3.2 关闭广播	239
29.3.3 发送数据	239
29.3.4 断开连接	240
29.4 BLE 结构体说明	240
29.4.1 广播参数	240
29.5 BLE 示例代码	240
29.5.1 关键说明	240
29.5.2 示例代码	241
29.6 操作说明	252
29.6.1 数据交互	252
30 USB 充电模式	255
30.1 USB 充电模式简介	255
30.2 USB 充电模式 Related API	255
30.2.1 vbat adc 校准第一步	255
30.2.2 充电参数校准	255
30.2.3 开始充电模式	256
30.2.4 停止充电模式	256
30.3 示例代码	256
30.3.1 关键说明	256
30.4 操作说明	257
30.4.1 运行现象	257
31 EZ_CONFIG 配网	259
31.1 EZ_CONFIG 简介	259
31.2 EZ_CONFIG Related API	259
31.2.1 开始 EZ_CONFIG	259
31.2.2 获取 EZ_CONFIG 状态	259
31.2.3 获取 EZ_CONFIG 解码结果	260
31.3 使用说明	260
32 BLE 配网	261
32.1 BLE 配网简介	261
32.2 BLE 配网 Related API	261



32.3 使用说明.....	261
33 AP 配网.....	262
33.1 AP 配网简介.....	262
33.2 AP 配网 Related A.....	262
33.3 使用说明.....	262

1 ADC

1.1 ADC简介

BK7251具有多路通用ADC检测模块，输出精度为10-16bit，可以支持单步及连续等操作模式。电压检测范围为0 ~ 2.4v, ADC channel如下：

通道	描述
0	检测vbat引脚电压，读取值为vbat电压值的1/2
1	检测gpio4引脚电压
2	检测gpio5引脚电压
3	检测gpio23引脚电压(与JTAG引脚复用)
4	检测gpio2引脚电压
5	检测gpio3引脚电压
6	检测gpio12引脚电压
7	检测gpio13引脚电压

Note: 其中ADC通道3与JTAG复用，如果需要用到ADC channel 3需要将JTAG功能关闭，ADC通道对应GPIO需要以上表格为主。

1.2 ADC Related API

ADC相关接口参考\beken378\func\saradc_intf.h，相关接口如下：

函数	描述
saradc_work_create()	创建adc检测线程
adc_obj_init()	配置adc通道以及回调函数
adc_obj_start()	启动adc检测功能
adc_obj_stop()	关闭adc检测功能

1.2.1 adc通用结构体说明

ADC_OBJ: adc对象的结构体说明

user_data	用户数据
channel	adc通道
cb	回调函数
next	指向下一个adc对象的结构体

1.2.2 创建adc检测线程

```
void saradc_work_create(UINT32 scan_interval_ms);
```

参数	描述
----	----

scan_interval_ms	adc扫描间隔
返回	无

1.2.3 配置adc检测通道及回调函数

```
void adc_obj_init(ADC_OBJ* handle, adc_obj_callback cb, UINT32 channel, void *user_data);
```

参数	描述
handle	adc检测通道的结构体，参数包括通道号，回调函数等
adc_obj_callback_cb	读取电压值后的回调函数，用来对读取结果进行处理。该回调函数有两个参数：1、new_mv 读取的电压值，单位为mv；2、user_data:对应adc_obj_init()传递的参数user_data
channel	电压检测通道，0-7
user_data	用户数据
返回	无

1.2.4 启动adc

```
int adc_obj_start(ADC_OBJ* handle);
```

参数	描述
handle	adc检测通道的结构体
返回	0: 成功; -1: 失败

1.2.5 关闭adc

```
int adc_obj_stop(ADC_OBJ* handle);
```

参数	描述
handle	adc检测通道的结构体
返回	0: 成功; -1: 失败

1.3 ADC示例代码

ADC示例代码参考test\adc_test.c，在test_config.h打开宏定义：ADC_TEST后，即可看到电池电量的打印信息，在串口输入命令adc_channel_test <start> <channel>，并在对应channel的引脚上接上输入电压，即可看到通道channel的电量打印。对于通道0，检测的电压值需要乘2，才是电池实际电压值。

```
/*
* 程序清单： 这是一个简单ADC程序使用例程，打开宏定义CONFIG_ADC_TEST，输入命令可以看到
电压打印信息。
* 命令调用格式： 开始检测： adc_channel_test start channel
                  结束检测： adc_channel_test stop
* 程序功能： 输入开始检测命令后，可以看到打印对应adc通道测量的电压值。
*/

#include "include.h"
#include "arm_arch.h"
#include "error.h"
#include "include.h"
#include <rthw.h>
#include <rtthread.h>
#include <rtdevice.h>
#include <stdint.h>
#include <stdlib.h>
#include <finsh.h>
#include <rtddef.h>
#include "saradc_intf.h"
#include "sys_ctrl_pub.h"

#define CONFIG_ADC_TEST
#ifdef CONFIG_ADC_TEST
static ADC_OBJ test_adc;

/****channel 1 - 7****/
static void adc_detect_callback(int new_mv, void *user_data)
{
    static int cnt = 0;
    test_adc.user_data = (void*)new_mv;

    if(cnt++ >= 100)
    {
        cnt = 0;
        rt_kprintf("adc channel%d voltage:%d,%x\r\n",test_adc.channel,new_mv,test_adc.user_data);
    }
}
```

```
void adc_channel_test(int argc,char *argv[])
{
    int channel;

    if (strcmp(argv[1], "start") == 0)
    {
        if(argc == 3)
        {
            channel = atoi(argv[2]);
            rt_kprintf("---adc channel:%d---\r\n",channel);
            saradc_work_create(20);
            adc_obj_init(&test_adc, adc_detect_callback, channel, &test_adc);
            adc_obj_start(&test_adc);
        }
        else
        {
            rt_kprintf("input param error\r\n");
        }
    }
    if(strcmp(argv[1], "stop") == 0)
    {
        adc_obj_stop(&test_adc);
    }
}

MSH_CMD_EXPORT(adc_channel_test,adc test);
#endif
```

2 PWM

2.1 PWM简介

BK7251具有6路PWM输出，每一路的周期及占空比都可以单独配置。

通道	描述
0	对应gpio6引脚
1	对应gpio7引脚
2	对应gpio8引脚
3	对应gpio9引脚
4	对应gpio24引脚
5	对应gpio26引脚

2.2 PWM Related API

pwm相关接口参考beken378\func\user_driver\BkDriverPwm.h，相关接口如下：

函数	描述
bk_pwm_initialize()	PWM初始化
bk_pwm_start()	启动PWM功能
bk_pwm_stop()	停止PWM功能

2.2.1 pwm枚举类型说明

bk_pwm_t:

BK_PWM_0	pwm0
BK_PWM_1	pwm1
BK_PWM_2	pwm2
BK_PWM_3	pwm3
BK_PWM_4	pwm4
BK_PWM_5	pwm5

2.2.2 初始化pwm

```
OSStatus bk_pwm_initialize(bk_pwm_t pwm, uint32_t cycle, uint32_t duty_cycle);
```

参数	描述
pwm	选择的pwm通道：0 ~ 5
cycle	设置pwm的方波周期
duty_cycle	设置pwm的占空值

返回	0: 成功; -1: 错误
----	---------------

2.2.3 启动pwm功能

```
OSStatus bk_pwm_start(bk_pwm_t pwm);
```

参数	描述
pwm	选择的pwm通道: 0 ~ 5
返回	0: 成功; -1: 错误

2.2.4 停止pwm功能

```
OStatus bk_pwm_stop(bk_pwm_t pwm);
```

参数	描述
pwm	选择的pwm通道: 0~5
返回	0: 成功; -1: 错误

2.3 PWM示例代码

示例代码参考test\pwm_test.c, 在test_config.h打开宏定义: PWM_TEST, 代码运行后, 串口输入命令 : pwm_test <channel> <duty_cycle> <cycle> 即可在对应的pin脚上检测到波形。

```
/*
 * 程序清单: 这是一个简单PWM使用例程, 打开宏定义#define CONFIG_PWM_TEST, 开启测功能。
 * 命令调用格式: pwm_test 1 8000 16000
 * 程序功能: 输入命令可以检测到对应的PWM通道上输出PWM波形。
 */
#include "rtos_pub.h"
#include "BkDriverPwm.h"
#include "pwm_pub.h"
#include "error.h"
#include <stdint.h>
#include <stdlib.h>
#include <finsh.h>

#define CONFIG_PWM_TEST
#ifdef CONFIG_PWM_TEST

static void pwm_test(int argc, char *argv[])
```

```
{
    UINT32 channel,duty_cycle,cycle;
    if(argc != 4)
        return;
    channel  = atoi(argv[1]);
    duty_cycle  = atoi(argv[2]);
    cycle      = atoi(argv[3]);
    if(cycle < duty_cycle) {
        rt_kprintf("pwm param error: end < duty\r\n");
        return;
    }
    rt_kprintf("---pwm %d test--- \r\n",channel);
    bk_pwm_initialize(channel, cycle, duty_cycle);    /*pwm 模块初始化，设置对应通道的占空比*/
    bk_pwm_start(channel);                          /*启动pwm */
    rt_thread_delay(100);
    bk_pwm_stop(channel);                          /*关闭pwm */
    rt_kprintf("---pwm test stop---\r\n");
}
MSH_CMD_EXPORT(pwm_test,pwm test);
#endif
```

2.4 操作说明

2.4.1 打开配置

示例代码参考test\pwm_test.c，打开宏定义：CONFIG_PWM_TEST，编译完成后，将固件下载至设备。

2.4.2 运行现象

串口输入命令：pwm_test <channel> <duty_cycle> <cycle>即可在对应的gpio上检测到波形。分别输入channel:1~5的命令，即可用逻辑分析仪看到相应的gpio输出周期性的方波。如下图所示：



图2.4.2-1 5个pwm channel同时输出波形图

2.5 注意事项

- pwm channel0 已经被cpu用来做timer，所以其pwm功能不能使用。
- pwm输出的时候，其时钟源选择的是26M。

3 Audio

3.1 Audio简介

BK7251芯片具有audio播放以及录音功能，利用麦克风采集音频数据，通过dac输出声音。

3.2 Audio Related API

audio相关接口参考\drivers\audio\audio_device.h，相关接口如下：

函数	描述
audio_device_init()	找到sound 和 mic设备，将这两个设备挂在总线上
audio_device_mic_open()	打开mic
audio_device_mic_set_channel()	设置adc通道
audio_device_mic_set_rate()	设置adc采样率
audio_device_mic_read()	mic采集声音数据
audio_device_mic_close()	关闭mic设备
audio_device_open()	打开dac
audio_device_set_rate()	设置dac采样率
audio_device_set_volume()	设置dac音量 0 ~ 16
audio_device_write()	audio播放音频数据
audio_device_close()	关闭dac

3.2.1 audio通用结构体说明

audio_device:

Struct	rt_device *snd	sound 设备的句柄
struct	rt_device *mic	mic 设备的句柄
struct	rt_mempool mp	memory pool
int	state	audio状态
void (*evt_handler)(void *parameter, int state)		事件中断处理
Void	*parameter	audio参数

3.2.2 audio设备初始化

初始化audio设备包括需找设备“sound”和“mic”句柄，以及分配内存。由于设备已经在开机的时候自动注册了这两个设备，所以只需要在初始化的时候找到这个设备，拿到设备句柄即可。

```
int audio_device_init(void)
```

参数	描述
void	空
返回	RT_EOK: 成功; 错误码: 失败

3.2.3 打开mic

打开 mic device,设置成只读模式。

```
void audio_device_mic_open(void);
```

参数	描述
void	空
返回	空

3.2.4 设置mic数据采集通道

```
void audio_device_mic_set_channel(int channel);
```

参数	描述
channel	adc通道
返回	空

3.2.5 设置mic采样率

设置mic 采集数据通道,采样率分别有48k,44.1k,32k,16k,8k等。

```
void audio_device_mic_set_rate(int sample_rate);
```

参数	描述
sample_rate	adc通道
返回	空

3.2.6 采集mic声音数据

```
int audio_device_mic_read(void *buffer, int size);
```

参数	描述
buffer	mic读取的buffer
size	mic读取数据长度
返回	length:返回读取数据的长度

3.2.7 关闭mic

```
void audio_device_mic_close(void);
```

参数	描述
void	空
返回	空

3.2.8 打开audio设备

```
void audio_device_open(void);
```

参数	描述
void	空
返回	空

3.2.9 设置dac采样率

设置dac的采样率，采样率分别有48k、44.1k、32k、16k、8k等。

```
void audio_device_set_rate(int sample_rate);
```

参数	描述
sample_rate	采样率
返回	空

3.2.10 设置dac音量

```
void audio_device_set_volume(int volume);
```

参数	描述
volume	音量大小
返回	空

3.2.11 获取dac数据，播放音频

```
void audio_device_write(void *buffer, int size);
```

参数	描述
buffer	audio播放音频的数据
size	audio播放数据的长度
返回	空

3.2.12 关闭dac

```
void audio_device_close(void);
```

参数	描述
void	空
返回	空

3.3 Audio示例代码

示例代码参考test\mic_record.c，打开宏定义：MICPHONE_TEST，测试录音和播放功能必须关闭混音的宏定义CONFIG_SOUND_MIXER，串口输入命令record_and_play可以听到audio输出端口播放之前录的声音。

3.3.1 关键说明

• Audio宏定义

#define	TEST_BUFF_LEN	80*1024	/*buffer大小
#define	READ_SIZE	2048	/*读取buffer大小

3.3.2 示例代码

```
/*
 * 程序清单： 这是一个录音以及audio播放使用例程
 * 命令调用格式： record_and_play vad_on work_mode sample_rate
 * 程序功能： 设备录音完后可以通过audio播放。
 */

#include <rtthread.h>
#include <rtdevice.h>
#include <finsh.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "board.h"
#include "audio_device.h"
#include "vad.h"
#include "test_config.h"
#ifdef MICPHONE_TEST

static uint8_t *test_buf;
```

```
#define TEST_BUFF_LEN 80*1024
#define READ_SIZE 2048

/*****

    argv[1]: vad  off:0  on:1
    argv[2]: work_mode
    0: one microphone(mic1)
    1: two microphones( mic1 && mic2 ) -- use two microphone data
    2: two microphones( mic1 && mic2 ) -- only use mic1 data
    3: two microphones( mic1 && mic2 ) -- only use mic2 data
    argv[3]: sample_rate  8000/16000

    cmd format :record_and_play 0/1 0/1/2/3 8000/16000
    eg: record_and_play 1 0 8000

*****/

void record_and_play(int argc,char *argv[])
{
    int mic_read_len = 0;
    int actual_len,i;
    int dac_wr_len=0;
    uint16_t *buffer = NULL;
    uint16_t read_size;
    int sample_rate;
    int index;
    uint16_t *ptr;
    int work_mode;
    int vad_on;
    if(argc<4)
    {
        rt_kprintf("parameter error\r\n");
    }
    vad_on = atoi(argv[1]);
    work_mode = atoi(argv[2]);
    sample_rate = atoi(argv[3]);

    test_buf = sdram_malloc(TEST_BUFF_LEN);
    if(test_buf == NULL)
    {
```

```
    rt_kprintf("===not enough memory===\r\n");
    return;
}

audio_device_init();

audio_device_mic_open();

if(work_mode == 0)
{
    audio_device_mic_set_channel(1);
    read_size = READ_SIZE;
}
else
{
    audio_device_mic_set_channel(2);
    read_size = READ_SIZE * 2;
}

audio_device_mic_set_rate(sample_rate);

if (vad_on)
{
    rt_kprintf("Vad is ON !!!!!!!\r\n"); /*进入vad检测*/
    wb_vad_enter();
}

while(1)
{
    rt_thread_delay(10);
    if(mic_read_len > TEST_BUFF_LEN - READ_SIZE)
        break;
    actual_len = audio_device_mic_read(test_buf+mic_read_len,read_size);
    mic_read_len += actual_len;
    if(vad_on)
    {
        if(wb_vad_entry((char*)test_buf+mic_read_len, 320))/*vad process*/
        {
            rt_kprintf("-----vad end-----\r\n");
        }
    }
}
```

```
        break;
    }
}

rt_kprintf("mic_read_len is %d\r\n", mic_read_len);
audio_device_mic_close();
if (vad_on)
{
    wb_vad_deinit();          /*关闭vad检测*/
}

audio_device_open();
audio_device_set_rate(sample_rate);

while(1)
{
    buffer = (uint16_t *)audio_device_get_buffer(RT_NULL);
    if(dac_wr_len >= mic_read_len)
    {
        audio_device_put_buffer(buffer);
        break;
    }

    memcpy(buffer, test_buf + dac_wr_len, read_size);
    dac_wr_len += read_size;

    switch(work_mode)
    {
        case 0:
            //expand to 2 channels
            ptr = (uint16_t *)((uint8_t *)buffer + read_size * 2);
            ptr -= 1;
            for (index = 1; index < read_size / 2; index++)
            {
                *ptr = *(ptr - 1) = buffer[read_size / 2 - index];
                ptr -= 2;
            }
            audio_device_write((uint8_t *)buffer, read_size * 2);
            break;
        case 1:
```



```
        audio_device_write((uint8_t *)buffer, read_size);
        break;
    case 2:
        ptr = (uint16_t*)buffer;
        for(index = 0;index < read_size/2;)
        {
            ptr[index+1] = ptr[index];
            index += 2;
        }
        audio_device_write((uint8_t *)buffer, read_size);
        break;
    case 3:
        ptr = (uint16_t*)buffer;
        for(index = 0;index < read_size/2;)
        {
            ptr[index] = ptr[index+1];
            index += 2;
        }
        audio_device_write((uint8_t *)buffer, read_size);
        break;
    default:
        break;
}

}

audio_device_close();

if(test_buf)
    sdram_free(test_buf);
}

MSH_CMD_EXPORT(record_and_play, record play);

#endif
```

4 Button

4.1 Button简介

按键功能包含有按键长按，短按，双击等功能。

4.2 Button Related API

button相关接口函数参考samples\key\multi_button.h，相关接口如下：

函数	描述
button_init()	按键初始化
button_attach()	配置回调函数
button_start()	开始按键工作，将handle 加入到工作清单
button_stop()	结束按键工作

4.2.1 button初始化

```
void button_init(BUTTON_S* handle, uint8_t(*pin_level)(), uint8_t active_level,void *user_data);
```

参数	描述
BUTTON_S* handle	按键句柄
uint8_t(*pin_level)	读取HAL gpio
uint8_t active_level	gpio level
void *user_data	用户数据
返回	空

4.2.2 配置回调函数

```
void button_attach(BUTTON_S* handle, PRESS_EVT event, btn_callback cb);
```

参数	描述
BUTTON_S* handle	button 句柄
PRESS_EVT event	触发事件类型
btn_callback cb	回调函数
返回	空

4.2.3 开始button工作

```
int button_start(BUTTON_S* handle);
```

参数	描述
BUTTON_S* handle	button 句柄
返回	0: 成功; 其他: 失败

4.2.4 结束button工作

```
void button_stop(BUTTON_S* handle);
```

参数	描述
BUTTON_S* handle	button 句柄
返回	空

4.3 Button示例代码

4.3.1 关键说明

- **Button枚举类型**
触发按键的事件类型:

```
typedef enum {  
    PRESS_DOWN = 0,      //按键按下  
    PRESS_UP,            //不按  
    PRESS_REPEAT,        //重复按  
    SINGLE_CLICK,        //单击  
    DOUBLE_CLICK,        //双击  
    LONG_PRESS_START,    //长按开始  
    LONG_PRESS_HOLD,     //保持长按  
    NUMBER_OF_EVENT,     //按键事件数量  
    NONE_PRESS           //没有按按键  
}PRESS_EVT;
```

4.3.2 示例代码

在sample_config.h 打开宏定义 BUTTON_TEST。

```
/*  
 * 程序清单: 这是一个按键使用例程  
 * 命令调用格式: button_test gpio, 需要用到哪一个gpio作为按键就输入该gpio的序号  
 * 程序功能: 通过短按, 长按, 双击相应的按键, 会在串口打印出相应的状态。
```

```
*/

#include "include.h"
#include "typedef.h"
#include "arm_arch.h"
#include "gpio_pub.h"
#include "gpio_pub.h"
#include "uart_pub.h"
#include "multi_button.h"
#include "bk_rtos_pub.h"
#include "error.h"
#include "sys_ctrl_pub.h"

#define BUTTON_TEST
#ifdef BUTTON_TEST

#define TEST_BUTTON 4
static beken_timer_t g_key_timer;

static void button_short_press(void *param)
{
    rt_kprintf("button_short_press\r\n");
}

static void button_double_press(void *param)
{
    rt_kprintf("button_double_press\r\n");
}

static void button_long_press_hold(void *param)
{
    rt_kprintf("button_long_press_hold\r\n");
}

static uint8_t key_get_gpio_level(BUTTON_S*handle)
{
    return bk_gpio_input((uint32_t)handle->user_data);
}

BUTTON_S gpio_button_test[2];

void button_test(int argc, char *argv[])
{

```

```
OSStatus result;

int gpio ;
if(argc != 2)
{
    rt_kprintf("---! ! !param error---\r\n");
}
else
{
    gpio = atoi(argv[1]);

    rt_kprintf("---gpio%d as button : test start---n",gpio);

    if((gpio >=40)|| (gpio >= 40))
    {
        rt_kprintf("---! ! !gpio error---\r\n");
        return;
    }

    /*gpio key config:input && pull up*/
    gpio_config(gpio,GMODE_INPUT_PULLUP);

    button_init(&gpio_button_test[0], key_get_gpio_level, 0,(void*)gpio);    /*初始化按键*/

    /*配置按键事件的回调函数*/
    button_attach(&gpio_button_test[0], SINGLE_CLICK,button_short_press);
    button_attach(&gpio_button_test[0], DOUBLE_CLICK,button_double_press);
    button_attach(&gpio_button_test[0], LONG_PRESS_HOLD,button_long_press_hold);

    button_start(&gpio_button_test[0]);    /*开始按键检测*/
    result = bk_rtos_init_timer(&g_key_timer,    /*初始化按键状态检测时钟*/
                                TICKS_INTERVAL,
                                button_ticks,
                                (void *)0);

    ASSERT(kNoErr == result);

    result = bk_rtos_start_timer(&g_key_timer);    /*开启时钟*/
    ASSERT(kNoErr == result);
}
}
```

```
MSH_CMD_EXPORT(button_test,button test);
```

```
// eof
```

```
#endif
```

4.4 操作说明

Button示例代码参考\samples\key\button_test.c，使能后支持按键的短按、双击、长按功能。

4.4.1 打开配置

打开宏定义：BUTTON_TEST，编译下载后，调试串口输入button_test 3，使能S3按键，设备log如下：

```
---gpio3 as button : test start---  
msh />button_short_press
```

4.4.2 按键测试

- 短按测试

连续三次短按S3按键(大于300ms小于1s)，设备log如下：

```
---gpio3 as button : test start---  
msh />button_short_press  
button_short_press  
button_short_press  
button_short_press
```

- 长按测试

长按S3按键(大于1s)，设备log如下：

```
msh />button_test 3  
---gpio3 as button : test start---  
msh />button_short_press  
button_short_press  
button_short_press  
button_short_press  
button_long_press_hold  
button_long_press_hold  
button_long_press_hold  
button_long_press_hold
```

- 双击测试

双击S3按键，设备log如下：

```
msh />button_test 3
---gpio3 as button : test start---
msh />button_short_press
button_short_press
button_short_press
button_short_press
button_long_press_hold
button_long_press_hold
button_long_press_hold
button_long_press_hold
button_double_press
button_double_press
button_double_press
```

5 I2C总线

5.1 I2C简介

BK7251芯片上设有I2C模块，但是在RT_thread 实时操作系统中带有模拟I2C总线的驱动文件，而且已经关联到 RT-thread操作系统的标准设备操作函数集了。底层驱动文件利用MCU的GPIO模拟I2C总线时序，并不是用MCU的硬件I2C接口，I2C接口两个信号线SCL,SDA分别对应gpio2,gpio3。

5.2 模拟I2C Related API

I2C相关接口参考\rt-thread\components\drivers\include\drivers\i2c.h

函数	描述
rt_i2c_bus_device_find()	寻找总线
rt_i2c_transfer()	读/写数据

5.2.1 寻找总线获取设备句柄

```
struct rt_i2c_bus_device *rt_i2c_bus_device_find (const char *bus_name);
```

参数	描述
const char *bus_name	总线名称
返回	i2c总线句柄

5.2.2 对从设备的读写数据

```
rt_size_t rt_i2c_transfer(struct rt_i2c_bus_device *bus,  
                          struct rt_i2c_msg      msgs[],  
                          rt_uint32_t            num);
```

参数	描述
struct rt_i2c_bus_device *bus	获取的总线句柄，根据注册的总线句柄进行一系列的底层驱动操作
struct rt_i2c_msg msgs[]	包含从设备地址，读/写标志，读/写buffer等
rt_uint32_t num	传输数据次数
返回	RT_EOK: 成功；其他：失败

参数类型

rt_i2c_bus_device:	
rt_device parent	parent

<code>rt_i2c_bus_device_ops* ops</code>	i2c操作
<code>rt_uint16_t addr</code>	从设备地址
<code>rt_uint16_t flags</code>	读写标志位
<code>rt_mutex lock</code>	锁定结构体
<code>rt_uint32_t timeout</code>	超时标志
<code>rt_uint32_t retries</code>	重复次数
<code>void *priv</code>	设备私有数据
 <code>rt_i2c_msg:</code>	
<code>rt_uint16_t addr</code>	地址
<code>rt_uint16_t flags</code>	读写标志
<code>rt_uint16_t len</code>	buffer 长度
<code>rt_uint8_t *buf</code>	传送数据的buffer

5.3 模拟I2C总线示例代码

软件模拟I2C总线设备示例代码位于\test\i2c_rtt_test.c。打开宏定义：
I2C_RTT_TEST，开启i2c功能测试。

Note: 芯片中没有作为I2C从设备的器件，所以在测试其准确性的时候需要外挂一个EEPROM。

5.3.1 关键说明

• I2C宏定义

<code>#define</code>	<code>RT_USING_I2C</code>	使用MCU的I2C设备
<code>#define</code>	<code>RT_USING_I2C_BITOPS</code>	MCU的GPIO模拟I2C总线
<code>#define</code>	<code>BEKEN_USING_IIC</code>	使用I2C驱动

5.3.2 对从设备的读写数据

```
/*
 * 程序清单： 这是一个简单的I2C驱动程序使用例程，从设备使用的是地址为 0x50的EEPROM
 * 例程写出了I2C总线中主设备对从设备的读取操作，
 * 命令调用格式： i2c_test_rtt
 * 程序功能： 7251通过I2C总线对EEPROM的读写控制，来写入或者读取EEPROM的数据，测试过程中
               需要外挂一个eeprom。
 */
#include <rtthread.h>
#include <rtdevice.h>
#include "finsh.h"
```

```
#include <rthw.h>
#include <string.h>
#include <time.h>
#include <drv_iic.h>
#define eeprom_addr 0x50; /* 1010A2A1A0 -R/W */
/*****I2C sample*****/
static int i2c_test_rtt(int argc, char *argv)
{
    const char *i2c_bus_device_name = "i2c";
    struct rt_i2c_bus_device *i2c_device;
    struct rt_i2c_msg msgs[2];

    rt_uint8_t buffer1[2];
    rt_uint8_t buffer2[3];
    rt_size_t i, ret;
    rt_uint8_t ret1;

    ret1 = iic_bus_attach( ); /* gpio init and add bus */
    rt_kprintf("iic_bus_attach ret:%d\n", ret1);
    i2c_device = rt_i2c_bus_device_find(i2c_bus_device_name);
    if (i2c_device == RT_NULL)
    {
        rt_kprintf("i2c bus device %s not found!\n", i2c_bus_device_name);
        return -RT_ENOSYS;
    }
    else
    {
        rt_kprintf("find i2c success\n");
    }
    /*****step 1: read out.*****/
    buffer1[0] = 0;
    msgs[0].addr = eeprom_addr;
    msgs[0].flags = RT_I2C_WR; /* write to slave */
    msgs[0].buf = buffer1; /* eeprom offset. */
    msgs[0].len = 1;

    msgs[1].addr = eeprom_addr;
    msgs[1].flags = RT_I2C_RD; /* Read from slave */
    msgs[1].buf = buffer2;
    msgs[1].len = sizeof(buffer2);
```

```
if ( rt_i2c_transfer(i2c_device, msgs, 2) !=2 )      /* write or read data */
{
    rt_kprintf("--read eeprom fail--\r\n");
}
else
{
    rt_kprintf("--read eeprom sucess--\r\n");
}
for(i=0; i<sizeof(buffer2); i++)
{
    rt_kprintf("%02X ", buffer2[i]);
}

rt_thread_delay(rt_tick_from_millisecond(50));
rt_kprintf("\r\n---read test over---\r\n");
/*****step 2: write back. *****/

for(i=0; i<sizeof(buffer2); i++)
{
    buffer2[i] = buffer2[i]+5 ;
}
msgs[0].addr = eeprom_addr;
msgs[0].flags = RT_I2C_WR;          /* write to slave */
msgs[0].buf = buffer1;             /* eeprom offset. */
msgs[0].len = 1;

msgs[1].addr = eeprom_addr;
msgs[1].flags = RT_I2C_WR;
msgs[1].len = sizeof(buffer2);
if ( rt_i2c_transfer(i2c_device, msgs, 2) !=2 )
{
    rt_kprintf("---write eeprom fail---\r\n");
}
else
{
    rt_kprintf("---write eeprom sucess---\r\n");
}

rt_thread_delay(rt_tick_from_millisecond(50));
```

```
for(i=0; i<msgs[1].len; i++)
{
    rt_kprintf("%02X ", buffer2[i]);
}
rt_kprintf("\r\n ---write test over---\r\n");

/*****step 3: read out. *****/

buffer1[0] = 0;
msgs[0].addr = eeprom_addr;
msgs[0].flags = RT_I2C_WR;           /* write to slave */
msgs[0].buf = buffer1;              /* eeprom offset. */
msgs[0].len = 1;

msgs[1].addr = eeprom_addr;
msgs[1].flags = RT_I2C_RD;           /* Read from slave */
msgs[1].buf = buffer2;
msgs[1].len = sizeof(buffer2);

if ( rt_i2c_transfer(i2c_device, msgs, 2) !=2 )
{
    rt_kprintf("---re-read eeprom fail---\r\n");
}
else
{
    rt_kprintf("---re-read eeprom sucess---\r\n");
}
rt_thread_delay(rt_tick_from_millisecond(50));

for(i=0; i<msgs[1].len; i++)
{
    rt_kprintf("%02X ", buffer2[i]);
}
rt_kprintf("\r\n ---re-read test over---\r\n");
return 0;
}

#ifdef RT_USING_FINSH
#include <finsh.h>
FINSH_FUNCTION_EXPORT_ALIAS(i2c_test_rtt, __cmd_i2c_test_rtt, i2c test rtt cm);
#endif
```

5.4 硬件I2C总线示例代码

硬件I2C总线设备示例代码位于\test\i2_test.c。打开宏定义：I2C_TEST，开启i2c功能测试。

Note: 芯片中没有作为I2C从设备的器件，所以在测试其准确性的时候需要外挂一个EEPROM。

5.4.1 硬件I2C接口说明

函数	描述
i2c_device_init	初始化I2C
I2C_write_eeprom	写eeprom
I2C_read_eeprom	读eeprom
i2c_device_deinit	关闭I2C

5.4.2 对从设备的读写数据

```
/*
 * 程序清单： 这是一个硬件I2C驱动程序使用例程，从设备使用的是地址为 0x21的EEPROM
 * 例程写出了I2C总线中主设备对从设备的读写操作，
 * 命令调用格式： i2c_test_eeprom
 * 程序功能： 7251通过I2C总线对EEPROM的读写控制，来写入或者读取EEPROM的数据，测试过程中
               需要外挂一个eeprom。
 */
#include "include.h"
#include <rtthread.h>
#include <rthw.h>
#include <rtdevice.h>
#include <string.h>
#include "icu_pub.h"
#include "i2c_pub.h"
#include "drv_model_pub.h"
#include "target_util_pub.h"
#include "test_config.h"

#ifdef I2C_TEST

#define I2C_EEPROM_DEBUG
#ifdef I2C_EEPROM_DEBUG
#define I2C_EEPROM_PRT      os_printf
```

```
#define I2C_EEPROM_WARN    warning_prf
#define I2C_EEPROM_FATAL  fatal_prf
#else
#define I2C_EEPROM_PRT     null_prf
#define I2C_EEPROM_WARN    null_prf
#define I2C_EEPROM_FATAL  os_printf
#endif

#define I2C1                0
#define I2C2                1
#define I2C_DEV_ID         I2C1
#define I2C_SALVE_ID        0x21

static DD_HANDLE i2c_hdl;

static void i2c_device_init()
{
    unsigned int oflag,status;
    oflag = 0;
    #if I2C_DEV_ID
        i2c_hdl = ddev_open(I2C2_DEV_NAME, &status, oflag);
    #else
        i2c_hdl = ddev_open(I2C1_DEV_NAME, &status, oflag);
    #endif
}

static void i2c_device_deinit()
{
    ddev_close(i2c_hdl);
}

/*****
 * I2C1_write_eeprom
 * Description: I2C1 write FT24C02 eeprom
 * Parameters:  op_addr: operate address
 *              pData: data point
 *              len: data len
 * return:      unsigned long
 *****/
```

```
* error:      none
*/

static unsigned long I2C_write_eeprom(unsigned char op_addr, unsigned char *pData, unsigned char
len)
{
    unsigned char i;
    unsigned int status;
    I2C_OP_ST i2c_op;

    I2C_EEPROM_PRT("----- I2C1_write_eeprom start -----\\r\\n");

    i2c_op.op_addr = op_addr;
    i2c_op.salve_id = I2C_SALVE_ID;

    do
    {
        status = ddev_write(i2c_hdl, pData, len, (unsigned long)&i2c_op);
    } while (status != 0);

    I2C_EEPROM_PRT("----- I2C1_write_eeprom over -----\\r\\n");
    return 0;
}

/*****
* I2C1_read_eeprom
* Description: I2C1 read FT24C02 eeprom
* Parameters:  op_addr: operate address
*              pData: data point
*              len: data len
* return:      unsigned long
* error:      none
*/
static unsigned long I2C_read_eeprom(unsigned char op_addr, unsigned char *pData, unsigned char
len)
{
    unsigned char i;
    DD_HANDLE i2c_hdl;
    unsigned int status;
```

```
I2C_OP_ST i2c_op;

I2C_EEPROM_PRT("----- I2C1_read_eeprom start -----\\r\\n");

i2c_op.op_addr = op_addr;
i2c_op.salve_id = I2C_SALVE_ID;
do
{
    status = ddev_read(i2c_hdl, pData, len, (unsigned long)&i2c_op);
} while (status != 0);

for (i=0; i<8; i++)
{
    I2C_EEPROM_PRT("pData[%d] = 0x%x\\r\\n", i, pData[i]);
}

I2C_EEPROM_PRT("----- I2C1_read_eeprom over -----\\r\\n");
return status;
}

/*****
* I2C1_test_eeprom
* Description: I2C1 test FT24C02 eeprom
* Parameters: none
* return:      unsigned long
* error:       none
*/
static unsigned long i2c_test_eeprom(void)
{
    int i, j;
    DD_HANDLE i2c_hdl;
    unsigned char pReadData[8];
    unsigned char pWriteData[8];

    i2c_device_init();
    I2C_EEPROM_PRT("----- I2C1_test_eeprom start -----\\r\\n");

    for (j=0; j<100; j++)
    {
        delay_ms(100);
```



```
    for (i=0; i<8; i++)
    {
        pWriteData[i] = (i << 2) + 0x01 + j;
    }
    I2C_write_eeprom(0x00+j*8, pWriteData, 8);

    delay_ms(100);

    memset(pReadData, 0, 8);
    I2C_read_eeprom(0x00+j*8, pReadData, 8);

    if (memcmp(pReadData, pWriteData, 8) == 0)
    {
        os_printf("I2C_test_eeprom: memcmp %d ok!\r\n", j);
    }
    else
    {
        I2C_EEPROM_FATAL("I2C_test_eeprom: memcmp %d error!\r\n", j);
        for (i=0; i<8; i++)
        {
            I2C_EEPROM_FATAL("pReadData[%d]=0x%x, pWriteData[%d]=0x%x\r\n",
                              i, pReadData[i], i, pWriteData[i]);
        }
    }
}

I2C_EEPROM_PRT("----- i2c_test_eeprom over  ----- \r\n");
i2c_device_deinit();
return 0;
}

MSH_CMD_EXPORT(i2c_test_eeprom,i2c_test_eeprom);
```

6 I2S总线

6.1 I2S简介

BK7251芯片上设有I2S模块，I2S(Inter—IC Sound)总线是飞利浦公司为数字音频设备之间的音频数据传输而制定的一种总线标准，该总线专责于音频设备之间的数据传输，广泛应用于各种多媒体系统。I2S模块包含四根信号线，分别是I2S_CLK, I2S_SYNC,I2S_DIN,I2S_DOUT，对应gpio分别为gpio2, gpio3, gpio4, gpio5，I2S模块可分为I2S、Left justified、Right justified和2B+D等模式。

I2S_CLK:串行时钟信号,也称作BCLK,对应数字音频的每一位数，I2S_SYNC:采样率，I2S_DIN,I2S_DOUT分别为数据的输入和输出。

6.2 I2S Related API

I2S相关接口参考**beken378\driver\i2s \i2s.h**。

函数	描述
i2s_configure()	I2s模块初始化设置
i2s_transfer()	主/从设备发送接收数据

6.2.1 i2s通用结构体说明

i2s_trans_t:

* p_tx_buf	发送数据buffer
*p_rx_buf;	接收buffer
trans_done	传送数据完成标志位
tx_remain_data_cnt;	发送剩余数据
rx_remain_data_cnt	接收剩余数据

i2s_message:

* send_buf	发送数据buffer
send_len	发送长度
* recv_buf	接收数据buffer
recv_len	接收长度

6.2.2 i2s模块参数设置

```
i2s_configure(UINT32 fifo_level, UINT32 sample_rate, UINT32 bits_per_sample, UINT32 mode);
```

参数	描述
----	----

fifo_level	配置寄存器中的读写数据fifo水位
sample_rate	配置i2s模块采样率
bits_per_sample	位宽（每个声道的bit数）
mode	配置模式
返回	I2S_SUCCESS: 成功；其他：失败

6.2.3 i2s主从设备发送/接收数据

```
UINT32 i2s_transfer(UINT32 *i2s_send_buf , UINT32 *i2s_recv_buf, UINT32 count, UINT32 param );
```

参数	描述
i2s_send_buf	发送数据buffer
i2s_recv_buf	接收数据buffer
count	发送数据总长度
param	1: 主 0: 从
返回	0: 成功； 其他：失败

6.3 I2S示例代码

示例代码参考\test\i2s_test.c。打开宏定义：I2S_TEST，开启i2s功能测试。

6.3.1 关键说明

• I2S宏定义

#define	RT_USING_I2S	使用MCU的I2S模块
#define	BEKEN_USING_IIS	使用I2S驱动

• I2S工作模式的宏定义：

#define	I2S_MODE	(0 << 0)	I2S模式
#define	I2S_LEFT_JUSTIFIED	(1 << 0)	左对齐模式
#define	I2S_RIGHT_JUSTIFIED	(2 << 0)	右对齐模式
#define	I2S_RESERVE	(3 << 0)	保留
#define	I2S_SHORT_FRAME_SYNC	(4 << 0)	短帧同步
#define	I2S_LONG_FRAME_SYNC	(5 << 0)	长帧同步
#define	I2S_NORMAL_2B_D	(6 << 0)	正常2B+D模式
#define	I2S_DELAY_2B_D	(7 << 0)	延后2B+D模式
#define	I2S_LRCK_NO_TURN	(0 << 3)	lrck不反转
#define	I2S_SCK_NO_TURN	(0 << 4)	sck不反转
#define	I2S_MSB_FIRST	(0 << 5)	MSB先发送

#define I2S_SYNC_LENGTH_BIT	(8)	Sync长度（仅在长帧同步模式下有效）
#define I2S_PCM_DATA_LENGTH_BIT	(12)	D的长度（仅在2B+D模式下有效）

6.3.2 示例代码

```
/*
 * 程序清单： 这是一个简单的I2S驱动程序使用例程，两块demo板一个为主，一个为从设备，
 * 例程写出了i2s总线中主从设备接收/发送数据的操作，
 * 命令调用格式： i2s_test master/slave rate bit_length
 * 程序功能： 主从设备分别接收和发送数据，测试能否正常接受/发送数据，频率位宽是否能够达到要求。
 */

#include "include.h"
#include "arm_arch.h"
#include <rtthread.h>
#include <rthw.h>
#include <rtdevice.h>
#include <stdio.h>
#include <string.h>
#include <stdint.h>
#include <stdlib.h>
#include "typedef.h"
#include "icu_pub.h"
#include "i2s.h"
#include "i2s_pub.h"
#include "sys_ctrl_pub.h"
#include "drv_model_pub.h"
#include "mem_pub.h"
#include "sys_config.h"
#include "error.h"
#include "rtos_pub.h"

#define I2S_DATA_LEN      0x100

extern UINT32 i2s_configure(UINT32 fifo_level, UINT32 sample_rate, UINT32 bits_per_sample,
UINT32 mode);

volatile i2s_trans_t i2s_trans;
i2s_level_t i2s_fifo_level;
```

```
int i2s_test(int argc, char** argv)
{
    struct rt_device *i2s_device;
    struct i2s_message msg;
    uint32 i,rate,bit_length ;
    uint32 i2s_mode = 0;
    if(argc != 4)
    {
        rt_kprintf("---cmd error--\r\n");
        return RT_ERROR;
    }
    rate      = atoi(argv[2]);
    bit_length = atoi(argv[3]);

    msg.recv_len = I2S_DATA_LEN;
    msg.send_len = I2S_DATA_LEN;

    msg.recv_buf = rt_malloc(I2S_DATA_LEN * sizeof(msg.recv_buf[0]));
    if(msg.recv_buf == RT_NULL)
    {
        rt_kprintf("msg.recv_buf malloc failed\r\n");
    }
    //rt_kprintf("msg.recv_buf=%x\r\n",msg.recv_buf);

    msg.send_buf = rt_malloc(I2S_DATA_LEN * sizeof(msg.send_buf[0]));
    if(msg.send_buf == RT_NULL)
    {
        rt_kprintf("msg.send_buf malloc failed\r\n");
    }
    //rt_kprintf("msg.send_buf=%x\r\n",msg.send_buf);

    /* find device*/
    i2s_device = rt_device_find("i2s");
    if(i2s_device == RT_NULL)
    {
        rt_kprintf("---i2s device find failed---\r\n ");
        return 0 ;
    }

    /* init device*/
```

```
if(rt_device_init( i2s_device) != RT_EOK)
{
    rt_kprintf(" --i2s device init failed---\r\n ");
    return 0;
}

/* open audio , set fifo level set sample rate/datawidth */
i2s_mode = i2s_mode| I2S_MODE| I2S_LRCK_NO_TURN| I2S_SCK_NO_TURN|
I2S_MSB_FIRST| (0<<I2S_SYNC_LENGTH_BIT)| (0<<I2S_PCM_DATA_LENGTH_BIT);

/* write and recieve */
if(strcmp(argv[1], "master") == 0)
{
    rt_kprintf("---i2s_master_test_start---\r\n");

    if(msg.send_buf == NULL)

    {
        rt_kprintf("---msg.send_buf error --\r\n ");
        return 0;
    }

    for(i=0; i<I2S_DATA_LEN; i++)
    {
        msg.send_buf[i]= ((i+1)<<24) | ((i+1)<<16) | ((i+1)<<8) | ((i+1)<<0);
    }
    i2s_configure(FIFO_LEVEL_32, rate, bit_length, i2s_mode);
    i2s_transfer(msg.send_buf, msg.recv_buf, I2S_DATA_LEN, MASTER);

    for(i=0; i<I2S_DATA_LEN; i++)
    {
        rt_kprintf("msg.send_buf[%d]=0x%x ---msg.recv_buf[%d]=0x%x \r\n", i,
            msg.send_buf[i], i, msg.recv_buf[i]);
    }
    rt_kprintf("---i2s_master_test_over---\r\n");

}
else if(strcmp(argv[1], "slave") == 0)                                     //slave
{
    rt_kprintf("---i2s_slave_test_start---\r\n");
```

```
if(msg.send_buf == NULL)

{
    rt_kprintf("---msg.send_buf error --\r\n ");
    return 0;
}

for(i=0; i<I2S_DATA_LEN; i++)
{
    msg.send_buf[i]= ((i+1)<<24) | ((i+1)<<16) | ((i+1)<<8) | ((i+1)<<0) |0x80808080;
}

i2s_configure(FIFO_LEVEL_32, rate, bit_length, i2s_mode);
i2s_transfer(msg.send_buf, msg.recv_buf, I2S_DATA_LEN, SLAVE);

for(i=0; i<I2S_DATA_LEN; i++)
{
    rt_kprintf("msg.send_buf[%d]=0x%x , msg.recv_buf[%d]=0x%x \r\n", i, msg.send_buf[i],i,
    msg.recv_buf[i]);
}

rt_kprintf("---i2s_slave_test_over---\r\n");
}
else
{
    rt_kprintf("---no test command--\r\n");
}

i2s_trans.p_rx_buf = RT_NULL;
i2s_trans.p_tx_buf = RT_NULL;

if(msg.send_buf != RT_NULL)
{
    os_free(msg.send_buf);
    msg.send_buf= RT_NULL;
}

if(msg.recv_buf != RT_NULL)
```

```
{  
    os_free(msg.recv_buf);  
    msg.recv_buf= RT_NULL;  
}  
  
rt_kprintf("---i2s_test_over---\r\n");  
return 0;  
}  
  
MSH_CMD_EXPORT(i2s_test, i2s_test);
```


7 通用SPI

7.1 通用SPI简介

BK7251有硬件spi模块，特性如下：

- a) 数据交换长度可配，常以byte为单位，MSB先发，LSB后发；
- b) 支持主机模式，时钟可配置，最大速率30MHZ；
- c) 支持从机模式，能承受的最大速率10MHZ；
- d) 时钟极性（CPOL）和时钟相位（CPHA）可配置；
- e) 支持四线全双工（MOSI、MISO、CSN、CLK）和三线半双工（DATA、CS、CLK）。

7.2 通用SPI Related API

通用SPI的驱动,已经关联到 RT-thread操作系统的标准设备操作函数集了,所以直接调用RT-thread标准设备操作接口进行操作,相关接口参考\rt-thread\components\drivers\include\drivers\spi.h。

函数	描述
rt_spi_configure()	配置spi设备
rt_spi_send()	通过spi接口发送数据（从模式可能会挂起）
rt_spi_recv()	接口spi接口接收数据（从模式可能会挂起）

7.2.1 spi通用结构体说明

rt_spi_device:

parent	spi device对象
bus	spi bus句柄
config	spi 模式配置的结构体

7.2.2 spi模块配置

在使用SPI接口前，需配置SPI接口：

```
rt_err_t rt_spi_configure(struct rt_spi_device *device, struct rt_spi_configuration *cfg);
```

参数	描述
device	SPI设备接口的指针
cfg	SPI配置结构体，见如下说明
返回	RT_EOK(0): 成功; 其他: 出错

参数类型

rt_spi_configuration:

mode	spi工作模式
data_width	发送/接收 数据位宽
reserved	保留
max_hz	spi速率配置，仅master有效

7.2.3 spi发送数据

```
rt_inline rt_size_t rt_spi_send(struct rt_spi_device *device,
                                const void *send_buf,
                                rt_size_t length);
```

参数	描述
device	SPI设备接口的指针
send_buf	要发送的数据指针
length	要发送的数据长度
返回	此次已发送的字节数

主模式下，发送完所有数据后，立即返回。从模式下，可能会挂起，直到与之通信的SPI主发起spi时序，并且所有数据都发。

7.2.4 spi接收数据

```
rt_inline rt_size_t rt_spi_recv(struct rt_spi_device *device,
                                 void *recv_buf,
                                 rt_size_t length);
```

参数	描述
device	SPI设备接口的指针
recv_buf	存放接收的数据指针
length	要接收的数据长度
返回	此次已接收的字节数

主模式下，读SPI总线上的数据后，立即返回。从模式下，可能会挂起，直到与之通信的SPI主发起spi时序，收到非0长度的数据就会返回。

7.3 通用SPI示例代码

示例代码参考\test\general_spi_test.c。打开宏定义: GENERAL_SPI_TEST,

开启通用spi功能测试。

7.3.1 关键说明

• 通用SPI宏定义

#define	RT_USING_SPI	开启SPI模式
#define	CFG_USE_SPI_MASTER	开启master
#define	CFG_USE_SPI_SLAVE	开启slave

• SPI工作模式:

#define	RT_SPI_CPHA	(1<<0)	sck第二个边沿采样数据
#define	RT_SPI_CPOL	(1<<1)	sck空闲时处于高电平
#define	RT_SPI_LSB	(0<<2)	0-LSB
#define	RT_SPI_MSB	(1<<2)	1-MSB
#define	RT_SPI_MASTER	(0<<3)	master模式
#define	RT_SPI_SLAVE	(1<<3)	slave模式
#define	RT_SPI_MODE_0	(0 0)	CPOL = 0, CPHA = 0
#define	RT_SPI_MODE_1	(0 RT_SPI_CPHA)	CPOL = 0, CPHA = 1
#define	RT_SPI_MODE_2	(RT_SPI_CPOL 0)	CPOL = 1, CPHA = 0
#define	RT_SPI_MODE_3	(RT_SPI_CPOL RT_SPI_CPHA)	CPOL = 1, CPHA = 1
#define	RT_SPI_MODE_MASK	(RT_SPI_CPHA RT_SPI_CPOL RT_SPI_MSB RT_SPI_SLAVE)	所有bit位为1

7.3.2 示例代码

/*程序清单： 这是通用spi的使用例程，使用前确保函数 rt_hw_spi_device_init在系统初始化时调用。

* 命令调用格式： gspi_test master/slave tx/rx rate len

* 程序功能： 配置spi接口为主/从，传输速率rate，完成发送/接收

*/

```
#include <rtthread.h>
```

```
#include <rthw.h>
```

```
#include <rtdevice.h>
```

```
#include <stdio.h>
```

```
#include <string.h>
```

```
#include "sys_config.h"
```

```
#define SPI_BAUDRATE (10 * 1000 * 1000)
```

```
#define SPI_TX_BUF_LEN (32)
```

```
#define SPI_RX_BUF_LEN (32)
```

```
/*依赖 CFG_USE_SPI_MASTER 和 CFG_USE_SPI_SLAVE两个宏，位于sys_config.h中 */
#if ((CFG_USE_SPI_MASTER) &&(CFG_USE_SPI_SLAVE))
int gspi_test(int argc, char** argv)
{
    struct rt_spi_device *spi_device;
    struct rt_spi_configuration cfg;
    /*找到设备*/
    spi_device = (struct rt_spi_device *)rt_device_find("gspi");
    if (spi_device == RT_NULL) {
        rt_kprintf("spi device %s not found!\n", "gspi");
        return -RT_ENOSYS;
    }
    cfg.data_width = 8;
    if(strcmp(argv[1], "master") == 0)
    {
        /*设置成 主模式、MSB、CPOL = 0, CPHA = 0*/
        cfg.mode = RT_SPI_MODE_0 | RT_SPI_MSB | RT_SPI_MASTER;
    }
    else if(strcmp(argv[1], "slave") == 0)
    {
        /*设置成 从模式、MSB、CPOL = 0, CPHA = 0*/
        cfg.mode = RT_SPI_MODE_0 | RT_SPI_MSB | RT_SPI_SLAVE;
    }
    else
    {
        rt_kprintf("gspi_test master/slave tx/rx rate len\n");
        return -RT_ENOSYS;
    }
    /* SPI Interface with Clock Speeds Up to 30 MHz */
    if(argc == 5)
        cfg.max_hz = atoi(argv[3]);
    else
        cfg.max_hz = SPI_BAUDRATE;
    rt_kprintf("cfg:%d, 0x%02x, %d\n", cfg.data_width, cfg.mode, cfg.max_hz);
    /*配置设备*/
    rt_spi_configure(spi_device, &cfg);
    if(strcmp(argv[2], "tx") == 0)
    {
        rt_uint8_t *buf;
        int tx_len;
```

```
if(argc < 4)
    tx_len = SPI_TX_BUF_LEN;
else
    tx_len = atoi(argv[4]);
rt_kprintf("spi init tx_len:%d\n", tx_len);
buf = rt_malloc(tx_len * sizeof(rt_uint8_t));
if(buf)
{
    rt_memset(buf, 0, tx_len);
    for(int i=0; i<tx_len; i++)
    {
        buf[i] = i & 0xff;
    }
    /*发送数据， 从模式可能会挂起*/
    rt_spi_send(spi_device, buf, tx_len);
    for(int i=0; i<tx_len; i++)
    {
        rt_kprintf("%02x,", buf[i]);
        if((i+1)%32 == 0)
            rt_kprintf("\r\n");
    }
    rt_kprintf("\r\n");
    rt_free(buf);
}
}
else if(strcmp(argv[2], "rx") == 0)
{
    rt_uint8_t *buf;
    int rx_len;
    if(argc < 4)
        rx_len = SPI_RX_BUF_LEN;
    else
        rx_len = atoi(argv[4]);
    rt_kprintf("spi init rx_len:%d\n", rx_len);
    buf = rt_malloc(rx_len * sizeof(rt_uint8_t));
    if(buf) {
        rt_memset(buf, 0, rx_len);
        /*接收数据， 从模式可能会挂起*/
        rx_len = rt_spi_rcv(spi_device, buf, rx_len);
        rt_kprintf("rx ret:%d\r\n", rx_len);
    }
}
```

```

        for(int i=0; i<rx_len; i++)
        {
            rt_kprintf("%02x,", buf[i]);
            if((i+1)%32 == 0)
                rt_kprintf("\r\n");
        }
        rt_kprintf("\r\n");
        rt_free(buf);
    }
}
else
{
    rt_kprintf("gsapi_test master/slave   tx/rx   rate   len\r\n");
}
}
MSH_CMD_EXPORT(gsapi_test, gsapi_test);
#endif

```

7.4 操作说明

7.4.1 打开配置

sys_config.h, general_spi_test.c文件中，设置如下：

#define	RT_USING_SPI	开启SPI模式
#define	CFG_USE_SPI_MASTER	开启master
#define	CFG_USE_SPI_SLAVE	开启slave

7.4.2 运行现象

烧录完成后，接到在串口工具中，发送cmd命令。

gsapi_test master tx 1000000 128, 1M速率发送128字节数据

```

gsapi_test master tx 1000000 128
cfg:8, 0x04, 1000000
[SPI]:data_width = 8
[SPI]:max_hz = 1000000, mode:0x04
max_hz = 1000000
config spi clk source 26MHz
div = 12
spi_clk = 1000000
source_clk = 26000000
target frequency = 1000000, actual frequency = 1000000
[CTRL]:0x00c30c3f
spi init tx_len:128
00, 01, 02, 03, 04, 05, 06, 07, 08, 09, 0a, 0b, 0c, 0d, 0e, 0f, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 1a, 1b, 1c, 1d, 1e, 1f,
20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 2a, 2b, 2c, 2d, 2e, 2f, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 3a, 3b, 3c, 3d, 3e, 3f,
40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 4a, 4b, 4c, 4d, 4e, 4f, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 5a, 5b, 5c, 5d, 5e, 5f,
60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 6a, 6b, 6c, 6d, 6e, 6f, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 7a, 7b, 7c, 7d, 7e, 7f,

```

图7.4.1

接着发送gspi_test master rx 1000000 128, 1M速率接收128字节数据

```
msh />
msh />gspi_test master rx 1000000 128
cfg:8, 0x04, 1000000
[SPI]:data_width = 8
[SPI]:max_hz = 1000000, mode:0x04
max_hz = 1000000
config spi clk source 26MHz
div = 12
spi_clk = 1000000
source_clk = 26000000
target frequency = 1000000, actual frequency = 1000000
[CTRL]:0x00c30c3f
spi init rx_len:128
rx ret:128
00,01,02,03,04,05,06,07,08,09,0a,0b,0c,0d,0e,0f,10,11,12,13,14,15,16,17,18,19,1a,1b,1c,1d,1e,1f,
20,21,22,23,24,25,26,27,28,29,2a,2b,2c,2d,2e,2f,30,31,32,33,34,35,36,37,38,39,3a,3b,3c,3d,3e,3f,
40,41,42,43,44,45,46,47,48,49,4a,4b,4c,4d,4e,4f,50,51,52,53,54,55,56,57,58,59,5a,5b,5c,5d,5e,5f,
60,61,62,63,64,65,66,67,68,69,6a,6b,6c,6d,6e,6f,70,71,72,73,74,75,76,77,78,79,7a,7b,7c,7d,7e,7f,
```

图7.4.2

7.5 注意事项

- SPI的最高速率为30M，超过这个之后，会自动被限制在30M。
- Rate参数，单位不是M，比如1M，需要写1 000 000。

8 通用SPI FLASH设备

8.1 通用SPI FLASH简介

SPI FLASH设备是一个外挂的标准flash，特点如下：

- a) 使用SPI四线主模式；
- b) 最高访问速度达30MHz。

8.2 通用SPI FLASH Related API

通用SPI FLASH的驱动，已经关联到 RT-thread操作系统的标准设备操作函数集了，所以直接调用RT-thread标准设备操作接口进行操作。相关接口如下：

函数	描述
<code>rt_device_control()</code>	其他spi flash的操作，如：擦除指定位置，去/加写保护等

8.2.1 控制设备

需要对设备进行其他操作：

```
rt_err_t rt_device_control(rt_device_t dev, int cmd, void *arg);
```

参数	描述
<code>dev</code>	SPI FLASH设备的指针
<code>cmd</code>	设备定义的操作命令，具体见下面说明。
<code>*arg</code>	设备定义的操作命令参数，具体见下面说明。
返回	出错信息：RT_EOK(0)：成功；其他：出错

8.3 通用SPI FLASH示例代码

示例代码参考\test\general_spi_flash_test.c。打开宏定义：SPI_FLASH_TEST，开启通用spi flash功能测试。

8.3.1 关键说明

• FLASH工作命令：

<code>#define</code>	<code>BK_SPI_FLASH_ERASE_CMD</code>	擦除命令
<code>#define</code>	<code>BK_SPI_FLASH_PROTECT_CMD</code>	flash加写保护
<code>#define</code>	<code>BK_SPI_FLASH_UNPROTECT_CMD</code>	flash去写保护

8.3.2 示例代码

```
/*
```


* 程序清单： 这是spi flash的使用例程，使用前确保函数 `rt_spi_flash_hw_init(void)`函数在系统初始化自动调用。

* 命令调用格式： `gspi_flash_test`

* 程序功能： 测试 `spi flash` 读写数据的功能

*/

```
#include <rtthread.h>
```

```
#include <rthw.h>
```

```
#include <rtdevice.h>
```

```
#include <stdio.h>
```

```
#include <string.h>
```

```
#include "sys_config.h"
```

```
#ifdef BEKEN_USING_SPI_FLASH
```

```
/*SPI FLASH 需要关联BEKEN_USING_SPI_FLASH、 CFG_USE_SPI_MASTER 、  
CFG_USE_SPI_MST_FLASH 三个功能宏*/
```

```
#if ((CFG_USE_SPI_MASTER == 0) || (CFG_USE_SPI_MST_FLASH == 0))
```

```
#error "test gspi psram need 'CFG_USE_SPI_MASTER' and 'CFG_USE_SPI_MST_FLASH'"
```

```
#endif
```

```
#include "drv_spi_flash.h"
```

```
#define FTEST_BUF_SIZE      1024
```

```
#define FTEST_BASE          0x40
```

```
#define FTEST_ADDR          0x100000
```

```
void gspi_flash_test(int argc, char** argv)
```

```
{
```

```
    struct rt_device *flash;
```

```
    /*找到设备*/
```

```
    flash = rt_device_find("spi_flash");
```

```
    if (flash == NULL)
```

```
    {
```

```
        rt_kprintf("psram not found \n");
```

```
        return;
```

```
    }
```

```
    /*初始化设备 */
```

```
    if (rt_device_init(flash) != RT_EOK)
```

```
    {
```

```
        return;
```

```
    }
```

```
    /*打开设备*/
```

```
    if (rt_device_open(flash, 0) != RT_EOK)
```

```
{
    return;
}

uint8_t buffer[FTEST_BUF_SIZE], *ptr;
int i;
rt_kprintf("[SPIFLASH]: SPIFLASH test begin\n");
rt_memset(buffer, 0, FTEST_BUF_SIZE);
/*先读一次 */
rt_device_read(flash, FTEST_ADDR, buffer, FTEST_BUF_SIZE);
/*打印读到的数据 */
ptr = buffer;
rt_kprintf("flash data:%x\r\n", FTEST_ADDR);
for(i=0; i<FTEST_BUF_SIZE; i++)
{
    rt_kprintf("0x%02x,", ptr[i]);
    if((i+1)%16 == 0)
        rt_kprintf("\r\n");
}
rt_kprintf("\r\n");

/*初始化将写数据，数据来源于代码的 FTEST_BASE开始的地方 */
ptr = (uint8_t *)FTEST_BASE;
rt_kprintf("base data:%08x\r\n", ptr);
for(i=0; i<FTEST_BUF_SIZE; i++)
{
    rt_kprintf("0x%02x,", ptr[i]);
    buffer[i] = ptr[i];
    if((i+1)%16 == 0)
        rt_kprintf("\r\n");
}
rt_kprintf("\r\n");
/*写之前，先去写保护 */
rt_device_control(flash, BK_SPI_FLASH_UNPROTECT_CMD, NULL);
/*写数据 */
rt_device_write(flash, FTEST_ADDR, buffer, FTEST_BUF_SIZE);
rt_kprintf("write fin\r\n");
/*清0buffer */
rt_memset(buffer, 0, FTEST_BUF_SIZE);
/*再读回来 */
rt_device_read(flash, FTEST_ADDR, buffer, FTEST_BUF_SIZE);
```

```
rt_kprintf("read fin\r\n");
/*打印读回来的数据 */
ptr = buffer;
rt_kprintf("flash data:%x\r\n", FTEST_ADDR);
for(i=0; i<FTEST_BUF_SIZE; i++)
{
    rt_kprintf("0x%02x,", ptr[i]);
    if((i+1)%16 == 0)
        rt_kprintf("\r\n");
}
rt_kprintf("\r\n");
/*擦除flash */
rt_kprintf("earase\r\n");
BK_SPIFLASH_ERASE_ST erase_st;
erase_st.addr = FTEST_ADDR;
erase_st.size = 4 * 1024;
rt_device_control(flash, BK_SPI_FLASH_ERASE_CMD, &erase_st);
rt_kprintf("[SPIFLASH]: SPIFLASH test end\r\n");
/*加写保护 */
rt_device_control(flash, BK_SPI_FLASH_PROTECT_CMD, NULL);
/*关闭设备 */
rt_device_close(flash);
}
MSH_CMD_EXPORT(gspi_flash_test, gspi_flash_test);
#endif // BEKEN_USING_SPI_FLASH
```

8.4 操作说明

初始FLASH为空，然后写入数据，读出数据，读到的数据与写入数据相同，最后擦除数据。

8.4.1 运行现象

在串口输入gspi_flash_test即可启动该项功能，设备log如下：

```
gspi_flash_test
[SPIFLASH]: SPIFLASH test begin
flash data:100000
0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,
0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,

base data:00000040
0xef,0xbe,0xad,0xde,0x01,0x00,0x00,0x00,0x10,0x40,0x2d,0xe9,0x20,0x20,0x52,0xe2,
0x05,0x00,0x00,0x3a,0x18,0x50,0xb1,0xe8,0x20,0x20,0x52,0xe2,0x18,0x50,0xa0,0xe8,
0x18,0x50,0xb1,0xe8,0x18,0x50,0xa0,0xe8,0xf9,0xff,0xff,0x2a,0x02,0xce,0xb0,0xe1,
0x18,0x50,0xb1,0x28,0x18,0x50,0xa0,0x28,0x18,0x00,0xb1,0x48,0x18,0x00,0xa0,0x48,
0x10,0x40,0xbd,0xe8,0x02,0xcf,0xb0,0xe1,0x04,0x30,0x91,0x24,0x04,0x30,0x80,0x24,

write fin
read fin
flash data:100000
0xef,0xbe,0xad,0xde,0x01,0x00,0x00,0x00,0x10,0x40,0x2d,0xe9,0x20,0x20,0x52,0xe2,
0x05,0x00,0x00,0x3a,0x18,0x50,0xb1,0xe8,0x20,0x20,0x52,0xe2,0x18,0x50,0xa0,0xe8,
0x18,0x50,0xb1,0xe8,0x18,0x50,0xa0,0xe8,0xf9,0xff,0xff,0x2a,0x02,0xce,0xb0,0xe1,
0x18,0x50,0xb1,0x28,0x18,0x50,0xa0,0x28,0x18,0x00,0xb1,0x48,0x18,0x00,0xa0,0x48,
0x10,0x40,0xbd,0xe8,0x02,0xcf,0xb0,0xe1,0x04,0x30,0x91,0x24,0x04,0x30,0x80,0x24,
0x1e,0xff,0x2f,0x01,0x82,0x2f,0xb0,0xe1,0xb2,0x30,0xd1,0x20,0x01,0x20,0xd1,0x44.

earase
[SPIFLASH]: SPIFLASH test end
msh />
msh />
```

图8.4.1-1

8.5 注意事项

- 需要外接FLASH模块进行。
- 擦除后重新读取内容，如果全是0xFF，则说明已经擦除成功。

9 通用SPI PSRAM设备

9.1 通用SPI PSRAM简介

SPI PSRAM设备，即需要外挂psram，是基于通用SPI模块的一个具体应用：

- a) 使用SPI四线主模式；
- b) 最高访问速度达30MHZ。

9.2 通用SPI PSRAM related API

通用SPI PSRAM的驱动，已经关联到 RT-thread操作系统的标准设备操作函数集了，所以直接调用RT-thread标准设备操作接口进行操作。

9.3 通用SPI PSRAM示例代码

9.3.1 关键说明

• SPI PSRAM宏定义:

#define	BEKEN_USING_SPI_PSRAM	开启spi psram模块
----------------	------------------------------	---------------

9.3.2 示例代码

```
/*
 * 程序清单： 这是spi psram的使用例程，测试的时候需要外挂一个psram，使用前确保函数
             rt_spi_psram_hw_init ()会在系统初始化自动调用；
 * 命令调用格式： spi_psram_test
 * 程序功能： 测试 spi psram 读写数据的功能
 */
#include <rtthread.h>
#include <rthw.h>
#include <rtdevice.h>
#include <stdio.h>
#include <string.h>
#include "sys_config.h"

#ifdef BEKEN_USING_SPI_PSRAM
/*SPI PSRAM 需要关联BEKEN_USING_SPI_PSRAM、 CFG_USE_SPI_MASTER 、
CFG_USE_SPI_MST_PSRAM 三个功能宏*/
#if ((CFG_USE_SPI_MASTER == 0) || (CFG_USE_SPI_MST_PSRAM == 0))
#error "test gsapi psram need 'CFG_USE_SPI_MASTER' and 'CFG_USE_SPI_MST_PSRAM'"
#endif
#endif
```

```
void spi_psram_test(int argc, char** argv)
{
    struct rt_device *psram;
    /*找到设备*/
    psram = rt_device_find("spi_psram");
    if (psram == NULL)
    {
        rt_kprintf("psram not found \n");
        return;
    }
    /*初始化设备*/
    if (rt_device_init(psram) != RT_EOK)
    {
        return;
    }
    /*打开设备*/
    if (rt_device_open(psram, 0) != RT_EOK)
    {
        return;
    }
    uint8_t buffer[4096];
    int i;
    rt_kprintf("[PSRAM]: SPRAM test begin\n");
    /*初始化将要写入的设备*/
    for(i = 0; i < sizeof(buffer); i++)
    {
        buffer[i] = (uint8_t)i;
    }
    /*写设备*/
    rt_device_write(psram, 0, buffer, sizeof(buffer));
    /*清0 buffer*/
    rt_memset(buffer, 0, sizeof(buffer));
    /*读设备*/
    rt_device_read(psram, 0, buffer, sizeof(buffer));
    /*比较读到的数据与写入的数据是否一致，不一致的打印出来 */
    for(i = 0; i < sizeof(buffer); i++)
    {
        if(buffer[i] != (uint8_t)i)
        {

```

```
        rt_kprintf("[%02d]: %02x - %02x\n", i, (uint8_t)i, buffer[i]);
    }
}

rt_kprintf("[PSRAM]: SPRAM test end\n");
/*关闭设备*/
rt_device_close(psram);
}
MSH_CMD_EXPORT(spi_psram_test, spi_psram_test);
#endif // BEKEN_USING_SPI_PSRAM
```

9.4 操作说明

9.4.1 打开配置

示例代码参考\test\general_spi_psram_test.c。打开宏定义：BEKEN_USING_SPI_PSRAM，开启通用spi psram功能测试。

9.4.2 运行现象

在串口输入spi_psram_test启动此项功能，设备log如下图：

```
spi_psram_test
max_hz = 100000000
config spi clk source DCO
div = 8
spi_clk = 100000000
source_clk = 1800000000
target frequency = 100000000, actual frequency = 100000000
[CTRL]:0x00c3083f
MF ID:c8
KGD:40, (0x5D-pass, 0x55-fail)
[PSRAM]: SPRAM test begin
[00]: 00 - ff
[01]: 01 - ff
[02]: 02 - ff
[03]: 03 - ff
[04]: 04 - ff
[05]: 05 - ff
[06]: 06 - ff
[07]: 07 - ff
[08]: 08 - ff
[09]: 09 - ff
[10]: 0a - ff
[11]: 0b - ff
[12]: 0c - ff
[13]: 0d - ff
[14]: 0e - ff
[15]: 0f - ff
[16]: 10 - ff
[17]: 11 - ff
[18]: 12 - ff
[19]: 13 - ff
[20]: 14 - ff
[4090]: fa - ff
[4091]: fb - ff
[4092]: fc - ff
[4093]: fd - ff
[4094]: fe - ff
[PSRAM]: SPRAM test end
msh />
msh />
```

图9.4.2-1

9.5 注意事项

- 需要外接SRAM模块进行。

10 高速SPI从设备

10.1 高速SPI从设备简介

Highspeed spi slave设备(以下简称spi_hs)是为了解决通用spi从模式不能承受大spi clock的问题而诞生的:

- a) 支持四线全双工、三线半双工模块;
- b) 支持MSB、LSB可配置;
- c) 支持DMA传输;
- d) 承受spi clock 达50MHZ。

Note:驱动为了方便和简单, 固定了spi_hs的配置如下: 四线模式、MSB、使用DMA发送和接收。

10.2 高速SPI从设备Related API

spi_hs的驱动, 已经关联到 RT-thread操作系统的标准设备操作函数集了, 所以直接调用RT-thread标准设备操作接口进行操作。

10.3 高速SPI从示例代码

10.3.1 关键说明

• 高速SPI从设备宏定义:

#define	BEKEN_USING_SPI_HSLAVE	开启高速spi 从设备
#define	SPI_TX_BUF_LEN	高速spi 从设备发送数据长度
#define	SPI_RX_BUF_LEN	高速spi 从设备接收数据长度

10.3.2 示例代码

```
/*
 * 程序清单: 这是spi hs的使用例程, 使用前确保函数 rt_spi_hslave_hw_init () 在系统初始化自动调用。
 * 命令调用格式: spi_hs_test tx/rx len
 * 程序功能: 测试 spi hs 读写数据的功能
 */
#include <rtthread.h>
#include <rthw.h>
#include <rtdevice.h>
#include <stdio.h>
#include <string.h>
```



```
#include "sys_config.h"

#define SPI_TX_BUF_LEN    (512)
#define SPI_RX_BUF_LEN    (512)
#ifdef BEKEN_USING_SPI_HSLAVE

/*SPI HS需要关联BEKEN_USING_SPI_HSLAVE、 CFG_USE_HSLAVE_SPI 二个功能宏*/
#if (CFG_USE_HSLAVE_SPI == 0)
#error "spi_hs_test need 'CFG_USE_HSLAVE_SPI' and 'CFG_USE_SPI_MST_PSRAM'"
#endif

int spi_hs_test(int argc, char** argv)
{
    struct rt_device *spi_hs;
    /*找到设备*/
    spi_hs = (struct rt_device *)rt_device_find("spi_hs");
    if (spi_hs == RT_NULL)
    {
        rt_kprintf("spi device %s not found!\n", "spi_hs");
        return -RT_ENOSYS;
    }
    /*打开设备*/
    if (rt_device_open(spi_hs, 0) != RT_EOK)
    {
        return 0;
    }
    if(strcmp(argv[1], "tx") == 0)
    {
        rt_uint8_t *buf;
        int tx_len;
        if(argc < 3)
            tx_len = SPI_TX_BUF_LEN;
        else
            tx_len = atoi(argv[2]);
        rt_kprintf("spi hs tx_len:%d\n", tx_len);
        buf = rt_malloc(tx_len * sizeof(rt_uint8_t));
        if(buf)
        {
            rt_memset(buf, 0, tx_len);
            for(int i=0; i<tx_len; i++)
            {
                buf[i] = i & 0xff;
            }
        }
    }
}
```

```
    }
    /*写数据*/
    rt_device_write(spi_hs, 0, (const void *)buf, tx_len);
    for(int i=0; i<tx_len; i++)
    {
        rt_kprintf("%02x,", buf[i]);
        if((i+1)%32 == 0)
            rt_kprintf("\r\n");
    }
    rt_kprintf("\r\n");
    rt_free(buf);
}
}
else if(strcmp(argv[1], "rx") == 0)
{
    rt_uint8_t *buf;
    int rx_len;
    if(argc < 3)
        rx_len = SPI_RX_BUF_LEN;
    else
        rx_len = atoi(argv[2]);
    rt_kprintf("spi hs rx_len:%d\n", rx_len);
    buf = rt_malloc(rx_len * sizeof(rt_uint8_t));
    if(buf)
    {
        rt_memset(buf, 0, rx_len);
        /*接收数据*/
        rx_len = rt_device_read(spi_hs, 0, buf, rx_len);
        rt_kprintf("rx ret:%d\r\n", rx_len);
        for(int i=0; i<rx_len; i++)
        {
            rt_kprintf("%02x,", buf[i]);
            if((i+1)%32 == 0)
                rt_kprintf("\r\n");
        }
        rt_kprintf("\r\n");
        rt_free(buf);
    }
}
else
```

```
{
    rt_kprintf("spi_hs_test tx/rx len\r\n");
}
/*关闭设备*/
rt_device_close(spi_hs);
}
MSH_CMD_EXPORT(spi_hs_test, spi_hs_test);
#endif // BEKEN_USING_SPI_HSLAVE
```

10.4 操作说明

10.4.1 打开配置

示例代码参考/test/ highspeed_spi_slave_test.c。打开宏定义：
SPI_HSLAVE_TEST，开启高速spi功能的测试。

10.4.2 运行现象

编译运行后，在调试串口输入spi_hs_test tx 100，接收100字节数据

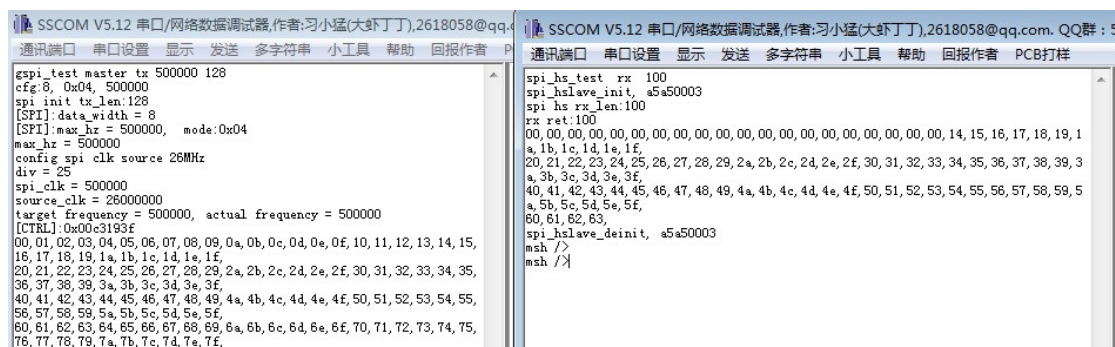


图10.4.2-1

10.5 注意事项

- 速率最高为50M。

11 GPIO

11.1 GPIO简介

BK7251芯片上的引脚一般分为4类：电源、时钟、模拟/控制与I/O，I/O口在使用模式上又分为General Purpose Input Output（通用输入/输出），简称GPIO，与功能复用I/O（如SPI/I2C/UART等）。

11.2 GPIO Related API

GPIO相关接口参考\rt-thread\components\drivers\include\drivers\pin.h
应用程序可通过以下API访问GPIO，相关接口如下所示：

函数	描述
rt_pin_mode()	设置引脚模式
rt_pin_write()	设置引脚电平
rt_pin_read()	读取引脚电平
rt_pin_attach_irq()	绑定引脚中断回调函数
rt_pin_irq_enable()	使能引脚中断
rt_pin_detach_irq()	脱离引脚中断回调函数

11.2.1 设置引脚模式

引脚在使用前需要先设置好输入或者输出模式，通过如下函数完成：

```
void rt_pin_mode(rt_base_t pin, rt_base_t mode);
```

参数	描述
pin	引脚编号
mode	引脚工作模式
返回	空

11.2.2 设置引脚电平

设置引脚输出电平的函数如下所示：

```
void rt_pin_write(rt_base_t pin, rt_base_t value);
```

参数	描述
pin	引脚编号
value	电平逻辑值，可取2种宏定义值之一： PIN_LOW 低电平， PIN_HIGH 高电平

返回

空

11.2.3 读取引脚电平

读取引脚电平的函数如下所示：

```
int rt_pin_read(rt_base_t pin);
```

参数	描述
pin	引脚编号
返回	PIN_LOW 低电平 PIN_HIGH 高电平

11.2.4 绑定引脚中断回调函数

若要使用到引脚的中断功能，可以使用如下函数将某个引脚配置为某种中断触发模式并绑定一个中断回调函数到对应引脚，当引脚中断发生时，就会执行回调函数：

```
rt_err_t rt_pin_attach_irq(rt_int32_t pin, rt_uint32_t mode, void (*hdr)(void *args), void *args);
```

参数	描述
pin	引脚编号
mode	中断触发模式
hdr	中断回调函数，用户需要自行定义这个函数
args	中断回调函数的参数，不需要时设置为RT_NULL
返回	RT_EOK：绑定成功； 错误码 ：绑定失败

11.2.5 使能引脚中断

绑定好引脚中断回调函数后使用下面的函数使能引脚中断：

```
rt_err_t rt_pin_irq_enable(rt_base_t pin, rt_uint32_t enabled);
```

参数	描述
pin	引脚编号
enabled	状态，可取2 种值之一：PIN_IRQ_ENABLE（开启），PIN_IRQ_DISABLE（关闭）
返回	RT_EOK：使能成功； 错误码 ：使能失败

11.2.6 脱离引脚中断回调函数

可以使用如下函数脱离引脚中断回调函数：

```
rt_err_t rt_pin_detach_irq(rt_int32_t pin);
```

参数	描述
pin	引脚编号
返回	RT_EOK: 脱离成功; 错误码 : 脱离失败

引脚脱离了中断回调函数以后，中断并没有关闭，还可以调用绑定中断回调函数再次绑定其他回调函数。

11.3 GPIO示例代码

示例代码的主要步骤如下：

1. 初始化LED控制引脚，输出高电平，点亮LED2和LED3。
2. 初始化S3和S4按键，设置下降沿方式触发中断并使能。

11.3.1 关键说明

• GPIO宏定义

目前支持的引脚工作模式可取如所示的4种宏定义值之一，每种模式对应的芯片实际支持的模式需参考PIN设备驱动程序的具体实现：

#define	PIN_MODE_OUTPUT	0x00	输出
#define	PIN_MODE_INPUT	0x01	输入
#define	PIN_MODE_INPUT_PULLUP	0x02	上拉输入
#define	PIN_MODE_INPUT_PULLDOWN	0x03	下拉输入
#define	PIN_IRQ_MODE_RISING	0x00	上升沿触发
#define	PIN_IRQ_MODE_FALLING	0x01	下降沿触发
#define	PIN_IRQ_MODE_RISING_FALLING	0x02	边沿触发（上升沿和下降沿）
#define	PIN_IRQ_MODE_HIGH_LEVEL	0x03	高电平触发
#define	PIN_IRQ_MODE_LOW_LEVEL	0x04	低电平触发

11.3.2 示例代码

```
/*
 * 程序清单： 这是一个PIN 设备使用例程
 * 例程导出了pin_led_sample 命令到控制终端
 * 命令调用格式： pin_led_sample
 * 程序功能： 通过按键控制led 对应引脚的电平状态控制led
 */
#include <rtthread.h>
#include <rtdevice.h>
#include "test_config.h"
```

```
#ifdef GPIO_DEMO
#define LED_PIN_NUM 24
#define LED1_PIN_NUM 26
#define KEY0_PIN_NUM 2
#define KEY1_PIN_NUM 3
void led_on(void *args) {
    rt_kprintf("turn on led!\n");
    rt_pin_write(LED_PIN_NUM, PIN_HIGH);
}
void led_off(void *args) {
    rt_kprintf("turn off led!\n");
    rt_pin_write(LED_PIN_NUM, PIN_LOW);
}
static void pin_led_sample(void) {
    /* led 引脚为输出模式*/
    rt_pin_mode(LED_PIN_NUM, PIN_MODE_OUTPUT);
    /* 默认低电平*/
    rt_pin_write(LED_PIN_NUM, PIN_HIGH);
    /* 按键0引脚为输入模式*/
    rt_pin_mode(KEY0_PIN_NUM, PIN_MODE_INPUT_PULLUP);
    /* 绑定中断，下降沿模式，回调函数名为beep_on */
    rt_pin_attach_irq(KEY0_PIN_NUM, PIN_IRQ_MODE_FALLING, led_on, RT_NULL);
    /* 使能中断*/
    rt_pin_irq_enable(KEY0_PIN_NUM, PIN_IRQ_ENABLE);
    /* 按键1引脚为输入模式*/
    rt_pin_mode(KEY1_PIN_NUM, PIN_MODE_INPUT_PULLUP);
    rt_pin_attach_irq(KEY1_PIN_NUM, PIN_IRQ_MODE_FALLING, led_off, RT_NULL);
    rt_pin_irq_enable(KEY1_PIN_NUM, PIN_IRQ_ENABLE);
    rt_pin_mode(LED1_PIN_NUM, PIN_MODE_OUTPUT);
    /* 默认低电平*/
    rt_pin_write(LED1_PIN_NUM, PIN_HIGH);
}
/* 导出到msh 命令列表中*/
MSH_CMD_EXPORT(pin_led_sample, pin led sample);
#endif
```

11.4 操作说明

GPIO设备示例代码位于\test\gpio_demo.c，SDK默认没有打开此功能，需

要打开功能后测试，在调试串口输入触发命令使能GPIO Demo，然后操作S3和S4按键控制LED灯，按下S3按键，LED2熄灭，按下S4按键，LED2重新点亮。

11.4.1 打开配置

打开宏定义：GPIO_DEMO，编译下载后，调试串口输入pin_led_sample，LED2和LED3点亮。



图11.4.1-1

11.4.2 运行现象

按下S3按键，LED2熄灭：

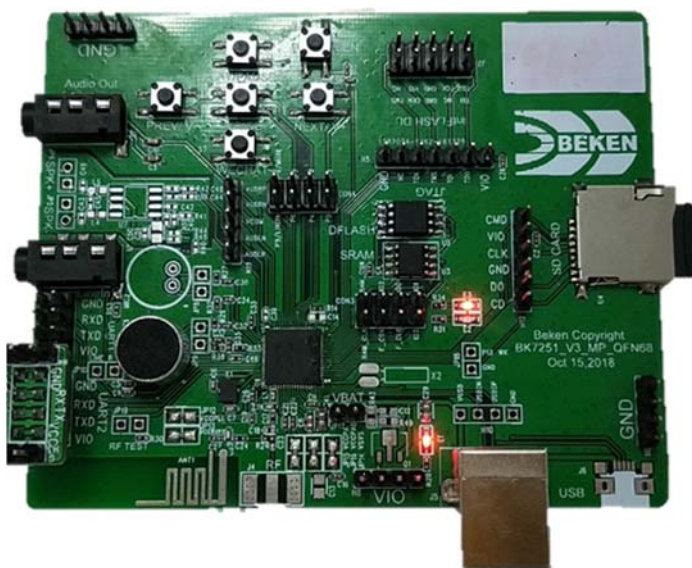


图11.4.1-2

12 UART

12.1 UART简介

UART (Universal Asynchronous Receiver/Transmitter) 通用异步收发传输器, UART 作为异步串口通信协议的一种, 工作原理是将传输数据的每个字符一位接一位地传输。是在应用程序开发过程中使用频率最高的数据总线。

UART 串口的特点是将数据一位一位地顺序传送, 只要2根传输线就可以实现双向通信, 一根线发送数据的同时用另一根线接收数据。UART串口通信有几个重要的参数, 分别是波特率、起始位、数据位、停止位和奇偶检验位, 对于两个使用UART 串口通信的端口, 这些参数必须匹配, 否则通信将无法完成。

12.2 UART Related API

应用程序通过BK7251 SDK提供的I/O 设备管理接口来访问串口硬件, 其API已经关联到 RT-thread操作系统的标准设备操作函数集了, 所以直接调用RT-thread标准设备操作接口进行操作相关接口如下所示:

函数	描述
<code>rt_device_control()</code>	控制设备

12.2.1 uart通用结构体说明

serial:rt_device类型的结构体

配置uart参数的结构体:

serial_configure:

结构体类型	成员
<code>rt_uint32_t baud_rate</code>	波特率设置: 一般为115200
<code>rt_uint32_t data_bits</code>	数据位: 一般为8bit
<code>rt_uint32_t stop_bits</code>	停止位: 一般为1
<code>rt_uint32_t parity</code>	奇偶校验位: 无校验位
<code>rt_uint32_t bit_order</code>	大小端: 一般为小端
<code>rt_uint32_t invert</code>	模式转化: 不转换
<code>rt_uint32_t bufsz</code>	接收buffer大小
<code>rt_uint32_t reserved</code>	保留

12.2.2 控制串口设备

通过命令控制字, 应用程序可以对串口设备进行配置, 通过如下函数完成:

```
rt_err_t rt_device_control(rt_device_t dev, rt_uint8_t cmd, void* arg);
```

参数	描述
dev	设备句柄
cmd	命令控制字，参考宏定义
arg	控制的参数，可取类型：struct serial_configure
返回	RT_EOK 函数执行成功 -RT_ENOSYS 执行失败，dev 为空 其他错误码 执行失败

接收缓冲区：

当串口使用中断接收模式打开时，串口驱动框架会根据RT_SERIAL_RB_BUFSZ 大小开辟一块缓冲区用于保存接收到的数据，底层驱动接收到一个数据，都会在中断服务程序里面将数据放入缓冲区。

12.3 UART示例代码

示例代码的主要步骤如下：

1. 首先查找串口设置获取设备句柄。
2. 初始化回调函数发送使用的信号量，然后以读写及中断接收方式打开串口设备。
3. 设置串口设备的接收回调函数，之后发送字符串，并创建读取数据线程。读取数据线程会尝试读取一个字符数据，如果没有数据则会挂起并等待信号量，当串口设备接收到数据时会触发中断并调用接收回调函数，此函数会发送信号量唤醒线程，此时线程会马上读取接收到的数据。

12.3.1 关键说明

• UART宏定义

BK7251SDK 提供的默认宏配置如下：

#define	BAUD_RATE_115200	115200	波特率
#define	DATA_BITS_8	8	数据位
#define	STOP_BITS_1	1	停止位
#define	PARITY_NONE	0	奇偶校验位
#define	BIT_ORDER_LSB	0	高位在前或者低位在前
#define	NRZ_NORMAL	0	模式
#define	RT_SERIAL_RB_BUFSZ	64	接收数据缓冲区大小

设备配置宏定义：

#define	RT_DEVICE_CTRL_CONFIG	0x03	配置对应的设备
---------	-----------------------	------	---------

设置波特率：

#define	BAUD_RATE_2400	2400
#define	BAUD_RATE_4800	4800
#define	BAUD_RATE_9600	9600
#define	BAUD_RATE_19200	19200
#define	BAUD_RATE_38400	38400
#define	BAUD_RATE_57600	57600
#define	BAUD_RATE_115200	115200
#define	BAUD_RATE_203400	203400
#define	BAUD_RATE_460800	460800
#define	BAUD_RATE_921600	921600
#define	BAUD_RATE_2000000	2000000
#define	BAUD_RATE_3000000	3000000

设置数据位:

#define	DATA_BITS_5	5
#define	DATA_BITS_6	6
#define	DATA_BITS_7	7
#define	DATA_BITS_8	8
#define	DATA_BITS_9	9

设置停止位:

#define	STOP_BITS_1	0
#define	STOP_BITS_2	1
#define	STOP_BITS_3	2
#define	STOP_BITS_4	3

设置奇偶校验位:

#define	PARITY_NONE	0
#define	PARITY_ODD	1
#define	PARITY_EVEN	2

设置高位在前:

#define	BIT_ORDER_LSB	0 高位在前
#define	BIT_ORDER_MSB	1 高位在后

模式选择

#define	NRZ_NORMAL	0 normal mode
#define	NRZ_INVERTED	1 inverted mode

12.3.2 示例代码

```
#include <rtthread.h>
#include "test_config.h"
#include <rtdevice.h>
```

```
#ifndef UART_DEMO
#define SAMPLE_UART_NAME "uart1"
/* 用于接收消息的信号量*/
static struct rt_semaphore rx_sem;
static rt_device_t serial;

/* 接收数据回调函数*/
static rt_err_t uart_input(rt_device_t dev, rt_size_t size)
{
    /* 串口接收到数据后产生中断， 调用此回调函数， 然后发送接收信号量*/
    rt_sem_release(&rx_sem);
    return RT_EOK;
}

static void serial_thread_entry(void *parameter)
{
    char ch;
    while (1)
    {
        /* 从串口读取一个字节的的数据， 没有读取到则等待接收信号量*/
        while (rt_device_read(serial, -1, &ch, 1) != 1)
        {
            /* 阻塞等待接收信号量， 等到信号量后再次读取数据*/
            rt_sem_take(&rx_sem, RT_WAITING_FOREVER);
        }
        /* 读取到的数据通过串口错位输出*/
        ch = ch + 1;
        rt_device_write(serial, 0, &ch, 1);
    }
}

static int uart_sample(int argc, char *argv[])
{
    rt_err_t ret = RT_EOK;
    char uart_name[RT_NAME_MAX];
    char str[] = "hello BK72xx!\r\n";
    struct serial_configure config = RT_SERIAL_CONFIG_DEFAULT;
    if (argc == 2) {
        rt_strncpy(uart_name, argv[1], RT_NAME_MAX);
    }
}
```

```
else {
    rt_strncpy(uart_name, SAMPLE_UART_NAME, RT_NAME_MAX);
}
serial = rt_device_find(uart_name);
if (!serial) {
    rt_kprintf("find %s failed!\n", uart_name);
    return RT_ERROR;
}
rt_sem_init(&rx_sem, "rx_sem", 0, RT_IPC_FLAG_FIFO);
/* 以中断接收及轮询发送模式打开串口设备*/
rt_device_open(serial, RT_DEVICE_FLAG_INT_RX);
/* 设置接收回调函数*/
rt_device_set_rx_indicate(serial, uart_input);

/* 设置配置参数 */
config.baud_rate = BAUD_RATE_115200;
config.data_bits = DATA_BITS_8;
config.stop_bits = STOP_BITS_1;
config.parity = PARITY_NONE;
config.bufsz = 2048;    /*can not change buffer size, must be 2048*/
/* 打开设备后才可修改串口配置参数 */
rt_device_control(serial, RT_DEVICE_CTRL_CONFIG, &config);
rt_device_write(serial, 0, str, (sizeof(str) - 1));
rt_thread_t thread = rt_thread_create("serial", serial_thread_entry, RT_NULL, 1024, 25, 10);
if (thread != RT_NULL) {
    rt_thread_startup(thread);
}
else {
    ret = RT_ERROR;
}
return ret;
}
/* 导出到msh 命令列表中*/
MSH_CMD_EXPORT(uart_sample, uart device sample);
#endif
```

12.4 操作说明

uart示例代码位于\test\uart_demo.c，修改配置信息后，可测试uart1的通信功能。

12.4.1 打开配置

打开宏定义：UART_DEMO，重新编译完成后，将固件下载至设备。

12.4.2 运行现象

• 硬件连接

串口转USB模块一端连接串口UART1，另一端插入PC。

• 运行

调试串口输入uart_sample,UART1会发送hello BK72xx!, PC上串口助手收到数据后，发送0x40给设备UATRT1,然后设备回复0x41,运行情况如下图所示：

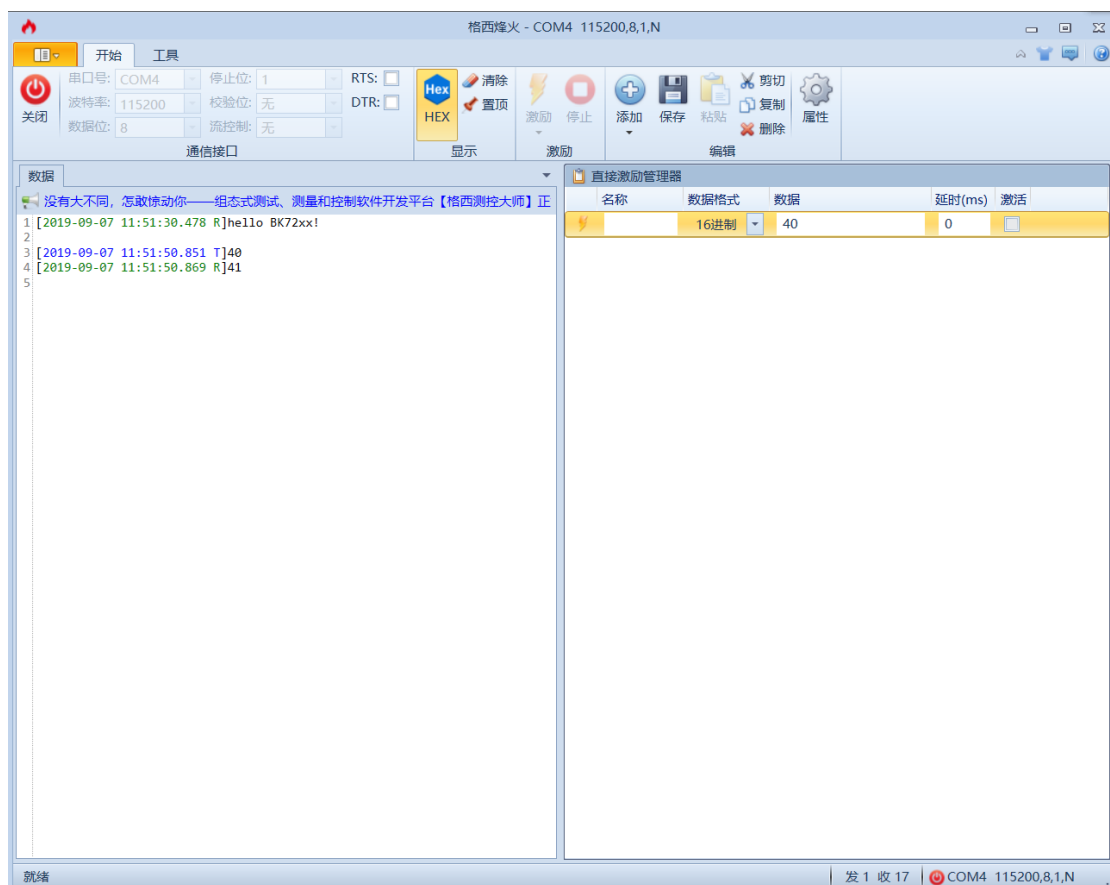


图12.4.2-1

12.5 注意事项

接收数据缓冲区大小默认64字节。若一次性数据接收字节数很多，没有及时读取数据，那么缓冲区的数据将会被新接收到的数据覆盖，造成数据丢失，建议调大缓冲区。

13 Player

13.1 Player简介

Player提供播放、暂停和恢复等功能。

13.2 Player Related API

Player相关接口参考components\player\player\basic\player.h。应用程序可通过以下APIs访问List Player，相关接口如下所示：

函数	描述
player_init()	初始化播放器
player_play()	开始/恢复播放
player_stop()	停止播放
player_pause()	暂停播放
player_do_seek()	设置当前歌曲播放的位置
player_set_uri ()	设置播放歌曲的uri
player_get_uri ()	获取当前歌曲的uri
player_get_state()	获取当前播放器的状态
player_play_data()	准备开始播放用户音频数据流
player_write_data()	写入用户音频数据到播放器
player_set_content_length	设置用户音频数据的总长度
player_set_event_callback()	设置播放层事件的回调函数
player_set_volume()	设置播放器音量
player_get_volume()	获取播放器音量
player_get_duration()	获取当前歌曲的持续时间
player_get_position()	获取当前歌曲播放的位置

13.2.1 初始化播放器

播放器使用前需要进行初始化：

```
int list_player_init(void)
```

参数	描述
void	空
返回	0：初始化成功，-1：初始化失败

13.2.2 开始/恢复播放

```
int player_play (void)
```



参数	描述
void	空
返回	-1: 失败, 0: 成功

13.2.3 停止播放

int player_stop (void)

参数	描述
void	空
返回	-1: 失败, 0: 成功

13.2.4 暂停播放

int player_pause (void)

参数	描述
void	空
返回	-1: 失败, 0: 成功

13.2.5 设置当前歌曲播放的位置

int player_do_seek (int sec)

参数	描述
sec	开始播放位置, 单位为秒
返回	-1: 失败, 0: 成功

13.2.6 设置播放歌曲的uri

int player_set_uri (const char *uri)

参数	描述
*uri	歌曲的uri地址
返回	-1: 失败, 0: 成功

13.2.7 获取当前歌曲的uri

const char* player_get_uri(void);

参数	描述
----	----

void	空
返回	返回当前歌曲的uri

13.2.8 获取当前播放器的状态

```
int player_get_state (void)
```

参数	描述
void	空
返回	当前播放器状态

13.2.9 准备开始播放用户音频数据流

```
int player_play_data(int codec_type, int len)
```

参数	描述
codec_type	用户音频数据流的解码器类型
len	用户音频数据流的长度，长度未知设为-1
返回	-1: 失败, 0: 成功

13.2.10 写入用户音频数据到播放器

```
int player_write_data(char *data, int len)
```

参数	描述
*data	用户音频数据流
len	要写入的用户音频数据流长度
返回	-1: 失败, 0: 成功

13.2.11 设置用户音频数据的总长度

```
int player_set_content_length(int len)
```

参数	描述
len	音频数据流的总长度
返回	-1: 失败, 0: 成功

13.2.12 设置播放层事件的回调函数

```
int player_set_event_callback(void (*callback)(int event, void *user_data), void *user_data)
```

参数	描述
callback	回调函数
*user_data	用户私有数据
返回	-1: 失败, 0: 成功

13.2.13 设置播放器音量

```
int player_set_volume(int volume)
```

参数	描述
volume	播放器音量 (0~99)
返回	-1: 失败, 0: 成功

13.2.14 设置播放器音量

```
int player_get_volume(void)
```

参数	描述
Void	空
返回	播放器音量 (0~99)

13.2.15 获取当前歌曲的持续时间

```
int player_get_duration(void)
```

参数	描述
void	空
返回	获取正在播放的歌曲的持续时间, 单位为秒

13.2.16 获取当前歌曲播放的位置

```
int player_get_position(void)
```

参数	描述
void	空
返回	当前歌曲播放的位置, 单位为毫秒

13.3 Player示例代码

```
/ #include "rtthread.h"
```

```
#include "optparse.h"

#include <stdlib.h>
#include "player_system.h"
#include "player.h"
#include "audio_codec.h"
#include "audio_stream.h"

#if defined(PPLAYER_ENABLE_NET_STREAM)
#include "netstream.h"
#include "stream_pipe.h"
#include "netstream_buffer.h"
#endif

#if defined(PPLAYER_ENABLE_NET_STREAM)
extern struct stream_pipe * netstream_get_pipe(void);
extern rt_uint32_t rb_buffer_data_len(struct rb_buffer *rb);
#endif

static struct optparse_long opts[] =
{
    {"version", 'V', OPTPARSE_NONE}, /* 版本 */
    {"help", 'h', OPTPARSE_NONE}, /* 帮助 */
    {"start", 's', OPTPARSE_REQUIRED}, /* 播放 */
    {"stop", 't', OPTPARSE_NONE}, /* 停止 */
    {"pause", 'p', OPTPARSE_NONE}, /* 暂停 */
    {"resume", 'r', OPTPARSE_NONE}, /* 恢复 */
    {"seek", 'k', OPTPARSE_REQUIRED}, /* 移动 */
    {"volume", 'v', OPTPARSE_REQUIRED}, /* 音量 */
    {"dump", 'd', OPTPARSE_NONE}, /* 信息 */
    {NULL, 0, OPTPARSE_NONE}
};

static void usage(void)
{
    rt_kprintf("usage: player [option] [target] ...\n\n");
    rt_kprintf("usage options:\n");
    rt_kprintf("  -V,      --version          Print player version message.\n");
    rt_kprintf("  -h,      --help            Print defined help message.\n");
    rt_kprintf("  -s URI,  --start=URI       Play music with URI(network links or local
```

```
files).\n");

    rt_kprintf(" -t,      --stop                Stop playing music.\n");
    rt_kprintf(" -p,      --pause                Pause the music.\n");
    rt_kprintf(" -r,      --resume                Resume the music.\n");
    rt_kprintf(" -k sec, --seek=sec                Seek the specified seconds to play.\n");
    rt_kprintf(" -v lvl, --volume=lvl            Change the volume(0~99).\n");
    rt_kprintf(" -d,      --dump                Dump play relevant information.\n");
}

#if defined(PPLAYER_ENABLE_NET_STREAM)
int stream_buffer(int argc, char **argv)
{
    struct stream_pipe *pipe = netstream_get_pipe();

    rt_kprintf("read_mirror : %d read_index : %d\n", pipe->ringbuffer.read_mirror,
pipe->ringbuffer.read_index);
    rt_kprintf("write_mirror : %d write_index : %d\n", pipe->ringbuffer.write_mirror,
pipe->ringbuffer.write_index);
    rt_kprintf("buffer_size : %d\n", pipe->ringbuffer.buffer_size);

    return 0;
}

int stream_pipe_dump(void)
{
    rt_uint32_t total_size, used_size, remain_size;
    struct stream_pipe *pipe = netstream_get_pipe();

    total_size = pipe->ringbuffer.buffer_size;
    used_size = rb_buffer_data_len(&pipe->ringbuffer);
    remain_size = total_size - used_size;

    rt_kprintf("\nPlayer NetCache:\n");
    rt_kprintf("total size    - %d \n", total_size);
    rt_kprintf("used size     - %d \n", used_size);
    // rt_kprintf("remain size   - %d \n", remain_size);
    // rt_kprintf("read_mirror   - %d \n", pipe->ringbuffer.read_mirror);
    // rt_kprintf("read_index    - %d \n", pipe->ringbuffer.read_index);
    // rt_kprintf("write_mirror  - %d \n", pipe->ringbuffer.write_mirror);
    // rt_kprintf("write_index   - %d \n", pipe->ringbuffer.write_index);
```

```
rt_kprintf("ready_wm      - %d \n", pipe->reader_ready_wm);
rt_kprintf("resume_wm     - %d \n", pipe->writer_resume_wm);

return 0;
}
#endif

static void dump_status(void)
{
    const char *state[] =
    {
        "STOPPED",
        "PLAYING",
        "PAUSED"
    };

    rt_kprintf("\nPlayer Dump Status:\n");
    rt_kprintf("status   - %s\n", state[player_get_state()]);
    rt_kprintf("URI      - %s\n", (player_get_uri() != NULL) ? player_get_uri() : "NULL");
    rt_kprintf("volume   - %d\n", player_get_volume());
    rt_kprintf("codec    - %s\n", audio_codec_tostring(audio_codec_get()));

    if (player_get_state() != PLAYER_STAT_STOPPED)
    {
        int value;

        value = player_get_duration();
        rt_kprintf("duration - %02d:%02d\n", value/60, value%60);

        value = player_get_position() / 1000;
        rt_kprintf("position - %02d:%02d\n", value/60, value%60);
    }

#ifdef PLAYER_ENABLE_NET_STREAM
    stream_pipe_dump();
#endif
}

int player(int argc, char **argv)
{

```

```
int ch;
int option_index;
struct optparse options;

rt_bool_t help    = RT_FALSE;
rt_bool_t start   = RT_FALSE;
rt_bool_t stop    = RT_FALSE;
rt_bool_t pause   = RT_FALSE;
rt_bool_t resume  = RT_FALSE;
rt_bool_t seek    = RT_FALSE;
rt_int32_t second = (-1);
rt_int8_t volume  = (-1);
rt_bool_t dump    = RT_FALSE;
rt_bool_t version = RT_FALSE;

rt_uint8_t action_cnt = 0;

char *uri = RT_NULL;

if(argc == 1)
{
    usage();
    return RT_EOK;
}

/* Parse cmd */
optparse_init(&options, argv);
while((ch = optparse_long(&options, opts, &option_index)) != -1)
{
    switch(ch)
    {
        case 'h': /* 帮助 */
            help = RT_TRUE;
            break;

        case 's': /* 播放 */
            start = RT_TRUE;
            uri = (options.optarg == RT_NULL) ? (RT_NULL) : rt_strdup(options.optarg);
            action_cnt++;
            break;
```

```
case 't': /* 停止 */
    stop = RT_TRUE;
    action_cnt++;
    break;

case 'p': /* 暂停 */
    pause = RT_TRUE;
    action_cnt++;
    break;

case 'r': /* 恢复 */
    resume = RT_TRUE;
    action_cnt++;
    break;

case 'k': /* 移动 */
    seek = RT_TRUE;
    second = (options.optarg == RT_NULL) ? (-1) : atoi(options.optarg);
    break;

case 'v': /* 音量 */
    volume = (options.optarg == RT_NULL) ? (-1) : atoi(options.optarg);
    break;

case 'd': /* 信息 */
    dump = RT_TRUE;
    break;

case 'V': /* 版本 */
    version = RT_TRUE;
    break;
}

// 判断 播放 暂停 停止 恢复 移动 命令是否多次使用 不能共存使用
if(action_cnt > 1)
{
    rt_kprintf("START STOP PAUSE RESUME SEEK parameter can't be used at the same
time.\n");
```

```
        return RT_EINVAL;
    }

    if(help == RT_TRUE)
    {
        usage();
        return RT_EOK;
    }

    // 播放器动作
    if((start == RT_TRUE) && (uri != RT_NULL) && (seek != RT_TRUE))
    {
        rt_kprintf("//////////////////// player_play \n");
        player_stop();
        player_set_uri(uri);
        player_play();
        rt_kprintf("//////////////////// player_play end \n");

        if(uri)
        {
            rt_free(uri);
        }
    }

    if((start == RT_TRUE) && (uri != RT_NULL) && (seek == RT_TRUE))
    {
        rt_kprintf("//////////////////// player_play_position \n");
        player_stop();
        player_set_uri(uri);
        player_play_position(second);
        rt_kprintf("//////////////////// player_play_position end \n");

        if(uri)
        {
            rt_free(uri);
        }
    }

    else if(stop == RT_TRUE)
    {
        rt_kprintf("//////////////////// player_stop \n");
        player_stop();
    }
}
```



```
    rt_kprintf("//////////////////// player_stop end \n");
    rt_kprintf("stop play.\n");
}
else if(pause == RT_TRUE)
{
    rt_kprintf("//////////////////// player_pause \n");
    player_pause();
    rt_kprintf("//////////////////// player_pause end \n");
    rt_kprintf("pause play.\n");
}
else if(resume == RT_TRUE)
{
    rt_kprintf("//////////////////// player_play(resume) \n");
    player_play();
    rt_kprintf("//////////////////// player_play(resume) end \n");
    rt_kprintf("resume play.\n");
}
else if((seek == RT_TRUE) && (second != (-1)))
{
    rt_kprintf("//////////////////// player_do_seek \n");
    player_do_seek(second);
    rt_kprintf("//////////////////// player_do_seek end \n");
    rt_kprintf("seek %dsec.\n", second);
}

if(volume != (-1))
{
    if((volume < 0) || (volume > 99))
    {
        rt_kprintf("set volume failed. volume needs to be set to 0~99.\n", volume);
    }
    else
    {
        player_set_volume(volume);
        rt_kprintf("set play volume %d%%.\n", volume);
    }
}

if(dump == RT_TRUE)
{
```

```
        dump_status();
    }

    if(version == RT_TRUE)
    {
        player_get_version();
    }

    return RT_EOK;
}

MSH_CMD_EXPORT(player, player func test cmd.);
```

13.4 操作说明

List Player示例代码参考\components\player\example\cmd\cmd_player.c, 开启Player功能重新编译, 下载运行后, 需要首先连接网络然后在进行功能测试。

13.4.1 运行现象

- 设备连接路由器

调试串口输入wifi_demo sta your_ssid your_key,设备开始连接路由器。

```
wifi_demo sta your_ssid your_key
sta_Command
ssid: your_ssid key: your_key
rl_sta_start
[sa_sta]MM_RESET_REQ
[sa_sta]ME_CONFIG_REQ
[sa_sta]ME_CHAN_CONFIG_REQ
[sa_sta]MM_START_REQ
hapd_intf_add_vif,type:2, s:0, id:0
[wlan_connect]:start tick = 0, connect done tick = 22379, total = 22379
[wlan_connect]:start tick = 0, connect done tick = 22385, total = 22385
[WLAN_MGNT]wlan sta connected evenew dtim period:2
nt callback
IP UP: 192.168.44.27
[ip_up]:start tick = 0, ip_up tick = 25797, total = 25797
```

- 开始播放

调试串口输入player -s http://183.193.243.90:9151/mp3/73865964.mp3,用耳机接在Audio Out接口上即可听到声音, 设备log信息如下所示:

```
[14:40:02.253]发->◇player -s http://183.193.243.90:9151/mp3/73865964.mp3
[14:40:02.259]收<-◆player -s http://183.193.243.90:9151/mp3/73865964.mp3
//////////////////////////////// player_play

[14:40:02.506]收<-◆[32m[I/web_work] web session:
http://183.193.243.90:9151/mp3/73865964.mp3.[0m
[32m[I/web_work] Mime type: audio/mpeg.[0m
[32m[I/web_work] position: 0.[0m
[32m[I/web_work] Content length: 4568685Bytes.[0m
[32m[I/audio_codec] current codec type MP3[0m

[14:40:02.943]收<-◆[icodec]:open sound device
audio_device_opened

===set fade in flag===

//////////////////////////////// player_play end
msh />
```

图13.4.1-1

14 网络接口

14.1 网络接口简介

BK7251的SDK给上层应用提供的网络接口用于：1.启动STATION模式，去连接指定的网络。2.关闭STATION模式。3.启动AP模式，供其他设备连接。4.关闭AP模式。5.启动监听模式，供上层配网。6.关闭监听模式。7.获取状态，如连接状态，加密方式，当前使用的信道等等。8.设置状态，如设置信道，IP地址等等。9.启动scan，并获取scan结果。

14.2 网络接口 Related API

网络接口相关接口参考**beken378\func\include\wlan_ui_pub.h**，应用程序可通过以下APIs控制网络，相关接口如下所示：

函数	描述
bk_wlan_start()	启动网络，包括STATION和AP
bk_wlan_start_sta_adv()	启动STATION快速连接
bk_wlan_stop()	关闭网络，包括STATION和AP
bk_wlan_start_scan()	启动scan
bk_wlan_scan_ap_reg_cb()	注册scan结束后的回调函数
bk_wlan_start_assign_scan()	scan特定的网络
bk_wlan_start_monitor()	启动监听模式
bk_wlan_stop_monitor()	关闭监听模式
bk_wlan_register_monitor_cb()	注册监听回调函数
bk_wlan_get_ip_status()	获取当前的网络状态
bk_wlan_get_link_status()	获取当前的连接状态
bk_wlan_get_channel()	获取当前的信道
bk_wlan_set_channel()	设置信道

14.2.1 启动网络

上层应该获得ssid与password之后，可以启动网络。通过如下函数完成：

```
OSStatus bk_wlan_start(network_InitTypeDef_st *inNetworkInitPara);
```

参数	描述
inNetworkInitPara	传入需要配置信息
返回	kNoErr: 成功；其他：失败

参数类型

network_InitTypeDef_st:

char	wifi_mode	WiFi模式
char	wifi_ssid[33]	需要连接或建立的网络SSID
char	wifi_key[64]	需要连接或建立的网络密码
char	local_ip_addr[16]	静态IP地址，在DHCP关闭时有效
char	net_mask[16]	静态子网掩码，在DHCP关闭时有效
char	gateway_ip_addr[16]	静态网关地址，在DHCP关闭时有效
char	dns_server_ip_addr[16]	静态DNS地址，在DHCP关闭时有效
char	dhcp_mode	DHCP模式
char	reserved[32]	保留
char	wifi_bssid[6];	Mac地址
Int	wifi_retry_interval	重连间隔，单位是毫秒

14.2.2 启动STATION快速连接

```
OSStatus bk_wlan_start_sta_adv(network_InitTypeDef_adv_st *inNetworkInitParaAdv);
```

参数	描述
inNetworkInitParaAdv	需要传入的网络参数
返回	kNoErr: 成功; 其他: 失败

参数类型

network_InitTypeDef_adv_st:

apinfo_adv_t	ap_info	需要快速连接的网络信息
char	key[64]	需要快速连接的网络密码
Int	key_len	网络密码长度
char	local_ip_addr[16]	静态IP地址，在DHCP关闭时有效
char	net_mask[16]	静态子网掩码，在DHCP关闭时有效
char	gateway_ip_addr[16]	静态网关地址，在DHCP关闭时有效
char	dns_server_ip_addr[16]	静态DNS地址，在DHCP关闭时有效
char	dhcp_mode	DHCP模式
char	reserved[32]	保留
int	wifi_retry_interval	重连时间，单位是毫秒

apinfo_adv_t:

char	ssid[32]	需要快速连接的网络信息
char	bssid[6]	需要快速连接的网络密码
uint8_t	channel	网络密码长度
wlan_sec_type_t	security	当前网络的加密方式
typedef uint8_t wlan_sec_type_t		

14.2.3 关闭网络

```
int bk_wlan_stop(char mode);
```

参数	描述
mode	需要关闭的模式，见枚举类型中关于mode的说明
返回	kNoErr: 成功；其他：失败

14.2.4 启动scan

```
void bk_wlan_start_scan(void);
```

参数	描述
void	无
返回	无

14.2.5 注册scan结束后的回调函数

```
void bk_wlan_scan_ap_reg_cb(FUNC_2PARAM_PTR ind_cb);
```

参数	描述
ind_cb	scan结束后回调的函数。函数定义： typedef void (*FUNC_2PARAM_PTR)(void *arg, uint8_t vif_idx);
返回	无

14.2.6 scan特定的网络

```
void bk_wlan_start_assign_scan(UINT8 **ssid_ary, UINT8 ssid_num);
```

参数	描述
ssid_ary	指定网络的SSID
ssid_num	指定网络的数量
返回	无

14.2.7 启动监听模式

```
int bk_wlan_start_monitor(void);
```

参数	描述
void	无

返回	kNoErr: 成功; 其他: 失败
----	--------------------

14.2.8 关闭监听模式

```
int bk_wlan_stop_monitor(void);
```

参数	描述
void	无
返回	kNoErr: 成功; 其他: 失败

14.2.9 注册监听回调函数

```
void bk_wlan_register_monitor_cb(monitor_data_cb_t fn);
```

参数	描述
fn	注册的回调函数。函数定义： typedef void (*monitor_data_cb_t)(uint8_t *data, int len, hal_wifi_link_info_t *info);
返回	无

14.2.10 获取当前的网络状态

```
OSStatus bk_wlan_get_ip_status(IPStatusTypedef *outNetpara, WiFi_Interface inInterface);
```

参数	描述
outNetpara	保存获取的网络状态。
inInterface	需要获取网络状态的模式。
返回	kNoErr: 成功; 其他: 失败

参数类型

IPStatusTypedef:

uint8_t	dhcp	获取的DHCP模式
char	ip[16]	获取的IP地址
char	gate[16]	获取的网关IP地址
char	mask[16]	获取的子网掩码
char	dns[16]	DNS服务IP地址
char	mac[16]	获取的mac地址
char	broadcastip[16]	获取的广播IP地址

```
#define WiFi_Interface wlanInterfaceTypeDef
```

```
typedef enum
{
    SOFT_AP,                /*AP模式*/
    STATION                  /*STATION模式*/
} wlanInterfaceTypeDef;
```

14.2.11 获取当前的连接状态

```
OSStatus bk_wlan_get_link_status(LinkStatusTypeDef *outStatus);
```

参数	描述
outStatus	保存获取的连接状态。具体参考该结构体的说明。
返回	kNoErr: 成功; 其他: 失败

参数类型

LinkStatusTypeDef:

msg_sta_states	conn_state	当前连接状态
int	wifi_strength	当前的信号强度
uint8_t	ssid[32]	当前网络的SSID
uint8_t	bssid[6]	当前网络的BSSID
int	channel	当前网络的信道
wlan_sec_type_t	security	当前网络的加密方式
		Typedef uint8_t wlan_sec_type_t

```
typedef enum {
    MSG_IDLE = 0,          /*无任何连接状态*/
    MSG_CONNECTING,        /*正在连接中*/
    MSG_PASSWD_WRONG,      /*密码错误*/
    MSG_NO_AP_FOUND,       /*未找到要连接的网络*/
    MSG_CONN_FAIL,         /*连接失败*/
    MSG_CONN_SUCCESS,      /*连接成功*/
    MSG_GOT_IP,            /*获得IP*/
}msg_sta_states;
```

14.2.12 获取当前的信道

```
int bk_wlan_get_channel(void);
```


参数	描述
void	无
返回	channel

14.2.13 设置信道

```
int bk_wlan_set_channel(int channel);
```

参数	描述
channel	传入的信道数值
返回	0: 成功; 其他: 失败

14.3 网络接口使用示例

14.3.1 关键说明

• DHCP宏定义说明

#define DHCP_DISABLE	(0)	/*DHCP关闭*/
#define DHCP_CLIENT	(1)	/*DHCP客户端模式*/
#define DHCP_SERVER	(2)	/* DHCP服务端模式*/

14.3.2 代码示例

启动一个STATION连接:

```
void demo_sta_app_init(char *oob_ssid,char *connect_key)
{
    /*定义一个结构体，用于传入参数*/
    network_InitTypeDef_st wNetConfig;
    int len;
    /*把这个结构体置空*/
    os_memset(&wNetConfig, 0x0, sizeof(network_InitTypeDef_st));

    /*检查SSID的长度，不能超过32字节*/
    len = os_strlen(oob_ssid);
    if(SSID_MAX_LEN < len)
    {
        bk_printf("ssid name more than 32 Bytes\r\n");
        return;
    }
}
```

```
/*将SSID跟密码传入结构体*/
os_strcpy((char *)wNetConfig.wifi_ssid, oob_ssid);
os_strcpy((char *)wNetConfig.wifi_key, connect_key);

/*当前为客户端模式*/
wNetConfig.wifi_mode = STATION;
/*采用DHCP CLIENT的方式获得，从路由器动态获取IP地址*/
wNetConfig.dhcp_mode = DHCP_CLIENT;
wNetConfig.wifi_retry_interval = 100;

bk_printf("ssid:%s key:%s\r\n", wNetConfig.wifi_ssid, wNetConfig.wifi_key);
/*启动WiFi连接*/
bk_wlan_start(&wNetConfig);
}
```

启动AP模式，提供其他客户端连接：

```
void demo_softap_app_init(char *ap_ssid, char *ap_key)
{
    /*定义一个结构体，用于传入参数*/
    network_InitTypeDef_adv_st wNetConfigAdv;
    int len;
    /*将结构体置空*/
    os_memset( &wNetConfigAdv, 0x0, sizeof(network_InitTypeDef_adv_st) );
    len = os_strlen(ap_ssid);
    if(SSID_MAX_LEN < len)
    {
        bk_printf("ssid name more than 32 Bytes\r\n");
        return;
    }
    /*传入要连接的ap ssid 和 ap key*/
    os_strcpy((char *)wNetConfig.wifi_ssid, ap_ssid);
    os_strcpy((char *)wNetConfig.wifi_key, ap_key);

    /*当前为ap模式*/
    wNetConfig.wifi_mode = SOFT_AP;
    /*采用DHCP SERVER模式，需要将静态地址分配为本地地址*/
    wNetConfig.dhcp_mode = DHCP_SERVER;
    wNetConfig.wifi_retry_interval = 100;
    os_strcpy((char *)wNetConfig.local_ip_addr, WLAN_DEFAULT_IP);
}
```

```
os_strcpy((char *)wNetConfig.net_mask, WLAN_DEFAULT_MASK);
os_strcpy((char *)wNetConfig.dns_server_ip_addr, WLAN_DEFAULT_IP);

bk_printf("ssid:%s key:%s\r\n", wNetConfig.wifi_ssid, wNetConfig.wifi_key);
/*启动ap*/
bk_wlan_start(&wNetConfig);}
```

启动STATION的快速连接:

```
void demo_sta_adv_app_init(char *oob_ssid,char *connect_key)
{
    /*定义一个结构体，用于传入参数*/
    network_InitTypeDef_adv_st wNetConfigAdv;
    /*将结构体置空*/
    os_memset( &wNetConfigAdv, 0x0, sizeof(network_InitTypeDef_adv_st) );
    /*传入要连接的SSID*/
    os_strcpy((char*)wNetConfigAdv.ap_info.ssid, oob_ssid);
    /*传入要连接的网络的bssid，下面这个bssid仅供参考*/
    hwaddr_aton("12:34:56:00:00:01", wNetConfigAdv.ap_info.bssid);
    /*要连接网络的加密方式。具体参数参考该结构体说明。*/
    wNetConfigAdv.ap_info.security = SECURITY_TYPE_WPA2_MIXED;
    /*要连接的网络的信道*/
    wNetConfigAdv.ap_info.channel = 11;
    /*要连接的网络密码以及密码长度*/
    os_strcpy((char*)wNetConfigAdv.key, connect_key);
    wNetConfigAdv.key_len = os_strlen(connect_key);
    /*通过DHCP的方式获取IP地址等网络信息*/
    wNetConfigAdv.dhcp_mode = DHCP_CLIENT;
    wNetConfigAdv.wifi_retry_interval = 100;
    /*启动快速连接*/
    bk_wlan_start_sta_adv(&wNetConfigAdv);
}
```

启动scan，并分析scan的结果:

```
/*回调函数，用于scan结束后解析scan结果*/
static void scan_cb(void *ctxt, uint8_t param)
{
    /*指向scan结果的指针*/
    struct scanu_rst_upload *scan_rst;
    /*保存解析结果的结构体*/
```

```
ScanResult apList;
int i;

apList.ApList = NULL;
/*启动scan*/
scan_rst = sr_get_scan_results();
/*如果什么都没有scan到，返回；否则记录scan到的网络数量*/
if( scan_rst == NULL )
{
    apList.ApNum = 0;
    return;
}
else
{
    apList.ApNum = scan_rst->scanu_num;
}
if( apList.ApNum > 0 )
{
    /*申请对应的内存，用于保存scan的结果*/
    apList.ApList = (void *)os_malloc(sizeof(*apList.ApList) * apList.ApNum);
    for( i = 0; i < scan_rst->scanu_num; i++ )
    {
        /*将scan到的网络ssid与rssi记录下来*/
        os_memcpy(apList.ApList[i].ssid, scan_rst->res[i]->ssid, 32);
        apList.ApList[i].ApPower = scan_rst->res[i]->level;
    }
}
if( apList.ApList == NULL )
{
    apList.ApNum = 0;
}
/*打印scan的结果*/
bk_printf("Got ap count: %d\r\n", apList.ApNum);
for( i = 0; i < apList.ApNum; i++ )
{
    if(os_strlen(apList.ApList[i].ssid) >= SSID_MAX_LEN)
    {
        char temp_ssid[33];
        os_memset(temp_ssid, 0, 33);
        os_memcpy(temp_ssid, apList.ApList[i].ssid, 32);
    }
}
```

```
        bk_printf("    %s, RSSI=%d\r\n", temp_ssid, apList.ApList[i].ApPower);
    }
    else
    {
        bk_printf("    %s, RSSI=%d\r\n", apList.ApList[i].ssid, apList.ApList[i].ApPower);
    }
}
bk_printf("Get ap end.....\r\n\r\n");

/*结束后释放申请的内存*/
if( apList.ApList != NULL )
{
    os_free(apList.ApList);
    apList.ApList = NULL;
}

#if CFG_ROLE_LAUNCH
    rl_pre_sta_set_status(RL_STATUS_STA_LAUNCHED);
#endif
    sr_release_scan_results(scan_rst);
}

void demo_scan_app_init(void)
{
    /*注册scan回调函数*/
    mhdr_scanu_reg_cb(scan_cb, 0);
    /*开始scan*/
    bk_wlan_start_scan();
}
```

连接成功后，获取连接后的网络状态

```
void demo_ip_app_init(void)
{
    /*定义一个用于保存网络状态的结构体*/
    IPStatusTypedef ipStatus;
    /*将该结构体置空*/
    os_memset(&ipStatus, 0x0, sizeof(IPStatusTypedef));
    /*获取网络状态，并保存在该结构体中*/
    bk_wlan_get_ip_status(&ipStatus, STATION);
}
```

```
/*打印获取的网络状态*/  
bk_printf("dhcp=%d ip=%s gate=%s mask=%s mac=" MACSTR "\n",  
          ipStatus.dhcp, ipStatus.ip, ipStatus.gate,  
          ipStatus.mask, MAC2STR((unsigned char*)ipStatus.mac));  
}
```

连接成功后，获取连接状态：

```
void demo_state_app_init(void)  
{  
    /*定义结构体用于保存连接状态*/  
    LinkStatusTypeDef linkStatus;  
    network_InitTypeDef_ap_st ap_info;  
    char ssid[33] = {0};  
    #if CFG_IEEE80211N  
        bk_printf("sta: %d, softap: %d, b/g\n\n", sta_ip_is_start(), uap_ip_is_start());  
    #else  
        bk_printf("sta: %d, softap: %d, b/g\n\n", sta_ip_is_start(), uap_ip_is_start());  
    #endif  
  
    /*STATION模式下的连接状态*/  
    if( sta_ip_is_start() )  
    {  
        /*将用于保存状态的结构体置空*/  
        os_memset(&linkStatus, 0x0, sizeof(LinkStatusTypeDef));  
        /*获取连接状态*/  
        bk_wlan_get_link_status(&linkStatus);  
        /*打印连接状态*/  
        os_memcpy(ssid, linkStatus.ssid, 32);  
  
        bk_printf("sta:rssi=%d,ssid=%s,bssid=" MACSTR ",channel=%d,cipher_type:",  
                  linkStatus.wifi_strength, ssid, MAC2STR(linkStatus.bssid), linkStatus.channel);  
        switch(bk_sta_cipher_type())  
        {  
            case SECURITY_TYPE_NONE:  
                bk_printf("OPEN\n");  
                break;  
            case SECURITY_TYPE_WEP :  
                bk_printf("WEP\n");  
                break;  
        }  
    }  
}
```

```
        case SECURITY_TYPE_WPA_TKIP:
            bk_printf("TKIP\r\n");
            break;
        case SECURITY_TYPE_WPA2_AES:
            bk_printf("CCMP\r\n");
            break;
        case SECURITY_TYPE_WPA2_MIXED:
            bk_printf("MIXED\r\n");
            break;
        case SECURITY_TYPE_AUTO:
            bk_printf("AUTO\r\n");
            break;
        default:
            bk_printf("Error\r\n");
            break;
    }
}

/*AP模式下的连接状态*/
if( uap_ip_is_start() )
{
    /*将用于保存连接状态的结构体置空*/
    os_memset(&ap_info, 0x0, sizeof(network_InitTypeDef_ap_st));
    /*获取连接状态*/
    bk_wlan_ap_para_info_get(&ap_info);
    /*打印出获取的连接状态值*/
    os_memcpy(ssid, ap_info.wifi_ssid, 32);
    bk_printf("softap:ssid=%s,channel=%d,dhcp=%d,cipher_type:",
    ssid, ap_info.channel, ap_info.dhcp_mode);
    switch(ap_info.security)
    {
        case SECURITY_TYPE_NONE:
            bk_printf("OPEN\r\n");
            break;
        case SECURITY_TYPE_WEP :
            bk_printf("WEP\r\n");
            break;
        case SECURITY_TYPE_WPA_TKIP:
            bk_printf("TKIP\r\n");
            break;
        case SECURITY_TYPE_WPA2_AES:
```

```
        bk_printf("CCMP\r\n");
        break;
    case SECURITY_TYPE_WPA2_MIXED:
        bk_printf("MIXED\r\n");
        break;
    case SECURITY_TYPE_AUTO:
        bk_printf("AUTO\r\n");
        break;
    default:
        bk_printf("Error\r\n");
        break;
}
bk_printf("ip=%s,gate=%s,mask=%s,dns=%s\r\n",
        ap_info.local_ip_addr, ap_info.gateway_ip_addr, ap_info.net_mask,
        ap_info.dns_server_ip_addr);
}
}

/* monitor 回调函数*/
void bk_demo_monitor_cb(uint8_t *data, int len, hal_wifi_link_info_t *info)
{
    os_printf("len:%d\r\n", len);

    //Only for reference
    /*
    User can get ssid and key by prase monitor data,
    refer to the following code, which is the way airkiss
    use monitor get wifi info from data
    */

    #if 0
        int airkiss_rcv_ret;
        airkiss_rcv_ret = airkiss_rcv(ak_context, data, len);
    #endif

}

/* 程序清单： 这是一个简单网络接口程序使用例程
* 命令调用格式： wifi_demo sta oob_ssid connect_key
* 程序功能： 输入相关命令可以启动网络，连接网络等。
*/
```



```
int wifi_demo(int argc, char **argv)
{
    char *oob_ssid = NULL;
    char *connect_key;

    if (strcmp(argv[1], "sta") == 0)
    {
        os_printf("sta_Command\r\n");
        if (argc == 3)
        {
            oob_ssid = argv[2];
            connect_key = "1";
        }
        else if (argc == 4)
        {
            oob_ssid = argv[2];
            connect_key = argv[3];
        }
        else
        {
            os_printf("parameter invalid\r\n");
            return -1;
        }
        if(oob_ssid)
        {
            demo_sta_app_init(oob_ssid, connect_key);
        }
        return 0;
    }
    if(strcmp(argv[1], "adv") == 0)
    {
        os_printf("sta_adv_Command\r\n");
        if (argc == 3)
        {
            oob_ssid = argv[1];
            connect_key = "1";
        }
        else if (argc == 4)
        {

```

```
        oob_ssid = argv[1];
        connect_key = argv[2];
    }
    else
    {
        os_printf("parameter invalid\r\n");
        return -1;
    }
    if(oob_ssid)
    {
        demo_sta_adv_app_init(oob_ssid, connect_key);
    }
    return 0;
}

if(strcmp(argv[1], "softap") == 0)
{
    os_printf("SOFTAP_COMMAND\r\n\r\n");
    if (argc == 3)
    {
        oob_ssid = argv[1];
        connect_key = "1";
    }
    else if (argc == 4)
    {
        oob_ssid = argv[1];
        connect_key = argv[2];
    }
    else
    {
        os_printf("parameter invalid\r\n");
        return -1;
    }
    if(oob_ssid)
    {
        demo_softap_app_init(oob_ssid, connect_key);
    }
    return 0;
}

if(strcmp(argv[1], "monitor") == 0)
{

```

```
    if(argc != 3)
    {
        os_printf("parameter invalid\r\n");
    }
    if(strcmp(argv[2], "start") == 0)
    {
        bk_wlan_register_monitor_cb(bk_demo_monitor_cb);
        bk_wlan_start_monitor();
    }
    else if(strcmp(argv[2], "stop") == 0)
    {
        bk_wlan_stop_monitor();
    }
    else
    {
        os_printf("parameter invalid\r\n");
    }
}
return 0;
}
MSH_CMD_EXPORT(wifi_demo, wifi_demo command);
```

14.4 操作说明

本节的示例代码均位于**\beken378\demo\ieee802_11_demo.c**,系统默认已经打开此功能,设备上电后,在调试串口输入相应指令即可运行不同程序。

14.4.1 启动STATION连接

设备上电后,调试串口输入**wifi_demo sta your_ssid your_key**,设备开始连接路由器。

```
wifi_demo sta your_ssid your_key
sta_Command
ssid: your_ssid key: your_key
rl_sta_start
[sa_sta]MM_RESET_REQ
[sa_sta]ME_CONFIG_REQ
[sa_sta]ME_CHAN_CONFIG_REQ
[sa_sta]MM_START_REQ
hapd_intf_add_vif,type:2, s:0, id:0
```

```
[wlan_connect]:start tick = 0, connect done tick = 22379, total = 22379
[wlan_connect]:start tick = 0, connect done tick = 22385, total = 22385
[WLAN_MGNT]wlan sta connected evenew dtim period:2
nt callback
IP UP: 192.168.44.27
[ip_up]:start tick = 0, ip_up tick = 25797, total = 25797
```

14.4.2 启动STATION快速连接

设备上电后，在调试串口输入wifi_demo adv your_ssid your_key,设备开始连接路由器，设备log如下：

```
wifi_demo adv your_ssid your_key
sta_adv_Command
[sa_sta]MM_RESET_REQ
[sa_sta]ME_CONFIG_REQ
[sa_sta]ME_CHAN_CONFIG_REQ
[sa_sta]MM_START_REQ
bssid 48-ee-0c-48-93-12
security2cipher 2 3 24 8 security=6
cipher2security 2 3 24 8
-----SM_CONNECT_IND_ok
wpa_driver_assoc_cb
Cancelling scan request
hapd_intf_add_key CCMP
add sta_mgmt_get_sta
sta:1, vif:0, key:0
sta_mgmt_add_key
add hw key idx:25
add TKIP
add is_broadcast_ether_addr
sta:255, vif:0, key:1
add hw key idx:1
ctrl_port_hdl:1
[wlan_connect]:start tick = 0, connect done tick = 31898, total = 31898
[wlan_connect]:start tick = 0, connect done tick = 31904, total = 31904
[WLAN_MGNT]wlan sta connected event callback
sta_ip_start
configuring interface wlan (with DHCP client)
dhcp_check_status_init_timer
IP UP: 192.168.44.49
```

```
[ip_up]:start tick = 0, ip_up tick = 35292, total = 35292
```

14.4.3 STATION模式获取状态

- 获取网络状态

连接路由器，方法参考14.4.1小节，然后串口输入wifi_demo status net获取设备当前网络状态，设备log如下：

```
msh />
msh />wifi_demo status net
dhcp=0 ip=192.168.44.52 gate=192.168.44.119 mask=255.255.255.0 mac=c8:47:8c:2f:4b:d2
```

图14.4.3-1

- 获取连接状态

连接路由器，方法参考14.4.1小节，然后串口输入wifi_demo status link获取设备当前连接状态，设备log如下：

```
msh />
msh />wifi_demo status link
sta: 1, softap: 0, b/g/n
sta:rssi=-65,ssid=wifi-team,bssid=c0:3f:0e:c7:91:4c ,channel=11,cipher_type:MIXED
```

图14.4.3-2

14.4.4 启动AP

设备上电后，在调试串口输入wifi_demo softap beken 12345678,设备开始连接路由器，设备log如下：

```
wifi_demo softap beken 12345678
SOFTAP_COMMAND

ssid:beken key:12345678
rl_ap_start
Soft_AP_start
[saap]MM_RESET_REQ
[saap]ME_CONFIG_REQ
[saap]ME_CHAN_CONFIG_REQ
[saap]MM_START_REQ
hapd_intf_add_vif,type:3, s:0, id:0
apm start with vif:0
-----beacon_int_set:100 TU
update_ongoing_1_bcn_update
vif_idx:0, ch_idx:0, bcmc_idx:2
update_ongoing_1_bcn_update
hapd_intf_add_key CCMP
add_is_broadcast_ether_addr
sta:255, vif:0, key:1
add_hw_key_idx:1
uap_ip_start
```

图14.4.4-1

14.4.5 AP模式获取状态

- 获取网络状态

连接路由器，方法参考14.4.4小节，然后串口输入wifi_demo status net获取设备当前网络状态，设备log如下：

```
msh />
msh />wifi_demo status net
dhcp=0 ip=20.240.159.229 gate=20.240.159.229 mask=20.240.159.229 mac=00:00:00:00:00:00
```

图14.4.5-1

- 获取连接状态

连接路由器，方法参考14.4.4小节，然后串口输入wifi_demo status link获取设备当前连接状态，设备log如下：

```
msh />wifi_demo status link
sta: 0, softap: 1, b/g/n
softap:ssid=beken,channel=11,dhcp=1,cipher_type:CCMP
ip=192.168.0.1,gate=255.255.255.255,mask=255.255.255.0,dns=0.0.0.0
```

图14.4.5-2

14.4.6 启动SCAN

- 扫描WIFI热点

设备上电后，在调试串口输入wifi_demo scan,设备开始扫描附近WIFI热点，设备log如下：

```
msh />wifi_demo scan
[sa_sta]MM_RESET_REQ
[sa_sta]ME_CONFIG_REQ
[sa_sta]ME_CHAN_CONFIG_REQ
[sa_sta]MM_START_REQ
scan_start_req_handler
msh />ethernetif_input no netif found 255
Got ap count: 13
Xiaomi_E21E, RSSI=210
labast, RSSI=206
HUAWEI-EZ7HKY, RSSI=204
FAST_5AC4, RSSI=200
ssid-tcj, RSSI=197
B-LINK_F11566, RSSI=196
antbang_195F4C, RSSI=193
bk7252_smart, RSSI=193
wifi-team, RSSI=192
beken_airport, RSSI=189
Honor Magic 2, RSSI=189
Bekencorp-Guest, RSSI=187
Bekencorp-WIFI, RSSI=177
Get ap end.....
```

图14.4.6-1

- 扫描指定WIFI热点

设备上电后，在调试串口输入wifi_demo scan Bekencorp-WIFI,设备开始扫描Bekencorp-WIFI，设备log如下：

```
msh />wifi_demo scan Bekencorp-WIFI
scan for ssid:Bekencorp-WIFI
[sa_sta]MM_RESET_REQ
[sa_sta]ME_CONFIG_REQ
[sa_sta]ME_CHAN_CONFIG_REQ
[sa_sta]MM_START_REQ
scan_start_req_handler
msh />Got ap count: 1
Bekencorp-WIFI, RSSI=186
Get ap end.....
```

图14.4.6-2

14.4.7 启动混杂包监听

设备上电后，在调试串口输入wifi_demo monitor start,设备开始监听混杂包，输入wifi_demo monitor stop,停止监听，设备log如下：

```
wifi_demo adv your_ssid your_key
```

```
msh />wifi_demo monitor
```

```
parameter invalid
```

```
parameter invalid
```

```
msh />wifi_demo monitor start
```

```
net_wlan_add_netif not vif idx found
```

```
Soft_AP_start
```

```
[saap]MM_RESET_REQ
```

```
[saap]ME_CONFIG_REQ
```

```
[saap]ME_CHAN_CONFIG_REQ
```

```
[saap]MM_START_REQ
```

```
apm start with vif:0
```

```
-----beacon_int_set:100 TU
```

```
update_ongoing_1_bcn_update
```

```
hal_machw_enter_monitor_mode
```

```
msh />len:136
```

```
len:260
```

```
len:166
```

```
len:173
```

```
len:225
```

```
len:136
```

```
len:270
```

```
len:260
```

```
len:166
```

```
len:270
```

```
len:173
```

```
len:136
```

```
len:225
```

```
len:260
```

```
len:225
```

```
wifi_demo monitor stop
```

```
msh />
```

15 RTOS接口

15.1 RTOS接口简介

RTOS接口提供RTOS的操作API，包括线程，互斥锁，时钟，信号量的操作。

15.2 RTOS Related APIs

RTOS相关接口参考beken378\rtos\include\bk_rtos_pub.h，应用程序可通过以下APIs控制RTOS，相关接口如下所示：

函数	描述
bk_rtos_create_thread()	创建一个新的线程
bk_rtos_delete_thread()	删除一个使用结束的线程
bk_rtos_thread_join()	使当前线程挂起，等待另一个线程终止
bk_rtos_thread_sleep()	使一个线程挂起一段时间，时间单位是：秒
bk_rtos_init_semaphore()	初始化一个信号量，并提供一个最大数
bk_rtos_set_semaphore()	发出信号量
bk_rtos_get_semaphore()	获取一个信号量，并提供超时机制
bk_rtos_deinit_semaphore()	销毁一个信号量
bk_rtos_init_mutex()	初始化一个互斥锁
bk_rtos_lock_mutex()	获得一个互斥锁
bk_rtos_unlock_mutex()	释放一个互斥锁
bk_rtos_deinit_mutex()	销毁一个互斥锁
bk_rtos_init_queue()	初始化一个消息队列
bk_rtos_push_to_queue()	将一个数据对象推入消息队列
bk_rtos_pop_from_queue()	从消息队列中取出一个数据对象
bk_rtos_deinit_queue()	销毁一个消息队列
bk_rtos_is_queue_empty()	查询一个队列是否为空
bk_rtos_is_queue_full()	查询一个队列是否已满
bk_rtos_init_timer()	初始化一个时钟，并传入回调函数
bk_rtos_start_timer()	启动一个时钟
bk_rtos_stop_timer()	停止一个时钟
bk_rtos_reload_timer()	重新加载一个过期的时钟
bk_rtos_deinit_timer()	销毁一个时钟
bk_rtos_is_timer_running()	获取一个时钟是否正在运行

15.2.1 RTOS结构体说明

beken_timer_t:



handle	rtos_init_timer创建的时钟句柄
function	时钟回调函数
arg	回调函数的参数
beken_worker_thread_t:	
thread	指向线程的指针
event_queue	线程的事件队列
beken_timed_event_t:	
function	事件句柄函数
arg	事件句柄函数参数
timer	时钟
thread	线程
beken2_timer_t:	
handle	指向时钟的句柄指针
function	时钟事件对应的回调函数，该函数有两个参数
left_arg	回调函数的第一个参数
right_arg	回调函数的第二个参数
beken_magic	魔术数

15.2.2 创建一个新的线程

```
OSStatus bk_rtos_create_thread( beken_thread_t* thread,
                                uint8_t priority,
                                const char* name,
                                beken_thread_function_t function,
                                uint32_t stack_size,
                                beken_thread_arg_t arg );
```

参数	描述
thread	beken_thread_t类型的指针，指向创建的线程句柄
priority	优先级数值越小，优先级越高。
name	线程的名字
function	线程的入口函数
stack_size	为该线程分配的堆栈大小
arg	线程入口函数的参数
返回	kNoErr: 成功；其他：失败

15.2.3 删除一个使用结束的线程

```
OSStatus bk_rtos_delete_thread( beken_thread_t* thread );
```

参数	描述
thread	beken_thread_t类型的指针，指向需要删除的线程句柄
返回	kNoErr: 成功；其他：失败

15.2.4 使当前线程挂起，等待另一个线程终止

```
OSStatus bk_rtos_thread_join(beken_thread_t* thread);
```

参数	描述
thread	beken_thread_t类型的指针，指向需要等待的线程句柄
返回	kNoErr: 成功；其他：失败

15.2.5 使一个线程挂起一段时间

```
void bk_rtos_thread_sleep(uint32_t seconds);
```

参数	描述
seconds	线程挂起的时间，单位是秒。
返回	无

15.2.6 初始化一个信号量

```
OSStatus bk_rtos_init_semaphore( beken_semaphore_t* semaphore, int maxCount );
```

参数	描述
semaphore	初始化的信号量。
返回	kNoErr: 成功；其他：失败

15.2.7 发出信号量

```
int bk_rtos_set_semaphore( beken_semaphore_t* semaphore );
```

参数	描述
semaphore	需要发出的信号量。
返回	kNoErr: 成功；其他：失败

15.2.8 获取一个信号量，并提供超时机制

```
OSStatus bk_rtos_get_semaphore( beken_semaphore_t* semaphore, uint32_t timeout_ms );
```

参数	描述
semaphore	需要获取的信号量。
timeout_ms	超时时间。
返回	kNoErr: 成功；其他：失败

15.2.9 销毁一个信号量

```
OSStatus bk_rtos_deinit_semaphore( beken_semaphore_t* semaphore );
```

参数	描述
semaphore	需要销毁的信号量。
返回	kNoErr: 成功；其他：失败

15.2.10 初始化一个互斥锁

```
OSStatus bk_rtos_init_mutex( beken_mutex_t* mutex );
```

参数	描述
mutex	指向要初始化的互斥锁的句柄的指针。
返回	kNoErr: 成功；其他：失败

15.2.11 获得一个互斥锁

```
OSStatus bk_rtos_lock_mutex( beken_mutex_t* mutex );
```

参数	描述
mutex	指向要获取的互斥锁的句柄的指针。
返回	kNoErr: 成功；其他：失败

15.2.12 释放一个互斥锁

```
OSStatus bk_rtos_unlock_mutex( beken_mutex_t* mutex );
```

参数	描述
mutex	指向要释放的互斥锁的句柄的指针。
返回	kNoErr: 成功；其他：失败

参数	描述
queue	指向要取出数据对象的消息队列的句柄的指针。
message	要获取的数据对象，因此必须保证此缓存区足够大，否则将导致内存崩溃。
timeout_ms	返回前等待的毫秒数。
返回	kNoErr: 成功；其他：失败

15.2.17 销毁一个消息队列

```
OSStatus bk_rtos_deinit_queue( beken_queue_t* queue );
```

参数	描述
queue	指向要销毁的消息队列的句柄的指针。
返回	kNoErr: 成功；其他：失败

15.2.18 查询一个队列是否为空

```
BOOL bk_rtos_is_queue_empty( beken_queue_t* queue );
```

参数	描述
queue	指向要查询的消息队列的句柄的指针。
返回	1: 空；0: 非空

15.2.19 查询一个队列是否已满

```
BOOL bk_rtos_is_queue_full( beken_queue_t* queue );
```

参数	描述
queue	指向要查询的消息队列的句柄的指针。
返回	1: 满；0: 不满

15.2.20 初始化一个时钟，并传入回调函数

```
OSStatus bk_rtos_init_timer( beken_timer_t *timer,  
                             uint32_t time_ms,  
                             timer_handler_t function,  
                             void* arg );
```

参数	描述
timer	指向要创建的时钟的句柄的指针。

time_ms	时钟，单位是毫秒。
function	时钟到期后执行的回调函数
arg	回调函数的参数
返回	kNoErr：成功；其他：失败

15.2.21 启动一个时钟

```
OSStatus bk_rtos_start_timer( beken_timer_t* timer );
```

参数	描述
timer	指向要启动的时钟的句柄的指针。
返回	kNoErr：成功；其他：失败

15.2.22 停止一个时钟

```
OSStatus bk_rtos_stop_timer( beken_timer_t* timer );
```

参数	描述
timer	指向要停止的时钟的句柄指针。
返回	kNoErr：成功；其他：失败

15.2.23 重新加载一个过期的时钟

```
OSStatus bk_rtos_reload_timer( beken_timer_t* timer );
```

参数	描述
timer	指向要加载的时钟的句柄指针。
返回	kNoErr：成功；其他：失败

15.2.24 销毁一个时钟

```
OSStatus bk_rtos_deinit_timer( beken_timer_t* timer );
```

参数	描述
timer	指向要销毁的时钟的句柄指针。
返回	kNoErr：成功；其他：失败

15.2.25 获取一个时钟是否正在运行

```
BOOL bk_rtos_is_timer_running( beken_timer_t* timer );
```

参数	描述
----	----

timer	指向要查询的时钟的句柄指针。
返回	1: 运行; 其他: 停止

15.3 RTOS关键说明

• RTOS宏定义说明

执行返回值:

#define RTOS_SUCCESS (1)	<i>/*执行成功*/</i>
#define RTOS_FAILURE (0)	<i>/*执行失败*/</i>

RTOS优先级配置:

#define BEKEN_DEFAULT_WORKER_PRIORITY (6)	<i>/*默认优先级为6*/</i>
#define BEKEN_APPLICATION_PRIORITY (7)	<i>/*应用优先级为7*/</i>

RTOS时间配置:

#define kNanosecondsPerSecond	1000000000UUL
#define kMicrosecondsPerSecond	1000000UL
#define kMillisecondsPerSecond	1000
#define NANOSECONDS	1000000UL
#define MICROSECONDS	1000
#define MILLISECONDS	(1)
#define SECONDS	(1000)
#define MINUTES	(60 * SECONDS)
#define HOURS	(60 * MINUTES)
#define DAYS	(24 * HOURS)

RTOS等待配置:

#define BEKEN_NEVER_TIMEOUT	(0xFFFFFFFF)
#define BEKEN_WAIT_FOREVER	(0xFFFFFFFF)
#define BEKEN_NO_WAIT	(0)

• RTOS枚举类型说明

等待事件说明如下所示:

```
typedef enum
{
    WAIT_FOR_ANY_EVENT,    /*任何事件可唤醒*/
    WAIT_FOR_ALL_EVENTS,   /*所有事件可唤醒*/
} beken_event_flags_wait_option_t;
```

16 OTA

16.1 OTA简介

支持网络端远程升级固件，采用http协议从服务器下载ota固件，然后烧录到download分区中，设备重启后bootloader会将ota分区的固件拷贝到app运行分区，并加载新的app分区固件。ota固件支持压缩和加密，ota固件制作使用rt_ota_pac aging_tool工具。

16.2 OTA Related API

OTA相关接口参考\rt-thread\samples\ota\http\http_client_ota.c, 应用程序可通过以下APIs使用OTA，相关接口如下所示：

函数	描述
<code>fal_init()</code>	fal初始化
<code>http_ota_fw_download()</code>	远程下载固件

16.2.1 fal初始化

初始化所有flash设备和分区，必须在http_ota_fw_download之前被调用，函数如下所示：

```
int fal_init(void);
```

参数	描述
<code>void</code>	无
返回	总分区数目

16.2.2 远程下载固件

从服务器下载ota固件，然后烧录到download分区中，调用之前应该初始化fal使用fal_init函数，函数如下所示：

```
int http_ota_fw_download(const char *url);
```

参数	描述
<code>url</code>	http服务器上的文件地址，完整的url
返回	0

16.3 OTA示例代码

OTA示例代码参考\samples\ota\http\ http_client_ota.c，具体使用方式可以参考如下示例代码：

```
/*
 * 程序清单： 这是一个ota使用例程
 * 例程导出了http_ota 命令到控制终端
 * 命令调用格式： http_ota url
 * 程序功能： 通过ota下载远程固件到download分区
 */

void http_ota(uint8_t argc, char **argv)
{
    int parts_num;
    parts_num = fal_init();    //fal初始化

    if (parts_num <= 0)
    {
        log_e("Initialize failed! Don't found the partition table.");
        return;
    }
    if (argc < 2)
    {
        rt_kprintf("using url: " HTTP_OTA_URL "\n");
        http_ota_fw_download(HTTP_OTA_URL);    //固件下载
    }
    else
    {
        http_ota_fw_download(argv[1]);
    }
}

/**
 * msh />http_ota [url]
 */
MSH_CMD_EXPORT(http_ota, OTA by http client: http_ota [url]);
```

16.4 操作说明

16.4.1 生成rbl升级文件

使用“rt_ota_packaging_tool”，可以生成rbl文件，配置如下：



图16.4.1-1

其中：压缩算法一定要选gzip，加密算法选AES256，加密密钥与加密IV应与boot中的对应。

```
static const uint8_t iv_table[16 + 1] = "0123456789ABCDEF";  
static const uint8_t key_table[32 + 1] =  
"0123456789ABCDEF0123456789ABCDEF";
```

16.4.2 搭建本地HTTP Server环境

如果没有HTTP服务器，可以先搭建本地HTTP Server环境进行http_ota的功能验证。打开“MyWebServer3621.exe”，界面如下所示：

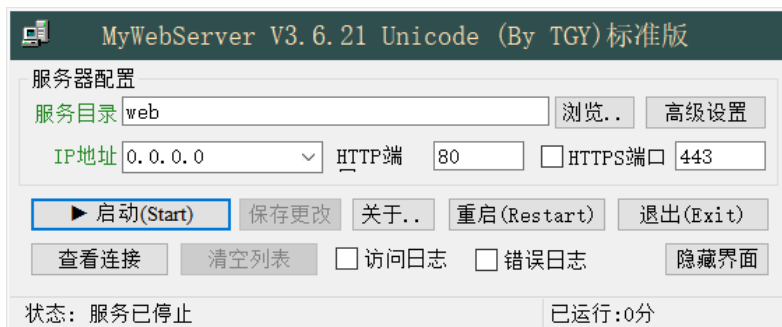


图16.4.2-1

点击浏览按钮，选择存放3.4.1生成的rbl文件的路径，修改端口为8080，默认是80，然后点击启动按钮，



图16.4.1-2

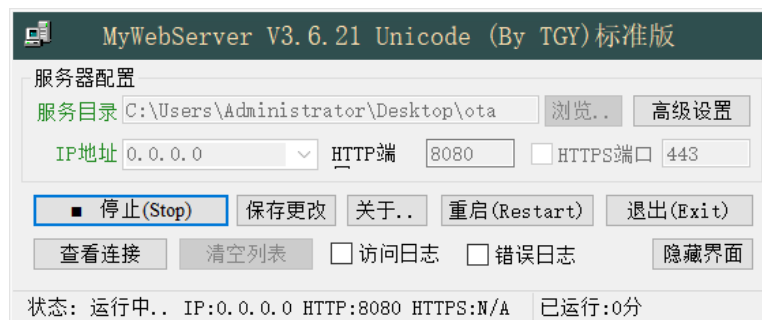


图16.4.1-3

url为http://local_ip:8080/rtthread.rbl,浏览器输入后可以下载到rtthread.rbl说明环境搭建成功。

16.4.3 运行现象

• 连接路由器

设备上电后，调试串口输入wifi_demo sta your_ssid your_key,设备开始连接路由器。

```
wifi_demo sta your_ssid your_key
sta_Command
ssid: your_ssid key: your_key
rl_sta_start
[sa_sta]MM_RESET_REQ
[sa_sta]ME_CONFIG_REQ
[sa_sta]ME_CHAN_CONFIG_REQ
[sa_sta]MM_START_REQ
hapd_intf_add_vif,type:2, s:0, id:0
[wlan_connect]:start tick = 0, connect done tick = 22379, total = 22379
[wlan_connect]:start tick = 0, connect done tick = 22385, total = 22385
[WLAN_MGNT]wlan sta connected evenew dtim period:2
nt callback
```



IP UP: 192.168.44.27

[ip_up]:start tick = 0, ip_up tick = 25797, total = 25797

• OTA升级

调试串口输入http_ota http://local_ip:8080/rtthread.rbl, 设备log如下所示:

http_ota http://local_ip:8080/rtthread.rtl

current firmware name: app, version: 2M.1220, timestamp: 1568000867

dl_part->name : download

dl_part->flash : beken_onchip

dl_part->offset: 0x00143000

dl_part->len : 665600

[I/HTTP_OTA] OTA file size is (624560)

[I/HTTP_OTA] OTA file raw size 755580 bytes.

[I/HTTP_OTA] OTA file describe partition name app, version: 2M.1221, timestamp: 1568000867.

[I/HTTP_OTA] FLASH_PROTECT_HALF.

Download:

[=====]
=====] 100%

[I/OTA] Verify 'download' partition(fw ver: 2M.1221, timestamp: 1568000867) success.

[I/HTTP_OTA] FLASH_UNPROTECT_LAST_BLOCK.

reboot system

.....

17 Bootloader

17.1 Bootloader简介

bootloader分成两级，一级为L_boot，二级为UP_boot。一级boot提供uart下载功能，二级boot实现ota功能。在使用bootloader之前，需要先根据项目的情况确定分区，并把分区信息保存到原始的bootloader.bin中。

17.2 分区表的设置

17.2.1 Bootloader分区

flash_name为beken_onchip_crc，所以offset = 0x10000是逻辑地址，该分区在FLASH中实际的物理地址为也为0,分区实际大小len = $(60K * 34) / 32 = 65280Byte$;

17.2.2 App分区

该分区为应用代码。flash_name为beken_onchip_crc，所以offset = 0x10000是逻辑地址，该分区在FLASH中实际的物理地址为： $(0x10000 * 34) / 32 = 0x0011000$ ，分区实际大小len = $(1152K * 34) / 32 = 1224 K$;

17.2.3 Download分区

该分区为OTA时下载数据存放区。flash_name为beken_onchip，所以offset = 0x143000为该分区在FLASH中实际的物理地址，分区实际大小为748K。

除了以上3个分区之外，可以根据需求添加其它分区。另外，bootloader分区的起始地址和长度不能变，app分区的起始地址不能变，但长度可以变。其它分区的起始地址和长度都可以根据方案的实际情况进行修改。

以BK7251 SDK中提供的2M分区表信息为例（partition_audio_2M.json），对其格式的解释如下：

字段	描述
name	分区名称，固件中查找分区的依据，不能重复
flash_name	所在介质名称，通常为FLASH。常用beken_onchip_crc与beken_onchip。对于前者，其offset和len字段都以逻辑地址表示，对于后者则是以物理地址

	表示
offset	分区起始地址，十六进制表示
len	分区长度，十进制表示

17.3 L_boot

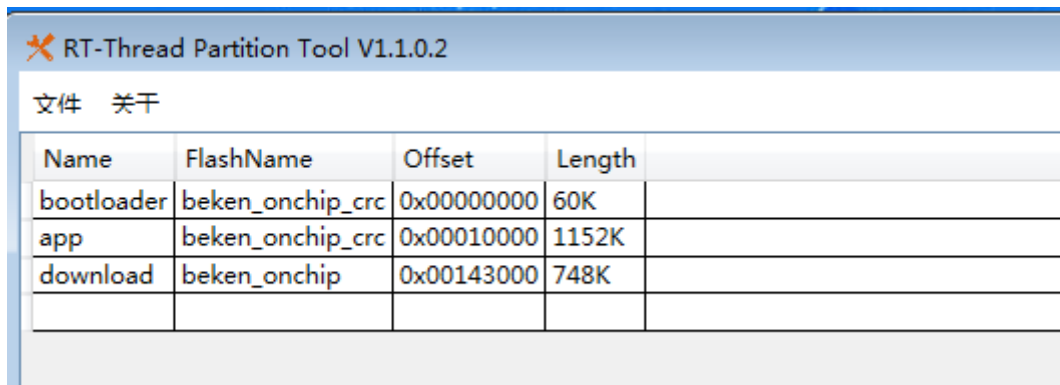
一级boot文件位于packages\boot\l_boot.bin,包含uart下载功能。一级boot应该被烧录到flash 0地址处，运行完成跳转到二级boot: CPU地址0x1F00处。

17.4 UP_boot

UP_boot必须从地址0x1F00处开始，UP_boot支持rttos的ota升级功能, ota升级功能会将download分区的rbl文件解密并解压到OS执行分区。二级boot运行完成后跳转到OS分区: CPU地址0x10000处。二级boot文件位于packages\boot\up_boot.bin。

17.5 获取bootloader.bin文件

打开rt_partition_tool软件，加载原始的bootloader.bin，然后导入分区表partition_audio_2M_sd.json，最后把分区表保存到bootloader.bin中，操作完成后该bootloader.bin即可和应用代码一起通过打包工具beken_packager生成最终的bin文件。



The screenshot shows the RT-Thread Partition Tool V1.1.0.2 interface. It displays a table with the following data:

Name	FlashName	Offset	Length
bootloader	beken_onchip_crc	0x00000000	60K
app	beken_onchip_crc	0x00010000	1152K
download	beken_onchip	0x00143000	748K

图17.5-1

17.6 生成all.bin文件

在得到有分区表的bootloader bin文件之后，就可以通过打包工具生成完整的bin文件。执行SDK目录\tool\beken_packager下的打包工具beken_packager.exe，即可生成完整的bin文件all_2M.1220.bin，以及串口升级所用的bin文件rtthread_uart_2M.1220.bin。

对于生成all.bin的config.json文件说明如下：

字段	描述
firmware	各分区打包输入的Bin文件
version	版本号
partition	分区名称，与bootloader.bin中对应分区表的名称相同
start_addr	分区起始地址，为物理地址，以十六进制表示，与分区表中对应分区的物理起始地址相同
size	分区实际大小，十进制表示，与分区表中对应分区的实际长度相同

17.7 Bootloader示例代码

17.7.1 2M分区表信息配置文件partition_audio_2M.json示例

```
{
  "part_table": [
    {
      "name": "bootloader",
      "flash_name": "beken_onchip_crc",
      "offset": "0x00000000",
      "len": "60K"
    },
    {
      "name": "app",
      "flash_name": "beken_onchip_crc",
      "offset": "0x00010000",
      "len": "1152K"
    },
    {
      "name": "download",
      "flash_name": "beken_onchip",
      "offset": "0x00143000",
      "len": "748K"
    }
  ]
}
```

17.7.2 UP_boot示例

```
/*
 * 程序清单： 这是一个二级boot使用例程
 * 程序功能： 程序实现了ota加密，解压拷贝分区等工作
 */
int ota_main(UINT32 * ex)
{
    int result = 0;
    size_t i, part_table_size;
    const struct fal_partition *dl_part = NULL;
    const struct fal_partition *part_table = NULL;
    const char *dest_part_name = NULL;

    if (rt_ota_init() >= 0)
    {
        /* verify bootloader partition
         * 1. Check if the BL partition exists
         * 2. CRC BL FW HDR
         * 3. HASH BL FW
         */
        if (rt_ota_part_fw_verify_header(fal_partition_find(RT_BK_BL_PART_NAME)) < 0)
        {
            //TODO upgrade bootloader to safe image
            // firmware HDR crc failed or hash failed. if boot verify failed, may not jump to app
            running
            #if !BOOT_OTA_DEBUG // close debug
                return -1;
            #endif
        }

        // 4. Check if the download partition exists
        dl_part = fal_partition_find(RT_BK_DL_PART_NAME);
        if (!dl_part)
        {
            log_e("download partition is not exist, please check your configuration!");
            return -1;
        }

        /* 5. Check if the target partition name is bootloader, skip ota upgrade if yes */
    }
```



```
dest_part_name = rt_ota_get_fw_dest_part_name(dl_part);
if (dest_part_name && !strcmp(dest_part_name, RT_BK_BL_PART_NAME,
strlen(RT_BK_BL_PART_NAME)))
{
    log_e("Can not upgrade bootloader partition!");
    goto _app_check;
}

/* do upgrade when check upgrade OK
* 5. CRC DL FW HDR
* 6. Check if the dest partition exists
* 7. CRC APP FW HDR
* 8. Compare DL and APP HDR, containing fw version
*/
log_d("check upgrade...");
if ((result = rt_ota_check_upgrade()) == 1) // need to upgrade
{
    if((rt_ota_get_fw_algo(dl_part) & RT_OTA_CRYPT_STAT_MASK) ==
RT_OTA_CRYPT_ALGO_NONE)
    {
        log_e("none encryption Not allow!");
        goto _app_check;
    }

    /* verify OTA download partition
    * 9. CRC DL FW HDR
    * 10. CRC DL FW
    */
    if (rt_ota_part_fw_verify(dl_part) == 0)
    {
        // 11. rt_ota_custom_verify
        // 12. upgrade
        set_flash_protect(NONE);
        if (rt_ota_upgrade() < 0)
        {
            log_e("OTA upgrade failed!");
            /*
            * upgrade failed, goto app check. If success, jump to app to run, otherwise
            goto recovery factory firmware.
            */

```

```
        goto _app_check;
    }
    ota_erase_dl_rbl();
}
else
{
    goto _app_check;
}
}
else if (result == 0)
{
    log_d("No firmware upgrade!");
}
else if (result == -1)
{
    goto _app_check;
}
else
{
    log_e("OTA upgrade failed! Need to recovery factory firmware.");
    return -1;
}
```

_app_check:

```
part_table = fal_get_partition_table(&part_table_size);
/* verify all partition */
for (i = 0; i < part_table_size; i++)
{
    /* ignore bootloader partition and OTA download partition */
    if (!strcmp(part_table[i].name, RT_BK_APP_NAME, FAL_DEV_NAME_MAX))
    {
        // verify app firmware
        if (rt_ota_part_fw_verify_header(&part_table[i]) < 0)
        {
            // TODO upgrade to safe image
            log_e("App verify failed! Need to recovery factory firmware.");
            return -1;
        }
    }
    else
    {
```

```
        *ex = part_table[i].offset;
        result = 0;
    }
}
}
else
{
    result = -1;
}

return result;
}
```

17.7.3 生成all.bin的配置文件config_sample.json示例

```
{
  "magic": "RT-Thread",
  "version": "0.1",
  "count": 2,
  "section": [
    {
      "firmware": "bootloader.bin",
      "version": "2M.1220",
      "partition": "bootloader",
      "start_addr": "0x00000000",
      "size": "65280"
    },
    {
      "firmware": "../rtthread.bin",
      "version": "2M.1220",
      "partition": "app",
      "start_addr": "0x00011000",
      "size": "1224K"
    }
  ]
}
```

18 低功耗

18.1 低功耗简介

BK7251低功耗模式包括了MCU睡眠，RF睡眠以及Deep Sleep睡眠模式，Deep Sleep唤醒模式包括RTC唤醒和GPIO唤醒。

18.2 低功耗 Related API

低功耗相关接口参考**beken378\func\include\wlan_ui_pub.h** 和 **manual_ps_pub.h**，相关接口如下：

函数	描述
bk_wlan_enter_powersave()	低功耗模式
bk_enter_deep_sleep_mode()	deep_sleep模式

18.2.1 进入低功耗模式

进入低功耗模式的函数如下所示：

```
int bk_wlan_enter_powersave(struct rt_wlan_device *device, int level);
```

参数	描述
struct rt_wlan_device *device	wlan设备句柄
level	0: mcu,rf都不睡眠; 1: mcu睡眠, rf不睡眠; 2: mcu不睡眠, rf睡眠 3: mcu, rf都睡眠
返回	RT_EOK(0): 成功; 其他: 出错

18.2.2 deep_sleep 模式

进入deep_sleep 模式的函数如下所示：

```
void bk_enter_deep_sleep_mode(PS_DEEP_CTRL_PARAM *deep_param);
```

参数	描述
PS_DEEP_CTRL_PARAM *deep_param	进入deep_sleep之前的参数设置
返回	空

参数类型

PS_DEEP_CTRL_PARAM:

PS_DEEP_WAKEUP_WAY deep_wkway 唤醒模式的枚举类型

UINT32 gpio_index_map 每个bit位对应gpio0-gpio31, 0: 不被设置;
1: 相应的gpio可以在deep_sleep被唤醒。

UINT32 gpio_edge_map	每个bit位对应gpio0-gpio31唤醒模式, 0: 上升沿唤醒; 1: 下降沿唤醒, 其中gpio1为uart rx, 必须设为1。
UINT32 gpio_last_index_map	低8位bit位对应gpio32-gpio39, 0: 不被设置; 1: 相应的gpio可以在deep_sleep被唤醒。
UINT32 gpio_last_edge_map	低8位bit位对应gpio32-gpio39唤醒模式, 0: 上升沿醒; 1: 下降沿唤醒。
UINT32 sleep_time	timer唤醒模式下的唤醒时间

18.3 低功耗示例代码

mcu睡眠, rf睡眠示例代码参考\test\test_pm.c , deep sleep模式示例代码\test\deep_sleep.c, 打开宏定义: PM_TEST, 开启mcu,rf睡眠功能测试; 打开宏定义: DEEP_SLEEP_TEST, 开启deeo_sleep测试, 示例代码如下:

18.3.1 关键说明

• 低功耗枚举型说明

deep_sleep模式下支持3种唤醒模式:

```
typedef enum {
    PS_DEEP_WAKEUP_GPIO = 1,    /*GPIO唤醒模式
    PS_DEEP_WAKEUP_RTC = 2,     /*RTC timer唤醒模式
    PS_DEEP_WAKEUP_GPIO_RTC = 4, /*USB唤醒模式
} PS_DEEP_WAKEUP_WAY;
```

• 低功耗宏定义

在进入低功耗模式必须开启宏定义: CFG_USE_MCU_PS才能进入低功耗模式。

#define	CFG_USE_MCU_PS	使用MCU的低功耗模式
#define	CFG_USE_STA_PS	使用RF的低功耗模式

18.3.2 示例代码

```
/*
* 程序清单: 这是一个低功耗 和deep_sleep模式的函数
* 命令格式: 输入命令: wifi ap, 再输入命令: wifi w0 join wifiname password 连接网络, 最后输入命令 pm_level level 进入低功耗模式。
           测试deep sleep 模式下, 输入命令: sleep_mode 2 2 0 0 10 deep_wkway进入deep_sleep 模式, deep_wkway选择唤醒模式, 可参考结构体类型说明。
* 程序功能: 实现低功耗和deep_sleep功能
*/
```

```
#include "error.h"
#include "include.h"
#include "arm_arch.h"
#include "gpio_pub.h"
#include "uart_pub.h"
#include "music_msg_pub.h"
#include "manual_ps_pub.h"

#include "co_list.h"
#include "saradc_pub.h"
#include "temp_detect_pub.h"
#include "sys_rtos.h"
#include "rtos_pub.h"
#include "saradc_intf.h"
#include "pwm_pub.h"
#include "pwm.h"
#include <stdint.h>
#include <stdlib.h>
#include <finsh.h>

/* mcu睡眠和rf睡眠模式示例*/
static int pm_level(int argc, char **argv)
{
    uint32_t level;
    if(argc != 2)
    {
        rt_kprintf("input argc is err!\n");
        return -1;
    }
    level = atoi(argv[1]);
    if(level > 3) {
        rt_kprintf("nonsupport level %d\n", level);
        return -1;
    }

    {
        struct rt_wlan_device *sta_device = (struct rt_wlan_device
*)rt_device_find(WIFI_DEVICE_STA_NAME);
        if (NULL != sta_device) {
            bk_wlan_enter_powersave(sta_device, level);
        }
    }
}
```

```
    }

    }

    return 0;
}

static int htoi(char s[])
{
    int i;
    int n = 0;
    if (s[0] == '0' && (s[1]=='x' || s[1]=='X'))
    {
        i = 2;
    }
    else
    {
        i = 0;
    }
    for (; (s[i] >= '0' && s[i] <= '9') || (s[i] >= 'a' && s[i] <= 'z') || (s[i] >= 'A' && s[i] <= 'Z'); ++i)
    {
        if (tolower(s[i]) > '9')
        {
            n = 16 * n + (10 + tolower(s[i]) - 'a');
        }
        else
        {
            n = 16 * n + (tolower(s[i]) - '0');
        }
    }
    return n;
}

static void enter_deep_sleep_test(int argc, char *argv[])
{
    rtos_delay_milliseconds(10);
    PS_DEEP_CTRL_PARAM deep_sleep_param;

    deep_sleep_param.wake_up_way = 0;

    deep_sleep_param.gpio_index_map = htoi(argv[1]);
    deep_sleep_param.gpio_edge_map = htoi(argv[2]);
    deep_sleep_param.gpio_last_index_map = htoi(argv[3]);
}
```

```

deep_sleep_param.gpio_last_edge_map    = htoi(argv[4]);
deep_sleep_param.sleep_time            = htoi(argv[5]);
deep_sleep_param.wake_up_way           = htoi(argv[6]);

if(argc == 7)
{
    rt_kprintf("---deep sleep test param : 0x%0X 0x%0X 0x%0X 0x%0X %d %d\r\n",
               deep_sleep_param.gpio_index_map,
               deep_sleep_param.gpio_edge_map,
               deep_sleep_param.gpio_last_index_map,
               deep_sleep_param.gpio_last_edge_map,
               deep_sleep_param.sleep_time,
               deep_sleep_param.wake_up_way);

    bk_enter_deep_sleep_mode(&deep_sleep_param);
}
else
{
    rt_kprintf("---argc error!!! \r\n");
}
}
FINISH_FUNCTION_EXPORT_ALIAS(enter_deep_sleep_test, __cmd_sleep_mode, test sleep
mode);

```

18.4 操作说明

18.4.1 连接万用表

测量低功耗模式下的电流，需要将电源接到vbat引脚上以及串联万用表，如图：

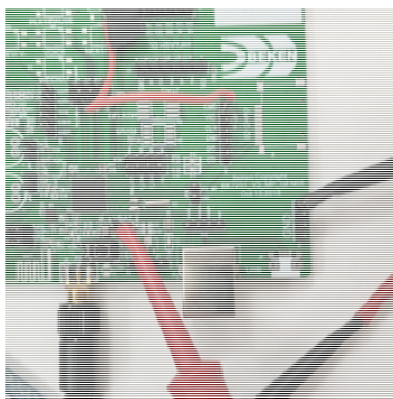


图18.3.2-1

18.4.2 运行现象

- mcu睡眠, rf睡眠示例

设备上电后, 输入命令: `wifi w0 join wifiname password` 连接网络, , 输入命令 `pm_level level` 进入低功耗模式。其中, level值表示含义: 0: mcu, rf都不睡眠; 1: mcu睡眠, rf不睡眠; 2: mcu不睡眠, rf睡眠 3: mcu, rf都睡眠, 可以看到电流表上显示芯片的电流值会发生变化。

- Deep Sleep模式

设备上电后, 输入命令: `sleep_mode 1c 0 1c 0 10 1/2/4` 进入 `deep_sleep` 模式, `deep_wkway`选择唤醒模式, 1:gpio唤醒, 2: rtc唤醒, 4: usb唤醒, 具体可参考结构体类型说明。进入Deep Sleep模式下, 电流可以达到8uA左右。

19 混音

19.1 混音简介

混音的功能是用BK7251芯片连接网络播放音乐,line in 接口接入音频作为背景音频,芯片可以同时播放两种音频数据,也可以消除line in的背景音频。

19.2 混音 Related API

mixer 相关接口参考\function\mixer.h,应用程序可通过以下APIs使用mixer功能,相关接口如下所示:

函数	描述
<code>mixer_init()</code>	混音初始化
<code>mixer_pause()</code>	暂停背景音乐,录音的时候必须暂停
<code>mixer_replay()</code>	重新播放背景音乐

19.2.1 混音初始化

混音模块初始化函数包括了audio延迟初始化,semaphoremutex,mq的创建。

```
uint32_t mixer_init(void);
```

参数	描述
<code>void</code>	空
返回	1(MIXER_SUCCESS): 成功 0(MIXER_FAILURE) : 错误

19.2.2 暂停背景音播放

```
void mixer_pause(void);
```

参数	描述
<code>void</code>	无
返回	void

19.2.3 重新播放背景音

```
void mixer_replay(void);
```

参数	描述
<code>void</code>	无

返回

void

19.3 混音示例代码

19.3.1 关键说明

• 混音宏定义

#define	CONFIG_SOUND_MIXER	必须开启宏定义，进入混音模式
#define	MIXER_FAILURE	1: 返回失败
#define	MIXER_SUCCESS	0: 返回成功

19.3.2 示例代码

```
/*
 * 程序清单： 这是一个混音使用例程，播放设备要同时播放两种音乐，一种音乐使用line in 接口接入
 其他设备播放的音乐，另一种音乐使用云端播放。
 * 命令调用格式： 配网成功之后播放云端的音乐，在输入命令： mixer_set_value 1 停止背景音乐的播
 放 命令mixer_set_value 0 播放背景音乐
 * 程序功能： 例程通过调用命令来控制背景音乐的播放与停止
 */
#include "rtconfig.h"
#if CONFIG_SOUND_MIXER
#include "mixer.h"
void mixer_set_value(int argc, char** argv)
{
    int val;
    val = atoi(argv[1]);
    if(val == 1) {
        rt_kprintf("mixer_set_value:%d pause\r\n", val);
        mixer_pause();                                     /*暂停*/
    } else if(val == 0) {
        rt_kprintf("mixer_set_value:%d replay\r\n", val);
        mixer_replay();                                    /*重新播放*/
    }
}
}
MSH_CMD_EXPORT(mixer_set_value, mixer_set_value test);
```

19.4 操作说明

19.4.1 打开配置

混音示例代码参考\samples\Mixer\mixer_demo.c，打开宏定义：MIXER_DEMO，开启混音功能测试，设备需要播放云端音乐，所以必须开启list player的功能。

19.4.2 运行现象

- 使用配网命令，将设备连网成功，并且混音播放音乐

连网成功后，输入播放云端音乐的命令：play_list, audio out接口接入耳机，可以听到云端播放的音乐；demo板line in 接口需要接入播放音乐的播放设备作为背景音乐，这样可以同时听到云端播放的音乐和其他设备播放的音乐。设备连接如图：

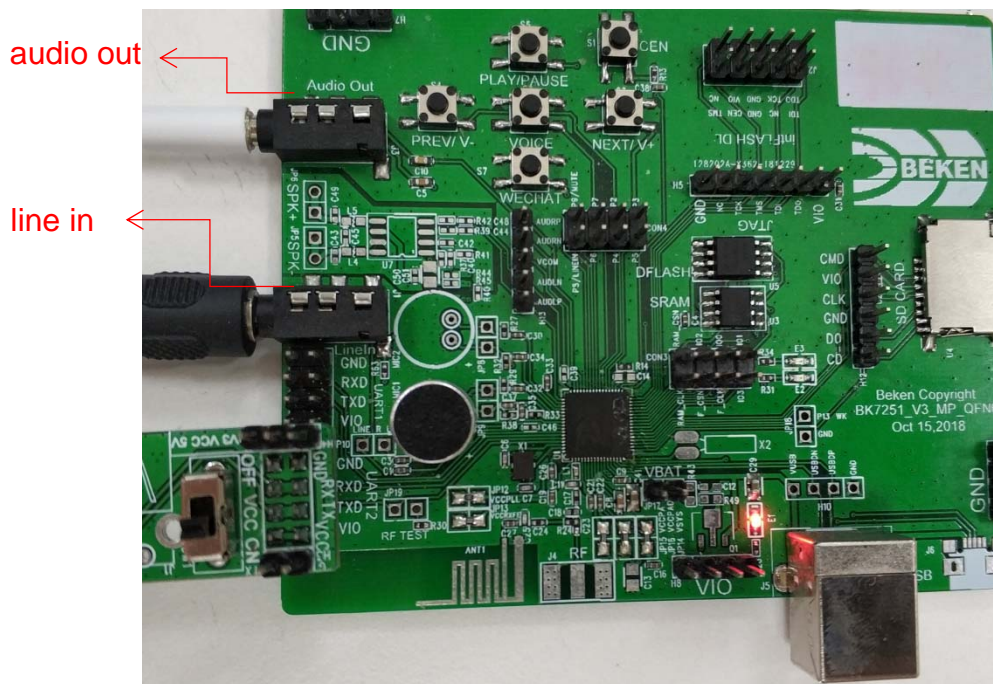


图19.4.1-1

- 发送命令停止混音

输入命令：mixer_set_value 1 停止背景音乐的播放，mixer_set_value 0 播放背景音乐。

20 Airkiss 配网

20.1 Airkiss简介

Airkiss是微信硬件平台提供了一种wifi设备快速入网配置技术，要使用微信客户端的方式配置设备入网，需要设备支持airkiss技术。除了配网之外，还包括进场发现功能，该功能是使用型号码必备的功能，用来绑定设备。

20.2 Airkiss Related API

Airkiss相关接口参考\samples\airkiss\airkiss.h，相关接口如下：

函数	描述
<code>airkiss()</code>	开始airkiss
<code>airkiss_get_status()</code>	获取airkiss状态
<code>airkiss_get_result()</code>	当airkiss_recv()返回AIRKISS_STATUS_COMPLETE后，调用此函数来获取AirKiss解码结果

20.2.1 开始airkiss

```
int airkiss(void);
```

参数	描述
<code>void</code>	空
返回	0:成功；其他：失败

20.2.2 获取airkiss状态

```
uint32_t airkiss_get_status(void);
```

参数	描述
<code>void</code>	空
返回	airkiss状态

20.2.3 获取airkiss解码结果

```
int airkiss_get_result(airkiss_context_t *context, airkiss_result_t *result);
```

参数	描述
<code>airkiss_context_t* context,</code>	为airkiss库分配的内存
<code>airkiss_result_t *result</code>	Airkiss解码后的结果



返回	0:成功；其他：失败
----	------------

参数类型

airkiss_result_t:

char *pwd	wifi密码
char *ssid	wifi ssid
unsigned char pwd_length	wifi密码长度
unsigned char ssid_length	wifi ssid长度
unsigned char random	随机值，根据AirKiss协议，当wifi连接成功后，需要通过udp向10000端口广播这个随机值，这样AirKiss发送端（微信客户端或者AirKissDebugger）就能知道AirKiss已配置成功
unsigned char reserved	保留值

20.3 Airkiss示例代码

20.3.1 关键说明

• Airkiss枚举类型说明

airkiss_status_t: airkiss状态枚举类型

```
typedef enum
{
    /* 解码正常，无需特殊处理，继续调用airkiss_recv()直到解码成功 */
    AIRKISS_STATUS_CONTINUE = 0,
    /* wifi信道已经锁定，上层应该立即停止切换信道 */
    AIRKISS_STATUS_CHANNEL_LOCKED = 1,
    /* 解码成功，可以调用airkiss_get_result()取得结果 */
    AIRKISS_STATUS_COMPLETE = 2
} airkiss_status_t;
```

20.3.2 示例代码

```
/*
 * 程序清单： 这是一个airkiss配网使用例程
 * 命令调用格式： start_airkiss
 * 程序功能： 通过微信平台给设备配网
 */
#include <rtthread.h>
#include <rtdevice.h>
#include <rthw.h>
#include <wlan_dev.h>
```

```
#include <wlan_mgnt.h>
#include "airkiss.h"
#include "bk_rtos_pub.h"
#include <stdio.h>
#include <sys/socket.h>
#include "error.h"

int start_airkiss(int argc, char *argv[])
{
    if(g_cfg_done_sem == RT_NULL)
    {
        if(1 == airkiss())
        {
            rt_kprintf("airkiss start\r\n");

            rt_thread_delay(rt_tick_from_millisecond(1000));

            while(g_cfg_done_sem)
            {
                uint32_t res;
                res = airkiss_get_status();
                if(res == AIRKISS_STATUS_COMPLETE)
                {
                    airkiss_result_t *result;
                    result = airkiss_result_get();
                    rt_kprintf("---ssid:%s , key:%s---\r\n", result->ssid, result->pwd);
                    break;
                }

                rt_thread_delay(rt_tick_from_millisecond(100));
            }
        }
        else
            rt_kprintf("airkiss fail\r\n");
    }
}

#ifdef FINSH_USING_MSH
#include "finsh.h"

MSH_CMD_EXPORT(start_airkiss, start_arksss);
```

```
#endif
```

20.4 操作说明

示例代码位于\samples\airkiss目录下，打开宏定义：RT_USING_AIRKISS，开启Airkiss配网测试。需要在调试串口输入触发命令使设备进入Airkiss配网模式，然后操作APP进行配网。

20.4.1 扫描微信airkiss配网二维码或下载Airkiss调试工具



图20.4.1-1

微信官方Airkiss调试工具：[下载地址](#)
进入下载页面后，下载下图所示工具：

WiFi设备

AirKiss技术简介：[下载](#)

AirKiss调试工具：[下载](#)

AirLink调试工具：[下载](#)

图20.4.1-2

32.4.2 运行现象

- 设备触发配网

编译下载运行后，在调试串口输入命令start_airkiss，程序运行日志如下所示：


```
airkiss start
Airkiss version: airkiss-2.0.0-25360(Dec 17 2015 17:20:50);arm-none-eabi/gcc-4.9.3;ARM
[DRV_WLAN]set monitor callback
[DRV_WLAN]start monitor
Soft_AP_start
[saap]MM_RESET_REQ
[saap]ME_CONFIG_REQ
[saap]ME_CHAN_CONFIG_REQ
[saap]MM_START_REQ
apm start with vif:0
-----beacon_int:100 TU
update_ongoing_1_bcn_update
hal_machw_enter_monitor_mode
Switch channel 2
Switch channel 3
Switch channel 4
Switch channel 5
Switch channel 6
Switch channel 7
Switch channel 8
Switch channel 9
Switch channel 10
Switch channel 11
Switch channel 12
Switch channel 13
Switch channel 1
Switch channel 2
Switch channel 3
Switch channel 4
Switch channel 5
Switch channel 6
Switch channel 7
Switch channel 8
Switch channel 9
Switch channel 10
Switch channel 11
Switch channel 12
Switch channel 13
Switch channel 1
```

图20.4.2-1

• APP配网

打开调试APP，填入手机连接路由器的密码，点击发送，如下图所示：



图20.4.2-2



图20.4.2-3

- 配网完成

设备收到APP下发的路由器ssid和key后，显示日志如下：

```
Switch channel 9
Lock channel in 9
---vbat voltage:3084---
---vbat voltage:3086---
airkiss_get_result() ok!
  ssid = wifi-team
  pwd = stm32f215
  , ssid_length = 9
  pwd_length = 9
  random = 0x39
[DRV_WLAN]stop monitor
[DRV_WLAN]set monitor callback
[DRV_WLAN]drivers\wlan\drv_wlan.c L922 beken_wlan_control cmd: case WIFI_INIT!
[wifi_connect]: fast connect
psk = 4ebe695d2af768a2ebd94fcd165daaf7806fedefe956d54ea87e07b32316a813
[sa_sta]MM_RESET_REQ
[sa_sta]ME_CONFIG_REQ
[sa_sta]ME_CHAN_CONFIG_REQ
[sa_sta]---ssid: , key:
msh />a:wifi-team , keyssid C0-3F-0E-C7-91-4C
security2cipher 2 3 24 8 security=6
cipher2security 2 3 24 8
hapd_intf_add_vif,type:2, s:0, id:0
wpa_dInit
wpa_supplicant_req_scan
Setting scan request: 0.100000 sec
MANUAL_SCAN_REQ
wpa_supplicant_scan
Cancelling scan request
wpa_driver_associate
scan_start_req_handler
sm_auth_send:1
sm_auth_handler
sm_assoc_rsp_handler
rc_init: station_id=0 format_mod=2 pre_type=0 short_gi=1 max_bw=0
rc_init: nss_max=0 mcs_max=7 r_idx_min=0 r_idx_max=3 no_samples=10
```

图20.4.2-4

20.5 注意事项

- 手机需要连接2.4G的路由器。

21 声波配网

21.1 声波配网简介

通过voice_tools工具生成16bit, 48kHz, 1个channel的wav/pcm格式的文件, BK7251芯片可以通过识别此类格式的文件来连接网络。

21.2 声波配网 Related API

声波配网相关接口参考samples\voice_config\include\voice_config.h, 相关接口如下:

函数	描述
voice_config_work()	打开设备
voice_config_stop()	用户提前终止声波配网
voice_config_version()	获取声波配网版本号

21.2.1 声波配网开始

```
int voice_config_work(void *device,
                      uint32_t sample_rate,
                      uint32_t timeout,
                      struct voice_config_result *result)
```

参数	描述
device	录音设备
sample_rate	采样率(16000)
timeout	超时时间
result	声波识别结果
返回	0:成功; 其他:失败

参数类型

voice_config_result:	
uint32_t ssid_len	网络id长度
uint32_t passwd_len	网络密码长度
uint32_t custom_len	用户自定义数据的长度
char ssid[32+1]	ssid数组
char passwd[63+1]	密码数组
char custom[16+1]	用户自定义的数据

21.2.2 用户提前终止声波配网

```
void voice_config_stop(void)
```

参数	描述
void	无
返回	无

21.2.3 获取版本号

```
const char *voice_config_version(void)
```

参数	描述
void	无
返回	版本号

21.3 声波配网示例代码

声波配网示例代码参考test\samples\voice_config\voice_config.c。打开宏定义：VOICE_CONFIG_TEST，开启声波配网测试。

```
/*
 * 程序清单： 这是一个声波配网使用例程，声波配网需要用工具生成一个声音文件，在输入命令之后
              让demo板来获取声音，等待demo板配网，配网成功之后会有一系列的打印信息。
 * 命令调用格式： voice_netconfig_start
 * 程序功能： 手机上播放声音（声音需要voice_tools生成），demo板通过识别手机播放的声音可以连
              上网络
 */
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <stdint.h>

#include <rtthread.h>
#include <rtdevice.h>
#include <rthw.h>
#include "samples_config.h"
#include "voice_config.h"

#ifdef XIAOYA_OS
#include "parm_cache.h"
```

```
#include "player_manager.h"
#endif

#ifdef VOICE_CONFIG_TEST

#define DEBUG_PRINTF      rt_kprintf("[voice] ");rt_kprintf
#define SAMPLE_RATE      (16000)

#define malloc    rt_malloc
#define realloc   rt_realloc
#define free      rt_free

#define codec_device_lock(...)
#define codec_device_unlock(...)

/***** voice config start *****/
static unsigned char voice_config_ssid[32 + 1] = {0};
static unsigned char voice_config_password[64 + 1] = {0};

void *voice_malloc(int size)
{
    return rt_malloc(size);
}

void voice_free(void *mem)
{
    rt_free(mem);
}

int voice_read(void *device, void *buffer, int size)
{
    struct rt_device *dev = (struct rt_device *)device;
    rt_size_t read_bytes = 0;

    while (read_bytes < size)
    {
        rt_size_t rb = rt_device_read(dev, 0, (void *)((char *)buffer + read_bytes), size - read_bytes);

        if (rb == 0)
```

```
        break;

        read_bytes += rb;
    }

    return read_bytes;
}

#include <finsh.h>
#include <msh.h>
static void station_connect(const char *ssid, const char *passwd)
{
    char argv[64];

    memset(argv, 0, sizeof(argv));
    sprintf(argv, "wifi %s join %s %s", "w0", ssid, passwd);
    msh_exec(argv, strlen(argv));
}

static rt_thread_t tid = RT_NULL;
static void cmd_voice_config_thread(void *parameter)
{
    rt_device_t device = 0;
    struct voice_config_result result={0};
    int res;
    rt_kprintf("cmd_voice_config_thread start!\n");
    DEBUG_PRINTF("voice config version: %s\r\n", voice_config_version());

    /* open audio device and set tx done call back */
    device = rt_device_find("mic");
    if (device == RT_NULL)
    {
        DEBUG_PRINTF("audio device not found!\r\n");
        goto _err;
    }

    codec_device_lock();
    if(device->flag & RT_DEVICE_FLAG_ACTIVATED)
    {
        rt_device_close(device);
```

```
}
res = rt_device_open(device, RT_DEVICE_OFLAG_RDWR);
/* set samplerate */
if (RT_EOK == res)
{
    int SamplesPerSec = SAMPLE_RATE;
    if (rt_device_control(device, CODEC_CMD_SAMPLERATE, &SamplesPerSec)
        != RT_EOK)
    {
        rt_kprintf("[record] audio device doesn't support this sample rate: %d\r\n",
                    SamplesPerSec);
        goto _err;
    }
}
else
{
    DEBUG_PRINTF("open audio device fail!\r\n");
    goto _err;
}

rt_device_write(device, 0, 0, 100); // start to record
DEBUG_PRINTF("voice_config_work----\r\n");
res = voice_config_work(device, SAMPLE_RATE, NETCONFIG_TIMEOUT, &result);
if(res == 0)
{
    #ifndef XIAOYA_OS
        xiaoya_player_tips(TIP_FIND_AP_INFO,0);
        /*not real save,just cache*/
        parm_set_wechat_openid_str((uint8_t *)result.custom);
        sta_cfg_t sta_cfg;
        memcpy(sta_cfg.ssid_str,result.ssid,strlen(result.ssid)+1);
        memcpy(sta_cfg.pwd_str,result.passwd,strlen(result.passwd)+1);
        parm_set_sta_cfg(&sta_cfg);
    #endif
    rt_kprintf("ssid len=%d, [%s]\n", result.ssid_len, result.ssid);
    rt_kprintf("passwd L=%d, [%s]\n", result.passwd_len, result.passwd);
    rt_kprintf("custom L=%d, [%s]\n", result.custom_len, result.custom);

    station_connect(result.ssid,result.passwd);
```

```
}
else
{
    rt_kprintf("voice_config res:%d\n", res);
}

_err:
    if (device)
    {
        rt_device_close(device);
        codec_device_unlock();
    }

    tid = RT_NULL;

    return;
}

int voice_netconfig_start()
{
    rt_kprintf("voice_config start!\n");
    if (tid)
    {
        rt_kprintf("voice config already init.\n");
        return -1;
    }

    tid = rt_thread_create("voice_config",
                           cmd_voice_config_thread,
                           RT_NULL,
                           1024 * 6,
                           20,
                           10);

    if (tid != RT_NULL)
    {
        rt_thread_startup(tid);
    }

    return 0;
}
```



```
}

void voice_netconfig_stop(void)
{
    if (tid != RT_NULL)
    {

        rt_kprintf("voice config cancel .\n");
        voice_config_stop();
        tid = NULL;
    }
}

#ifdef FINSH_USING_MSH
#include "finsh.h"

MSH_CMD_EXPORT(voice_netconfig_start, start voice config);
MSH_CMD_EXPORT(voice_netconfig_stop, stop voice config);

#endif /* FINSH_USING_MSH */

#endif
```

21.4 操作说明

21.4.1 打开配置

声波配网示例代码参考\test\samples\voice_config\voice_config.c。打开宏定义：VOICE_CONFIG_TEST，开启声波配网测试。

21.4.2 运行现象

- 使用**voice_tools**生成**.wav**声音文件

运行voice_tools.exe，在cmd输入命令：

voice_tools "tp link" "passwd" "openid" wifi.wav 生成wifi.wav文件。运行cmd命令如下：

```
D:\111DDD\tools\voice_tool>voice_tools " " " " "openid" wifi
4.wav
Shanghai Real Thread Electronic Technology Co.,Ltd.
voice config tools. V2.0.1
build Feb 15 2019 10:45:01

ssid[9]:
password[9]:
custom[6]: openid
raw data: data: 1A AC 77 69 66 69 2D 74 65 61 6D 00 73 74 6D 33 32 66 32 31 35
00 6F 70 65 6E 69 64
```

图21.4.2-1

• 输入命令:voice_config

用手机或者其他工具播放wifi.wav声音文件，demo板获取声音数据，连接生成.wav文件的网络。运行log如下：

```
voice_config
msh />
msh />[voice] voice config version: 2.0.0
adc-buf:00900cc8, adc-buf-len:5120, ch:1
set adc sample rate 16000
ssid len=9, [ ]
passwd L=9, [ ]
custom L=3, [ ]
[DRV_WLAN]drivers\wlan\drv_wlan.c L922 beken_wlan_control cmd: case
WIFI_INIT!
[Flash]ENV isn't initialize OK.
[Flash]ENV isn't initialize OK.
[Flash]ENV isn't initialize OK.
[Flash]ENV isn't initialize OK.
[Flash]ENV isn't initialize OK.
[wifi_connect]: read ap_info is empty
[wifi_connect]: normal connect
_wifi_easyjoin: ssid:wifi-team key:stm32f215
rl_sta_start
[sa_sta]MM_RESET_REQ
[sa_sta]ME_CONFIG_REQ
[sa_sta]ME_CHAN_CONFIG_REQ
[sa_sta]MM_START_REQ
hapd_intf_add_wif, type:2, s:0, id:0
wpa_dInit
wpa_suppllicant_req_scan
Setting scan request: 0.100000 sec
MANUAL_SCAN_REQ
wpa_suppllicant_scan
wpa_drv_scan
wpa_send_scan_req
scan_start_req_handler
wpa_driver_scan_cb
wpa_get_scan_rst:1
```

图21.4.2-2

网络连接成功log如下：

```
-----SM_CONNECT_IND_ok
wpa_driver_assoc_cb
Cancelling scan request
hapd_intf_add_key CCMP
add sta_mgmt_get_sta
sta:0, vif:0, key:0
sta_mgmt_add_key
add hw key idx:24
add TKIP
add is_broadcast_ether_addr
sta:255, vif:0, key:1
add hw key idx:1
ctrl_port_hdl:1
[wlan_connect]:start tick = 8176, connect done tick = 12325, total = 4149
[wlan_connect]:start tick = 8176, connect done tick = 12331, total = 4155
[WLAN_MGMT]wlan sta connected event callback
sta_ip_start

configuring interface wlan (with DHCP client)
dhcp_check_status_init_timer

new dtim period:2
IP UP: 19 . . . .
[ip_up]:start tick = 8176, ip_up tick = 15288, total = 7112
[Flash]ENV isn't initialize OK.
[Flash]ENV isn't initialize OK.
[Flash]ENV isn't initialize OK.
[Flash]ENV isn't initialize OK.
[Flash]ENV isn't initialize OK.
[Flash]ENV isn't initialize OK.
write new profile to flash 0x001FF000 72 byte!
[Flash]ENV isn't initialize OK.
[Flash]ENV isn't initialize OK.
[Flash]ENV isn't initialize OK.
[Flash]ENV isn't initialize OK.
[Flash]ENV isn't initialize OK.
[Flash]ENV isn't initialize OK.
[Flash]ENV isn't initialize OK.
*-[31:22m[E/NTP]: ERROR select the socket timeout(10s)-[0m
```

图21.4.2-3

22 Vad

22.1 Vad自动语音检测简介

Vad功能是声音边界检测，检测声音的开始和结束。当芯片中有音频数据该功能就会检测到数据存在，并且打印检测到声音。

22.2 Vad Related API

vad相关接口参考\beken378\func\vad.h，相关接口如下：

函数	描述
wb_vad_enter()	进入vad检测模式
wb_vad_get_frame_len()	获取帧的长度
wb_vad_entry()	vad入口函数
wb_vad_deinit()	关闭vad模块

22.2.1 进入vad检测模式

vad检测模式包括vad初始化，buffer长度设置。

```
int wb_vad_enter(void);
```

参数	描述
void	空
返回	0: 成功; 其他 : 错误

22.2.2 获取帧的长度

```
int wb_vad_get_frame_len(void);
```

参数	描述
void	空
返回	WB_FRAME_LEN : 帧的长度

22.2.3 vad入口函数

进入vad检测模式，函数如下：

```
int wb_vad_entry(char *buffer, int len);
```

参数	描述
buffer	测试buffer



len	测试buffer长度
返回	vad_flag

22.2.4 关闭vad

```
void wb_vad_deinit(void);
```

参数	描述
void	空
返回	空

22.3 Vad示例代码

```
/*
 * 程序清单： 这是一个vad使用例程
 * 命令调用格式： record_and_play 1
 * 程序功能： 例程通过录音和播放功能验证vad的准确性
 */
#include <rtthread.h>
#include <rtdevice.h>
#include <finsh.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "board.h"
#include "audio_device.h"

#define MICPHONE_TEST
#ifdef MICPHONE_TEST

#define TEST_BUFF_LEN 60*1024
#define READ_SIZE 1024

static uint8_t *test_buf;

void record_and_play(int argc, char *argv[])
{
    int mic_read_len = 0;
    int actual_len, i;
    int dac_wr_len = 0;
```

```
uint16_t *buffer = NULL;

int vad_on;

#if CONFIG_SOUND_MIXER
    mixer_pause();
#endif

vad_on = atoi(argv[1]);

test_buf = sdram_malloc(TEST_BUFF_LEN);
if(test_buf == NULL)
{
    rt_kprintf("===not enough memory===\r\n");
    return;
}

audio_device_init();                                /*初始化 sound mic设备*/

audio_device_mic_open();                             /*打开mic设备*/
audio_device_mic_set_channel(1);                     /*设置adc通道*/
audio_device_mic_set_rate(16000);                   /*设置adc采样率*/

if (vad_on)
{
    rt_kprintf("Vad is ON !!!!!!!\r\n"); /*进入vad检测*/
    wb_vad_enter();
}

while(1)
{
    if (vad_on)
        rt_thread_delay(5);
    else
        rt_thread_delay(20);

    int chunk_size = wb_vad_get_frame_len();//320
    char *val = NULL;

    if(mic_read_len > TEST_BUFF_LEN - READ_SIZE)
```

```
        break;

    if (!vad_on)
    {
        actual_len = audio_device_mic_read(test_buf+mic_read_len,READ_SIZE);
    }
    else
    {
        /*mic 采集声音数据*/
        actual_len = audio_device_mic_read(test_buf+mic_read_len,chunk_size);
        if(wb_vad_entry(test_buf+mic_read_len, actual_len))
        {
            rt_kprintf("Vad Detected !!!!!!!\r\n");           /*检测到声音*/
            break;
        }
    }

    mic_read_len += actual_len;
}

if (vad_on)
{
    wb_vad_deinit();           /*关闭vad检测*/
}

rt_kprintf("mic_read_len is %d\r\n", mic_read_len);
audio_device_mic_close();     /*关闭mic设备*/

audio_device_open();          /*打开dac设备*/
audio_device_set_rate(8000);  /*设置dac采样率*/

while(1)
{
    buffer = (uint16_t *)audio_device_get_buffer(RT_NULL);
    if(dac_wr_len >= mic_read_len)
    {
        audio_device_put_buffer(buffer);
        break;
    }
}
```

```
memcpy(buffer,test_buf+dac_wr_len,READ_SIZE);
dac_wr_len += READ_SIZE;

audio_device_write((uint8_t *)buffer, READ_SIZE); /*dac播放数据*/
}
audio_device_close(); /*关闭dac设备*/

if(test_buf)
    sdram_free(test_buf); /*释放ram内存*/

#if CONFIG_SOUND_MIXER
    mixer_replay();
#endif
}
MSH_CMD_EXPORT(record_and_play, record play);
#endif
```

22.4 操作说明

Vad示例代码参考\test\mic_record.c，具体使用方式如下：

22.4.1 打开配置

打开宏定义：MICPHONE_TEST，开启list player的功能测试，编译后下载到设备。

22.4.2 运行现象

上电后，在调试串口输入record_and_play，可听到识别的芯片中的声音，同时串口Log如下所示：

```
msh />record_and_play
adc-buf:00900cc8, adc-buf-len:5120, ch:1
audio_device_mic_opened
adc-buf:00900cc8, adc-buf-len:5120, ch:1
set adc channel 1
audio_device_mic_set_channel:1
set adc sample rate 16000
audio_device_mic_set_rate:16000
mic_read_len is 61440
audio_device_mic_closed
```




```
[icodec]:open sound device
```

```
audio_device_opened
```

```
===set fade in flag===
```

```
[icodec]:close sound device
```

```
audio_device_closed
```

```
msh />
```

23 AMR编码器

23.1 AMR编码器简介

AMR编码将接收到的语音信息编码成AMR格式的音频文件,其中编解码器所有的原文件被打包成库。

23.2 AMR编码器 Related API

AMR编码器APIs参考\components\codec\lib_amr_encode\amrnb_encode.h, 相关接口如下:

函数	描述
<code>amrnb_encoder_init()</code>	amr编码初始化
<code>amrnb_encoder_encode()</code>	amr编码
<code>amrnb_encoder_deinit()</code>	退出amr编码

23.2.1 AMR-NB编码器初始化

```
int32_t amrnb_encoder_init(void** amrnb, uint32_t dtx, void* pmalloc, void* pfree);
```

参数	描述
<code>amrnb</code>	AMR-NB编码器的指针
<code>dtx</code>	0: 连续传输数据 1: 不连续传输数据
<code>pmalloc</code>	malloc函数指针
<code>pfree</code>	free函数指针
返回	RT_EOK: 成功; 其他: 失败

23.2.2 AMR-NB编码

```
int32_t amrnb_encoder_encode(void* amrnb, uint32_t mode, const int16_t  
in[AMRNB_ENCODER_SAMPLES_PER_FRAME], uint8_t  
out[AMRNB_ENCODER_MAX_FRAME_SIZE])
```

参数	描述
<code>amrnb</code>	AMR-NB编码器的指针
<code>mode</code>	amr编码模式
<code>in</code>	输入的语音
<code>out</code>	输出的语音
返回	>0:read_byte:读取的字节数; 其他: 错误

23.2.3 释放AMR-NB编码

```
int32_t amrnb_encoder_deinit(void** amrnb);
```

参数	描述
amrnb	AMR-NB编码器的指针
返回	RT_EOK: 成功; 其他: 失败

23.3 AMR编码器示例代码

AMR编码器示例代码参考\test\record_tcp.c, 打开宏定义: RECORD_COM_TCP_TEST, 开启amr编码功能测试。

23.3.1 关键说明

• AMR编码器宏定义

定义AMR编码器每帧中数据的大小

```
#define AMRNB_ENCODER_SAMPLES_PER_FRAME (160)
```

定义AMR编码器最大帧的大小

```
#define AMRNB_ENCODER_MAX_FRAME_SIZE (32)
```

• AMR编码器枚举类型说明

AMR编码速率枚举类型:

```
enum Mode {  
    AMRNB_MODE_MR475 = 0, /* 4.75 kbps */  
    AMRNB_MODE_MR515,    /* 5.15 kbps */  
    AMRNB_MODE_MR59,     /* 5.90 kbps */  
    AMRNB_MODE_MR67,     /* 6.70 kbps */  
    AMRNB_MODE_MR74,     /* 7.40 kbps */  
    AMRNB_MODE_MR795,    /* 7.95 kbps */  
    AMRNB_MODE_MR102,    /* 10.2 kbps */  
    AMRNB_MODE_MR122,    /* 12.2 kbps */  
    AMRNB_MODE_MRDTX,    /* DTX */  
    AMRNB_MODE_N_MODES  /* Not Used */  
};
```

23.3.2 示例代码

程序清单: 这是一个音频编码器例程

* 命令调用格式: 配网成功之后, 使用网络串口调试助手接收网络端发过来的编码数据, 输入命令:

record_main start 编码模式 网络地址 网络端口号 生成对应格式的码流, 其中1代表amr编码模

式。后开始录音并且生成amr格式的数据流。

继续命令: record_main record_again

停止命令: record_main record_stop

* 程序功能: 例程通过调用命令将录制的音频信号转化成amr格式码流。

```
*/
#include <rtthread.h>
#include <rtdevice.h>
#include <finsh.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "board.h"
#include "audio_device.h"
#include "vad.h"
#include "record.h"
#include "opus.h"
#include "amrnb_encoder.h"
#include <sys/socket.h>
#include "netdb.h"

/*record thread received msg*/
#define MSG_RECORD_START      1
#define MSG_RECORD_CANCEL    2
#define USING_VAD
#define DURATION_PER_FRAME    20
#define NOTIFY_FRAME_COUNT    (200/DURATION_PER_FRAME) /*200ms*/
#define FRAME_COUNT_PER_SECOND (1000/DURATION_PER_FRAME)
#define AMR_MAGIC_NUMBER     "#!AMR\n"

#define MAX_DATA_BUF_SIZE (320 *60)

typedef struct tcp_net_worker
{
    char *url;
    int port;
    int sock;
}tcp_net_worker_t;

typedef enum
{

```

```
    REC_ENCODE_NONE = 0,
    REC_ENCODE_START,
    REC_ENCODE_GOING,
    REC_ENCODE_CANCEL,
}REC_ENCODE_STATE;

typedef enum
{
    ARM_ENCODE_MODE = 0,
    OPUS_ENCODE_MODE,
    NO_ENCODE_MODE,
}REC_ENCODE_MODE;

struct rec_encoder_manager
{
    rt_event_t rec_evt;
    rt_mailbox_t rec_mb;
    uint16_t encoded_len;
    REC_ENCODE_MODE encoder_mode;
    int sample_rate;
    rt_uint8_t data_buf[MAX_DATA_BUF_SIZE];
};

static tcp_net_worker_t tcpclient;
static struct rec_encoder_manager *rec_encoder = NULL;

static void set_start_event(void)
{
    rt_event_send(rec_encoder->rec_evt, EVENT_TCP_START);
}

static void set_end_event(void)
{
    rt_event_send(rec_encoder->rec_evt, EVENT_TCP_END);
}

static void free_rec_enc(void)
{
    if(rec_encoder != RT_NULL)
    {
```

```
        if(rec_encoder->rec_evt)
            rt_event_delete(rec_encoder->rec_evt);
        if(rec_encoder->rec_mb)
            rt_mb_delete(rec_encoder->rec_mb);
        rt_free(rec_encoder);
        rec_encoder = RT_NULL;
    }
}

/*main thread process record and amr-encode*/
static void rec_encoder_thread(void *parameter)
{

    enum AMRNB_MODE amr_enc_mode = AMRNB_MODE_MR122;
    void *amr = NULL;
    REC_ENCODE_STATE rec_state;
    OpusEncoder *opus_enc = RT_NULL;

    uint8_t vad_end_flag;
    int ret,i,read_bytes,enc_len;
    short *in_pcm_buf;
    rt_tick_t tmp_tick;
    rt_uint32_t mb_msg;
    uint16_t pcm_len_per_frame;

    rt_kprintf("record encoder start \r\n");
    rec_state = REC_ENCODE_NONE;
    rec_encoder->encoded_len = 0;
    rec_encoder->sample_rate = 8000;
    //just one channel data
    pcm_len_per_frame = (rec_encoder->sample_rate / FRAME_COUNT_PER_SECOND) * 2;
    in_pcm_buf = (short*)rt_malloc(pcm_len_per_frame);
    ASSERT(NULL != in_pcm_buf);

    rt_kprintf("record encoder start len=%d\r\n",pcm_len_per_frame);
    rt_thread_delay(100);

    while(1)
    {
```

```
ret = rt_mb_recv(rec_encoder->rec_mb,&mb_msg, RT_WAITING_NO);
if(RT_EOK == ret)
{
    rt_kprintf("---mb receive msg:%x---\r\n",mb_msg);
    if(REC_ENCODE_GOING == rec_state)
    {
        #ifdef USING_VAD
        wb_vad_deinit();
        #endif
        audio_device_mic_close();
        if(OPUS_ENCODE_MODE==rec_encoder->encoder_mode)
            opus_encoder_destroy(opus_enc);
        else if(ARM_ENCODE_MODE==rec_encoder->encoder_mode)
            amrnb_encoder_deinit(&amr);
    }

    if(MSG_RECORD_START == mb_msg )
    {
        rt_kprintf("----start---\r\n");
        rec_state = REC_ENCODE_START;
    }
    else if(MSG_RECORD_CANCEL == mb_msg )
    {
        if(REC_ENCODE_GOING == rec_state)
        {
            rt_kprintf("----cancel---\r\n");
            rec_state = REC_ENCODE_NONE;
        }
    }
}

switch(rec_state)
{
    case REC_ENCODE_NONE:
        rt_thread_delay(100);
        break;

    case REC_ENCODE_START:
        set_start_event();//nofity
        audio_device_mic_open();
}
```

```
audio_device_mic_set_channel(1);
audio_device_mic_set_rate(8000);

#ifdef USING_VAD
rt_kprintf("---wb_vad_enter---\r\n");
vad_end_flag = 0;
wb_vad_enter();//vad start
#endif

if(OPUS_ENCODE_MODE==rec_encoder->encoder_mode)
{

    int channels,application,complexity,errors;
    opus_int32 bitrate_bps;
    enc_len = opus_encoder_get_size(1);
    rt_kprintf("opus_encoder_get_size: 1 channel size: %d \n", enc_len);
    enc_len = opus_encoder_get_size(2);
    rt_kprintf("opus_encoder_get_size: 2 channel size: %d \n", enc_len);

    channels = 1;
    application = OPUS_APPLICATION_VOIP;
    complexity = 1; // 1 to 10
    opus_enc = opus_encoder_create(rec_encoder->sample_rate, channels,
application, &errors);

    if(errors != OPUS_OK)
    {
        rt_kprintf("[opus]:create opus encoder failed : %d! \n", errors);
    }
    rt_kprintf("----start opus encode--- \r\n");
    opus_encoder_set_complexity(opus_enc, complexity);
    opus_encoder_get_bitrate(opus_enc,bitrate_bps );
    rt_kprintf("[opus]:default bitrate %d\n", bitrate_bps);
    rec_encoder->encoded_len=0;
}
else if(ARM_ENCODE_MODE==rec_encoder->encoder_mode)
{
    ret = amrnb_encoder_init(&amr, 0, rt_malloc, rt_free);
    if(0 != ret)
    {
        rt_kprintf("[amr]:create amr encoder failed \n");
    }
}
```



```
    }
    rt_kprintf("---start amr encode--- \r\n");
    /* write amr head */
    memcpy(rec_encoder->data_buf, AMR_MAGIC_NUMBER,
strlen(AMR_MAGIC_NUMBER));
    rec_encoder->encoded_len = strlen(AMR_MAGIC_NUMBER);
}
else if(NO_ENCODE_MODE==rec_encoder->encoder_mode)
{
    rec_encoder->encoded_len=0;
}
rec_state = REC_ENCODE_GOING;
break;

case REC_ENCODE_GOING:
read_bytes = 0;
i = 0;
tmp_tick = rt_tick_get();

while(i<NOTIFY_FRAME_COUNT)
{

    if(rec_encoder->encoded_len >=MAX_DATA_BUF_SIZE)
    {
        rt_kprintf("+++++++record done+++++\r\n");
        rec_encoder->encoded_len=MAX_DATA_BUF_SIZE;
        break;
    }
    /* read data from sound device */
    read_bytes=audio_device_mic_read(in_pcm_buf, pcm_len_per_frame);
    rt_kprintf("read_bytes:%d\r\n",read_bytes);
    /*encode ....*/
    if(OPUS_ENCODE_MODE==rec_encoder->encoder_mode)
    {
        enc_len = opus_encode(opus_enc, in_pcm_buf, pcm_len_per_frame/2,
rec_encoder->data_buf+rec_encoder->encoded_len + 8, 120 - 8);
        if(enc_len>0)
        {
            /* write head */
            opus_uint32 enc_final_range;
```

```
        int_to_char_big_endian(enc_len,
rec_encoder->data_buf+rec_encoder->encoded_len);

        opus_encoder_get_final_range(opus_enc, enc_final_range);
        int_to_char_big_endian(enc_final_range,
rec_encoder->data_buf+rec_encoder->encoded_len+4);

        enc_len += 8;
        rec_encoder->encoded_len+=enc_len;
    }
    else
    {
        rt_kprintf("opus encode error!!\r\n");
    }
}
else if(ARM_ENCODE_MODE==rec_encoder->encoder_mode)
{
    enc_len =
amrnb_encoder_encode(amr,amr_enc_mode,in_pcm_buf,rec_encoder->data_buf +
rec_encoder->encoded_len);

    if(enc_len > 0)
    {
        rec_encoder->encoded_len += enc_len;
    }
    else
    {
        rt_kprintf("amr encode error!!\r\n");
    }
}
else if(NO_ENCODE_MODE==rec_encoder->encoder_mode)
{
    memcpy(rec_encoder->data_buf +
rec_encoder->encoded_len,in_pcm_buf,read_bytes);
    rec_encoder->encoded_len +=read_bytes;
}
#ifdef USING_VAD
if(wb_vad_entry((char*)in_pcm_buf, read_bytes))*vad process*/
{
    rt_kprintf("-----vad end-----\r\n");
    vad_end_flag = 1;
    break;
}
```

```
        }
        #endif
        i++;
    }
    rt_kprintf("--time:%d ms---\r\n",rt_tick_get()-tmp_tick);
    //set_data_event();//optional
    #ifdef USING_VAD
    if((rec_encoder->encoded_len >=MAX_DATA_BUF_SIZE)||((1 ==
vad_end_flag)){
        wb_vad_deinit();
    }
    #else
    if(rec_encoder->encoded_len >=MAX_DATA_BUF_SIZE){
    #endif
        rt_kprintf("--record end:len = %d---\r\n",rec_encoder->encoded_len);
        rt_thread_delay(5);
        set_end_event();//nofity
        audio_device_mic_close();
        if(OPUS_ENCODE_MODE==rec_encoder->encoder_mode)
            opus_encoder_destroy(opus_enc);
        else if(ARM_ENCODE_MODE==rec_encoder->encoder_mode)
            amrnb_encoder_deinit(&amr);
        rec_state = REC_ENCODE_NONE;
    }
    break;
default:
    rec_state = REC_ENCODE_NONE;
    break;
}
}
free_rec_enc();
}

rt_err_t get_record_event(rt_uint32_t *event,rt_int32_t timeout)
{
    return rt_event_rcv(rec_encoder->rec_evt,EVENT_TCP_ALL,\
        RT_EVENT_FLAG_OR|RT_EVENT_FLAG_CLEAR,timeout,event);
}

/*be called by talk&wechat proces*/
char *get_data_buf(void)
```

```
{
    return rec_encoder->data_buf;
}

/*be called by talk&wechat proces*/
int get_data_len(void)
{
    return rec_encoder->encoded_len;
}

static void net_transmit_thread_entry(void *parameter)
{
    int cmd;
    int ret,size;
    int sock;
    rt_uint32_t evt;
    char *buf=NULL;
    struct hostent *host;
    struct sockaddr_in server_addr;
    host = gethostbyname(tcpclient.url);
    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) == -1)
    {
        rt_kprintf("[tcp]:Socket error\n");
        return;
    }

    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(tcpclient.port);
    server_addr.sin_addr = *((struct in_addr *)host->h_addr);
    rt_memset(&(server_addr.sin_zero), 0, sizeof(server_addr.sin_zero));
    if (connect(sock, (struct sockaddr *)&server_addr, sizeof(struct sockaddr)) == -1)
    {
        rt_kprintf("[tcp]:Connect fail!\n");
        closesocket(sock);
        return ;
    }
    else
    {
        rt_kprintf("tcp connected success!\n");
    }
    while(1)
```

```
{
    ret = get_record_event(&evt, RT_WAITING_FOREVER);
    if(RT_EOK == ret)
    {
        if(evt & EVENT_TCP_START)
        {
            rt_kprintf("record start\r\n");
        }
        else if(evt & EVENT_TCP_END)
        {
            buf = get_data_buf();
            size = get_data_len();
            rt_kprintf("=====tcp send mic data =====\r\n");
            rt_kprintf("size:%d\r\n", size);
            send(sock, buf, size, 0);
        }
    }
}

/*be called by external thread*/
void record_start()
{
    if(NULL != rec_encoder->rec_mb)
        rt_mb_send(rec_encoder->rec_mb, MSG_RECORD_START);
}

/*be called by external thread*/
void record_cancel()
{
    if(NULL != rec_encoder->rec_mb)
        rt_mb_send(rec_encoder->rec_mb, MSG_RECORD_CANCEL);
}

static void record_help()
{
    rt_kprintf("eg: record_main start 0 192.168.1.100 8080 \r\n");
    rt_kprintf("eg: record_main record_again \r\n");
    rt_kprintf("eg: record_main record_stop \r\n");
}
```

```
    rt_kprintf("eg: record_main enc_mode_change 1 \r\n");
}

static void record_main(int argc,char **argv)
{
    rt_thread_t tid;
    if (strcmp(argv[1], "start") == 0)
    {
        if(NULL == rec_encoder)
        {
            if(argc<5)
            {
                rt_kprintf("argc error\r\n");
                record_help();
                return;
            }
            rec_encoder = rt_calloc(1, sizeof(struct rec_encoder_manager));
            if(NULL == rec_encoder)
            {
                rt_kprintf("rec_encoder_init error!!\r\n");
                return;
            }
            rec_encoder->rec_evt = rt_event_create("rec evt",RT_IPC_FLAG_FIFO);
            if(NULL == rec_encoder->rec_evt)
                goto exit;
            rec_encoder->rec_mb = rt_mb_create("rec mb",3,RT_IPC_FLAG_FIFO);
            if(NULL == rec_encoder->rec_mb)
                goto exit;

            rec_encoder->encoder_mode =
atoi(argv[2]);//ARM_ENCODE_MODE:0,OPUS_ENCODE_MODE:1,NO_ENCODE_MODE:2
            tcpclient.url = rt_strdup(argv[3]);    // eg:192.168.1.100
            tcpclient.port = atoi(argv[4]);

            /* create rec-encoder thread */
            tid = rt_thread_create("rec_enc",rec_encoder_thread,NULL,1024 * 32,20,10);
            if (tid)
                rt_thread_startup(tid);

            tid = rt_thread_create("net_send",net_transmit_thread_entry,RT_NULL,1024 *
```

```
8,21,10);

    if (tid)
        rt_thread_startup(tid);
    }
    record_start();
}

else if (strcmp(argv[1], "record_again") == 0)
{
    record_start();
}

else if (strcmp(argv[1], "enc_mode_change") == 0)
{
    if(argc<3)
    {
        rt_kprintf("argc error\r\n");
        record_help();
        return;
    }

    rec_encoder->encoder_mode =
    atoi(argv[2]); //ARM_ENCODE_MODE:0,OPUS_ENCODE_MODE:1,NO_ENCODE_MODE:2
    record_start();
}

else if (strcmp(argv[1], "record_stop") == 0)
{
    record_cancel();
}

else
{
    rt_kprintf("error argv!!!!\r\n");
    record_help();
}

return;

exit:
    rt_kprintf("error,exit\r\n");
    free_rec_enc();
}

//eg:
MSH_CMD_EXPORT(record_main,record_main);
```

23.4 操作说明

23.4.1 下载AMR Player工具

AMR Player工具: [下载地址](#)

23.4.2 网络调试助手设置

本示例需要借助PC端工具网络调试助手和AMR Player工具，其中AMR Player用来播放amr声音文件。调试助手设置如下图:



图23.4.2-1

23.4.3 打开配置

AMR编码器示例代码参考\test\record_tcp.c，打开宏定义：RECORD_COM_TCP_TEST，开启amr编码功能测试。

23.4.4 运行现象

输入命令：`record_main start 编码模式 网络地址 网络端口号`
开始录音并且网络调试助手接收编码的数据流，将数据流改为amr格式用AMR Player工具播放amr文件。

24 Opus编码器

24.1 Opus编码器简介

Opus编码将接收到的语音信息编码成opus格式的音频文件,其中编解码器所有的原文件被打包成库。

24.2 Opus编码器 Related API

opus编码器相关接口参考\components\codec\lib_opus\include\opus.h，相关接口如下：

函数	描述
<code>opus_encoder_create()</code>	创建opus编码
<code>opus_encoder_get_size()</code>	返回编码器所需内存的大小
<code>opus_encoder_set_complexity()</code>	修改编码器复杂度
<code>opus_encoder_get_bitrate()</code>	获取编码器的比特率
<code>opus_encoder_get_final_range()</code>	获取编码器最终状态
<code>opus_encode()</code>	opus编码
<code>opus_encoder_destroy()</code>	释放编码器对象

24.2.1 创建opus编码器

```
OpusEncoder *opus_encoder_create(opus_int32 Fs, int channels, int application, int *error);
```

参数	描述
Fs	输入信号的采样率（包括8k，16k）
channels	编码通道，只能为1或2
application	编码模式，由宏定义的3种编码模式
error	错误类型
返回	编码器对象的结构体

24.2.2 返回opus编码器所需内存的大小

```
int opus_encoder_get_size(int channels);
```

参数	描述
channels	通道必须为1或者2
返回	字节数

24.2.3 修改opus编码器的复杂度

```
opus_encoder_set_complexity(opus_enc, complexity);
```

参数	描述
opus_enc	opus编码器的结构体
complexity	编码器复杂度: 0-10;
返回	编码器对象的结构体

24.2.4 获取opus编码器的比特率

```
opus_encoder_get_bitrate(opus_enc, bitrate_bps);
```

参数	描述
opus_enc	opus编码器的结构体
bitrate_bps	编码器的比特率
返回	编码器对象的结构体

24.2.5 获取opus编码器的最终状态

```
opus_encoder_get_final_range(opus_enc, enc_final_range);
```

参数	描述
opus_enc	opus编码器的结构体
enc_final_range	编码器最终的熵
返回	编码器对象的结构体

24.2.6 opus编码

```
opus_int32 opus_encode (OpusEncoder *st,  
                        const opus_int16 *pcm,  
                        int frame_size,  
                        unsigned char *data,  
                        opus_int32 max_data_bytes);
```

参数	描述
st	编码器对象
pcm	输入信号
size	输入音频信号每个声道的采样数量
data	输出编码结果

max_data_bytes	为输出编码结果分配内存
返回	编码长度：成功； 负数：失败

24.2.7 释放opus编码器对象

```
void opus_encoder_destroy(OpusEncoder *st);
```

参数	描述
st	编码器对象
返回	空

24.3 Opus编码器示例代码

Opus编码器示例代码参考test\record_cp.c，打开宏定义：RECORD_COM_TCP_TEST，开启音频编码功能测试。

24.3.1 关键说明

• Opus编码器宏定义

三种opus编码模式的宏定义如下：

1.在给定的比特率条件下为声音信号提供最高质量，一般情况此种模式。

#define	OPUS_APPLICATION_VOIP	2048
---------	-----------------------	------

2.对大多数非语音信号在给定的比特率条件下提供最高的质量。

#define	OPUS_APPLICATION_AUDIO	2049
---------	------------------------	------

3.配置低延迟模式将为减少延迟禁用语音优化模式。

#define	OPUS_APPLICATION_RESTRICTED_LOWDELAY	2051
---------	--------------------------------------	------

24.3.2 示例代码

```
/* 程序清单： 这是一个音频编码器例程
* 命令调用格式： 配网成功之后，使用网络串口调试助手接收网络端发过来的编码数据，输入命令：
record_main start 编码模式 网络地址 网络端口号 生成对应格式的码流，其中0代表opus编码。
修改文件名称变为opus文件，使用工具转换成pcm文件，通过cool edit pro 播放生成的pcm格式文件。
继续命令： record_main record_again
停止命令： record_main record_stop
* 程序功能： 例程通过调用命令将音频信号转化成opus格式。
*/
#include <rtthread.h>
```

```
#include <rtdevice.h>
#include <finsh.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "board.h"
#include "audio_device.h"
#include "vad.h"
#include "record.h"
#include "opus.h"
#include "amrnb_encoder.h"
#include <sys/socket.h>
#include "netdb.h"

/*record thread received msg*/
#define MSG_RECORD_START      1
#define MSG_RECORD_CANCEL    2
#define USING_VAD
#define DURATION_PER_FRAME    20
#define NOTIFY_FRAME_COUNT   (200/DURATION_PER_FRAME) /*200ms*/
#define FRAME_COUNT_PER_SECOND (1000/DURATION_PER_FRAME)
#define AMR_MAGIC_NUMBER     "#!AMR\n"

#define MAX_DATA_BUF_SIZE (320 *60)

typedef struct tcp_net_worker
{
    char *url;
    int port;
    int sock;
}tcp_net_worker_t;

typedef enum
{
    REC_ENCODE_NONE = 0,
    REC_ENCODE_START,
    REC_ENCODE_GOING,
    REC_ENCODE_CANCEL,
}REC_ENCODE_STATE;
```

```
typedef enum
{
    ARM_ENCODE_MODE = 0,
    OPUS_ENCODE_MODE,
    NO_ENCODE_MODE,
}REC_ENCODE_MODE;

struct rec_encoder_manager
{
    rt_event_t rec_evt;
    rt_mailbox_t rec_mb;
    uint16_t encoded_len;
    REC_ENCODE_MODE encoder_mode;
    int sample_rate;
    rt_uint8_t data_buf[MAX_DATA_BUF_SIZE];
};

static tcp_net_worker_t tcpclient;
static struct rec_encoder_manager *rec_encoder = NULL;

static void set_start_event(void)
{
    rt_event_send(rec_encoder->rec_evt, EVENT_TCP_START);
}

static void set_end_event(void)
{
    rt_event_send(rec_encoder->rec_evt, EVENT_TCP_END);
}

static void free_rec_enc(void)
{
    if(rec_encoder != RT_NULL)
    {
        if(rec_encoder->rec_evt)
            rt_event_delete(rec_encoder->rec_evt);
        if(rec_encoder->rec_mb)
            rt_mb_delete(rec_encoder->rec_mb);
        rt_free(rec_encoder);
        rec_encoder = RT_NULL;
    }
}
```

```
}  
}  
  
/*main thread process record and amr-encode*/  
static void rec_encoder_thread(void *parameter)  
{  
  
    enum AMRNB_MODE amr_enc_mode = AMRNB_MODE_MR122;  
    void *amr = NULL;  
    REC_ENCODE_STATE rec_state;  
    OpusEncoder *opus_enc = RT_NULL;  
  
    uint8_t vad_end_flag;  
    int ret,i,read_bytes,enc_len;  
    short *in_pcm_buf;  
    rt_tick_t tmp_tick;  
    rt_uint32_t mb_msg;  
    uint16_t pcm_len_per_frame;  
  
    rt_kprintf("record encoder start \r\n");  
    rec_state = REC_ENCODE_NONE;  
    rec_encoder->encoded_len = 0;  
    rec_encoder->sample_rate = 8000;  
    //just one channel data  
    pcm_len_per_frame = (rec_encoder->sample_rate / FRAME_COUNT_PER_SECOND) * 2;  
    in_pcm_buf = (short*)rt_malloc(pcm_len_per_frame);  
    ASSERT(NULL != in_pcm_buf);  
  
    rt_kprintf("record encoder start len=%d\r\n",pcm_len_per_frame);  
    rt_thread_delay(100);  
  
    while(1)  
    {  
        ret = rt_mb_rcv(rec_encoder->rec_mb,&mb_msg, RT_WAITING_NO);  
        if(RT_EOK == ret)  
        {  
            rt_kprintf("---mb receive msg:%x---\r\n",mb_msg);  
            if(REC_ENCODE_GOING == rec_state)  
            {  

```

```
#ifdef USING_VAD
wb_vad_deinit();
#endif
audio_device_mic_close();
if(OPUS_ENCODE_MODE==rec_encoder->encoder_mode)
    opus_encoder_destroy(opus_enc);
else if(ARM_ENCODE_MODE==rec_encoder->encoder_mode)
    amrnb_encoder_deinit(&amr);
}

if(MSG_RECORD_START == mb_msg )
{
    rt_kprintf("----start---\r\n");
    rec_state = REC_ENCODE_START;
}
else if(MSG_RECORD_CANCEL == mb_msg )
{
    if(REC_ENCODE_GOING == rec_state)
    {
        rt_kprintf("----cancel---\r\n");
        rec_state = REC_ENCODE_NONE;
    }
}

switch(rec_state)
{
    case REC_ENCODE_NONE:
        rt_thread_delay(100);
        break;

    case REC_ENCODE_START:
        set_start_event();//nofity
        audio_device_mic_open();
        audio_device_mic_set_channel(1);
        audio_device_mic_set_rate(8000);

#ifdef USING_VAD
        rt_kprintf("---wb_vad_enter---\r\n");
        vad_end_flag = 0;
```

```
wb_vad_enter();//vad start
#endif

if(OPUS_ENCODE_MODE==rec_encoder->encoder_mode)
{

    int channels,application,complexity,errors;
    opus_int32 bitrate_bps;
    enc_len = opus_encoder_get_size(1);
    rt_kprintf("opus_encoder_get_size: 1 channel size: %d \n", enc_len);
    enc_len = opus_encoder_get_size(2);
    rt_kprintf("opus_encoder_get_size: 2 channel size: %d \n", enc_len);

    channels = 1;
    application = OPUS_APPLICATION_VOIP;
    complexity = 1; // 1 to 10
    opus_enc = opus_encoder_create(rec_encoder->sample_rate, channels,
application, &errors);

    if(errors != OPUS_OK)
    {
        rt_kprintf("[opus]:create opus encoder failed : %d! \n", errors);
    }
    rt_kprintf("---start opus encode--- \r\n");
    opus_encoder_set_complexity(opus_enc, complexity);
    opus_encoder_get_bitrate(opus_enc,bitrate_bps );
    rt_kprintf("[opus]:default bitrate %d\n", bitrate_bps);
    rec_encoder->encoded_len=0;
}
else if(ARM_ENCODE_MODE==rec_encoder->encoder_mode)
{
    ret = amrnb_encoder_init(&amr, 0, rt_malloc, rt_free);
    if(0 != ret)
    {
        rt_kprintf("[amr]:create amr encoder failed \n");
    }
    rt_kprintf("---start amr encode--- \r\n");
    /* write amr head */
    memcpy(rec_encoder->data_buf, AMR_MAGIC_NUMBER,
strlen(AMR_MAGIC_NUMBER));
    rec_encoder->encoded_len = strlen(AMR_MAGIC_NUMBER);
```



```
    }
    else if(NO_ENCODE_MODE==rec_encoder->encoder_mode)
    {
        rec_encoder->encoded_len=0;
    }
    rec_state = REC_ENCODE_GOING;
    break;

case REC_ENCODE_GOING:
    read_bytes = 0;
    i = 0;
    tmp_tick = rt_tick_get();

    while(i<NOTIFY_FRAME_COUNT)
    {

        if(rec_encoder->encoded_len >=MAX_DATA_BUF_SIZE)
        {
            rt_kprintf("++++++record done+++++\r\n");
            rec_encoder->encoded_len=MAX_DATA_BUF_SIZE;
            break;
        }
        /* read data from sound device */
        read_bytes=audio_device_mic_read(in_pcm_buf, pcm_len_per_frame);
        rt_kprintf("read_bytes:%d\r\n",read_bytes);
        /*encode ....*/
        if(OPUS_ENCODE_MODE==rec_encoder->encoder_mode)
        {
            enc_len = opus_encode(opus_enc, in_pcm_buf, pcm_len_per_frame/2,
rec_encoder->data_buf+rec_encoder->encoded_len + 8, 120 - 8);
            if(enc_len>0)
            {
                /* write head */
                opus_uint32 enc_final_range;
                int_to_char_big_endian(enc_len,
rec_encoder->data_buf+rec_encoder->encoded_len);
                opus_encoder_get_final_range(opus_enc, enc_final_range);
                int_to_char_big_endian(enc_final_range,
rec_encoder->data_buf+rec_encoder->encoded_len+4);
                enc_len += 8;
```

```
        rec_encoder->encoded_len+=enc_len;
    }
    else
    {
        rt_kprintf("opus encode error!!\r\n");
    }
}
else if(ARM_ENCODE_MODE==rec_encoder->encoder_mode)
{
    enc_len =
amrnb_encoder_encode(amr,amr_enc_mode,in_pcm_buf,rec_encoder->data_buf +
rec_encoder->encoded_len);
    if(enc_len > 0)
    {
        rec_encoder->encoded_len += enc_len;
    }
    else
    {
        rt_kprintf("amr encode error!!\r\n");
    }
}
else if(NO_ENCODE_MODE==rec_encoder->encoder_mode)
{
    memcpy(rec_encoder->data_buf +
rec_encoder->encoded_len,in_pcm_buf,read_bytes);
    rec_encoder->encoded_len +=read_bytes;
}
#ifdef USING_VAD
    if(wb_vad_entry((char*)in_pcm_buf, read_bytes))*vad process*/
    {
        rt_kprintf("-----vad end-----\r\n");
        vad_end_flag = 1;
        break;
    }
#endif
    i++;
}
rt_kprintf("--time:%d ms---\r\n",rt_tick_get()-tmp_tick);
//set_data_event();//optional
```

```
#ifndef USING_VAD
    if((rec_encoder->encoded_len >=MAX_DATA_BUF_SIZE)||((1 ==
vad_end_flag)){
        wb_vad_deinit();
    }
    #else
    if(rec_encoder->encoded_len >=MAX_DATA_BUF_SIZE){
    #endif
        rt_kprintf("--record end:len = %d---\r\n",rec_encoder->encoded_len);
        rt_thread_delay(5);
        set_end_event();//nofity
        audio_device_mic_close();
        if(OPUS_ENCODE_MODE==rec_encoder->encoder_mode)
            opus_encoder_destroy(opus_enc);
        else if(ARM_ENCODE_MODE==rec_encoder->encoder_mode)
            amrnb_encoder_deinit(&amr);
        rec_state = REC_ENCODE_NONE;
    }
    break;
default:
    rec_state = REC_ENCODE_NONE;
    break;
}
}
free_rec_enc();
}

rt_err_t get_record_event(rt_uint32_t *event,rt_int32_t timeout)
{
    return rt_event_rcv(rec_encoder->rec_evt,EVENT_TCP_ALL,\
        RT_EVENT_FLAG_OR|RT_EVENT_FLAG_CLEAR,timeout,event);
}

/*be called by talk&wechat proces*/
char *get_data_buf(void)
{
    return rec_encoder->data_buf;
}

/*be called by talk&wechat proces*/
int get_data_len(void)
{

```

```
        return rec_encoder->encoded_len;
    }

static void net_transmit_thread_entry(void *parameter)
{
    int cmd;
    int ret,size;
    int sock;
    rt_uint32_t evt;
    char *buf=NULL;
    struct hostent *host;
    struct sockaddr_in server_addr;
    host = gethostbyname(tcpclient.url);
    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) == -1)
    {
        rt_kprintf("[tcp]:Socket error\n");;
        return;
    }

    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(tcpclient.port);
    server_addr.sin_addr = *((struct in_addr *)host->h_addr);
    rt_memset(&(server_addr.sin_zero), 0, sizeof(server_addr.sin_zero));
    if (connect(sock, (struct sockaddr *)&server_addr, sizeof(struct sockaddr)) == -1)
    {
        rt_kprintf("[tcp]:Connect fail!\n");
        closesocket(sock);
        return ;
    }
    else
    {
        rt_kprintf("tcp connected success\r\n");
    }
    while(1)
    {
        ret = get_record_event(&evt,RT_WAITING_FOREVER);
        if(RT_EOK == ret)
        {
            if(evt & EVENT_TCP_START)
            {
```

```
        rt_kprintf("record start\r\n");
    }
    else if(evt & EVENT_TCP_END)
    {
        buf = get_data_buf();
        size = get_data_len();
        rt_kprintf("=====tcp send mic data =====\r\n");
        rt_kprintf("size:%d\r\n",size);
        send(sock,buf,size,0);
    }
}

/*be called by external thread*/
void record_start()
{
    if(NULL!=rec_encoder->rec_mb)
        rt_mb_send(rec_encoder->rec_mb, MSG_RECORD_START);
}

/*be called by external thread*/
void record_cancel()
{
    if(NULL!=rec_encoder->rec_mb)
        rt_mb_send(rec_encoder->rec_mb, MSG_RECORD_CANCEL);
}

static void record_help()
{
    rt_kprintf("eg: record_main start 0 192.168.1.100 8080 \r\n");
    rt_kprintf("eg: record_main record_again \r\n");
    rt_kprintf("eg: record_main record_stop \r\n");
    rt_kprintf("eg: record_main enc_mode_change 1 \r\n");
}

static void record_main(int argc,char **argv)
{
    rt_thread_t tid;
```

```
if (strcmp(argv[1], "start") == 0)
{
    if(NULL == rec_encoder)
    {
        if(argc<5)
        {
            rt_kprintf("argc error!\n\n");
            record_help();
            return;
        }
        rec_encoder = rt_calloc(1, sizeof(struct rec_encoder_manager));
        if(NULL == rec_encoder)
        {
            rt_kprintf("rec_encoder_init error!!\n\n");
            return;
        }
        rec_encoder->rec_evt = rt_event_create("rec evt",RT_IPC_FLAG_FIFO);
        if(NULL == rec_encoder->rec_evt)
            goto exit;
        rec_encoder->rec_mb = rt_mb_create("rec mb",3,RT_IPC_FLAG_FIFO);
        if(NULL == rec_encoder->rec_mb)
            goto exit;

        rec_encoder->encoder_mode =
atoi(argv[2]);//ARM_ENCODE_MODE:0,OPUS_ENCODE_MODE:1,NO_ENCODE_MODE:2
        tcpclient.url = rt_strdup(argv[3]);        // eg:192.168.1.100
        tcpclient.port = atoi(argv[4]);

        /* create rec-encoder thread */
        tid = rt_thread_create("rec_enc",rec_encoder_thread,NULL,1024 * 32,20,10);
        if (tid)
            rt_thread_startup(tid);

        tid = rt_thread_create("net_send",net_transmit_thread_entry,RT_NULL,1024 *
8,21,10);
        if (tid)
            rt_thread_startup(tid);
    }
    record_start();
}
```

```
else if (strcmp(argv[1], "record_again") == 0)
{
    record_start();
}
else if (strcmp(argv[1], "enc_mode_change") == 0)
{
    if(argc<3)
    {
        rt_kprintf("argc error\r\n");
        record_help();
        return;
    }
    rec_encoder->encoder_mode =
    atoi(argv[2]);//ARM_ENCODE_MODE:0,OPUS_ENCODE_MODE:1,NO_ENCODE_MODE:2
    record_start();
}
else if (strcmp(argv[1], "record_stop") == 0)
{
    record_cancel();
}
else
{
    rt_kprintf("error argv!!!!\r\n");
    record_help();
}
return;
exit:
    rt_kprintf("error,exit\r\n");
    free_rec_enc();
}

//eg:
MSH_CMD_EXPORT(record_main,record_main);
```

24.4 操作说明

24.4.1 下载Cool Edit Pro工具

Cool Edit Pror工具: [下载地址](#)

24.4.2 网络调试助手设置

本示例需要借助PC端工具网络调试助手和Cool Edit Pro工具，其中Cool Edit Pro用来播放opus声音文件。调试助手设置如下图



图24.4.2-1

24.4.3 打开配置

Opus编码器示例代码参考test\record_tcp.c，打开宏定义：RECORD_COM_TCP_TEST，开启opus编码功能测试。

24.4.4 运行现象

配网成功之后，使用网络串口调试助手接收网络端发过来的编码数据，借助opus.exe工具(工具位于tool目录下)，输入命令：record_main start 编码模式 网络地址 网络端口号 生成opus格式的码流，修改文件名称变为opus文件，使用opus工具转换成pcm文件，通过cool edit pro 播放生成的pcm格式文件。

25 EasyFlash

25.1 EasyFlash简介

EasyFlash是一款开源的轻量级嵌入式Flash存储器库，能快速保存产品参数，支持写平衡和掉电保护，降低了开发者对产品参数的处理难度，也保证了产品在后期升级时拥有更好的扩展性。

25.2 EasyFlash Related API

EasyFlash相关接口参考\packages\EasyFlash\inc\easyflash.h，相关接口如下：

函数	描述
<code>easyflash_init()</code>	easyflash初始化
<code>ef_get_env()</code>	获得easyflash环境变量
<code>ef_set_env()</code>	写数据到easyflash中
<code>ef_save_env()</code>	保存数据到flash

25.2.1 easyflash初始化

```
EfErrCode easyflash_init(void);
```

参数	描述
<code>void</code>	空
返回	0: 成功; 其他: 失败

25.2.2 获得easyflash环境变量

```
char *ef_get_env(const char *key);
```

参数	描述
<code>key</code>	环境变量名字
返回	value:变量地址

25.2.3 将数据写入到环境变量中

```
EfErrCode ef_set_env(const char *key, const char *value);
```

参数	描述
<code>key</code>	环境变量名字



value	要写入的值
返回	0: 成功; 其他: 失败

25.2.4 保存数据到flash

```
EfErrCode ef_save_env(void);
```

参数	描述
void	空
返回	0: 成功; 其他: 失败

25.3 EasyFlash示例代码

25.3.1 关键说明

• EasyFlash宏定义

#define	PKG_USING_EASYFLASH		使用EasyFlash必须开启
#define	EF_START_ADDR	0x1FD000	EasyFlash起始地址为0x1FD000
#define	ENV_USER_SETTING_SIZE	2 * 1024	EasyFlash用户大小

25.3.2 示例代码

```
#include "rtthread.h"
#include <dfs.h>
#include <dfs_fs.h>
#include "player.h"
#include "include.h"
#include "driver_pub.h"
#include "func_pub.h"
#include "app.h"
#include "ate_app.h"
#include "shell.h"
#include "flash.h"
#include <finsh.h>
#include "easyflash.h"
#include "test_config.h"

#ifdef EASY_FLASH_TEST

static void easy_flash_set(char *key, char *value)
```

```
{
    EfErrCode result = EF_NO_ERR;
    easyflash_init();          /*初始化 */
    result = ef_set_env(key, value);      /*将要写入的数据存放到 easy flash 环境变量 */
    if(result != EF_NO_ERR)
    {
        rt_kprintf("easy_flash set error\r\n");
        return;
    }
    result = ef_save_env();      /*保存数据 */
    if(result != EF_NO_ERR)
    {
        rt_kprintf("easy_flash save error\r\n");
        return;
    }
    rt_kprintf("---Flash Write over \r\n");
}

static void easy_flash_get(char *key, char *value) /*读取easy flash 写入的数据*/
{
    easyflash_init();

    value = ef_get_env(key);      /*获取easy flash存入的数据*/
    if( value )
    {
        rt_kprintf("%s\r\n",value);
    }
    else
    {
        rt_kprintf("easy_flash get error\r\n");
    }
    return ;
}

static void easy_flash_erase(char *key) /*读取easy flash 写入的数据*/
{
    EfErrCode result = EF_NO_ERR;
    char value = 0;
    easyflash_init();          /*初始化 */
    result = ef_set_env(key, &value);      /*将要写入的数据存放到 easy flash 环境变量 */
}
```

```
if(result != EF_NO_ERR)
{
    rt_kprintf("easy_flash set error\r\n");
    return;
}
result = ef_save_env();
if(result != EF_NO_ERR)
{
    rt_kprintf("easy_flash erase error\r\n");
}
else
{
    rt_kprintf("easy_flash erase success\r\n");
}
return;}

static int easy_flash(uint8_t argc, char **argv)
{
    char *key = NULL;
    char *value = NULL;

    if (strcmp(argv[1], "set") == 0)
    {
        os_printf("easyflash set command\r\n");
        if (argc == 4)
        {
            key = argv[2];
            value = argv[3];
        }
        else
        {
            os_printf("parameter invalid\r\n");
            return -1;
        }
        easy_flash_set(key, value);
        return 0;
    }
    else if (strcmp(argv[1], "get") == 0)
    {
        os_printf("easyflash get command\r\n");
        if (argc == 3)
        {
            {
```

```
        key = argv[2];
        easy_flash_get(key, value);
    }
    else
    {
        os_printf("parameter invalid\r\n");
        return -1;
    }
    return 0;
}
else if (strcmp(argv[1], "erase") == 0)
{
    os_printf("easyflash erase command\r\n");
    if (argc == 3)
    {
        key = argv[2];
        easy_flash_erase(key);
    }
    else
    {
        os_printf("parameter invalid\r\n");
        return -1;
    }
    return 0;
}
}

MSH_CMD_EXPORT(easy_flash, easy_flash_command: easy_flash <set/get/erase> <key> [value]);

#endif
```

25.4 操作说明

EasyFlash示例代码参考\test\easyflash_test.c，使能后支持参数的读、写、擦除功能。

25.4.1 打开配置

打开宏定义：EASY_FLASH_TEST，编译后重新下载到设备。

25.4.2 运行现象

- 写入

调试串口输入**easy_flash set test 111111111111**，其中key为“test”，value为“111111111111”，设备log如下：

```
msh />easy_flash set test 111111111111  
easyflash set command  
[Flash]EasyFlash V3.0.4 already initialize.  
[Flash]Erased ENV OK.  
[Flash]Saved ENV OK.  
---Flash Write over
```

- 读取

调试串口输入**easy_flash get test**，读取key为“test”的数据，设备log如下：

```
msh />easy_flash get test  
easyflash get command  
[Flash]EasyFlash V3.0.4 already initialize.  
111111111111
```

- 擦除

调试串口输入**easy_flash erase test**，擦除key为“test”的所有数据，设备log如下：

```
msh />easy_flash erase test  
easyflash erase command  
[Flash]EasyFlash V3.0.4 already initialize.  
easy_flash erase success
```

调试串口输入**easy_flash get test**，再次读取key为“test”的数据，以此来验证擦除操作是否成功，擦除操作成功，重新读取报错，设备log如下：

```
msh />easy_flash get test  
easyflash get command  
[Flash]EasyFlash V3.0.4 already initialize.  
easy_flash get error
```

26 Voice Changer

26.1 Voice Changer简介

Voice changer支持变声功能，能将声音变换成其他的声音特性。

26.2 Voice Changer Related API

Voice changer相关接口参考\components\voice_changer\app_vocie_changer.h，相关接口如下：

函数	描述
voice_changer_initial()	变声功能初始化
voice_changer_exit()	退出变声模式
voice_changer_start()	开始变声
voice_changer_stop()	停止变声
voice_changer_set_change_flag()	设置变声功能标志
voice_changer_get_need_mic_data()	获取麦克风数据
voice_changer_set_cost_data()	设置消耗的数据长度
voice_changer_data_handle()	处理声音数据

26.2.1 voice changer初始化

```
VC_ERR voice_changer_initial(uint32_t freq);
```

参数	描述
freg	频率
返回	0: 成功 其他: 失败

26.2.2 退出voice changer

```
void voice_changer_exit(void);
```

参数	描述
void	空
返回	无

26.2.3 开始voice changer

```
void voice_changer_start(void);
```

参数	描述
void	空
返回	无

26.2.4 停止voice changer

```
void voice_changer_stop(void);
```

参数	描述
void	空
返回	无

26.2.5 设置voice changer变声功能标志

```
void voice_changer_set_change_flag(void);
```

参数	描述
void	空
返回	无

26.2.6 voice changer获取mic数据

```
int voice_changer_get_need_mic_data(void);
```

参数	描述
void	空
返回	剩下数据长度

26.2.7 设置消耗数据的长度

```
int voice_changer_set_cost_data(int cost_len);
```

参数	描述
cost_len	消耗的数据长度
返回	剩下数据长度

26.2.8 处理数据

```
int voice_changer_data_handle(uint8_t *mic_in, int mic_len, uint8_t **vc_out);
```

参数	描述
----	----



mic_in	mic接收的数据
mic_len	mic接收的数据长度
vc_out	变声功能处理后的数据
返回	0: 成功 其他: 失败

26.3 Voice Changer示例代码

26.3.1 关键说明

• Voice Changer宏定义

#define	CONFIG_VOICE_CHANGER	使用voice changer必须开启
---------	----------------------	---------------------

• Voice Changer枚举类型

```
typedef enum {
    VC_STOP,      //停止
    VC_FIRST,     //第一个
    VC_START,     //开始
} VC_STA;
```

26.3.2 示例代码

```
/*
 * 程序清单： 这是一个voice changer用法例程
 * 命令调用格式： 输入命令： voice_changer_sample launch/shutoff/next
 * 程序功能： 将采集到的声音做变声处理。
 */
#include <rtthread.h>
#include "vc_config.h"

#define VOICE_CHANGER_SOFT_TIMER_HANDLER      1
#define VOICE_CHANGER_THREADT_TASK_HANDLER   2

#define VOICE_CHANGER_HANDLER
VOICE_CHANGER_THREADT_TASK_HANDLER

#define VOICE_CHANGER_MIC_CFG                1
#define VOICE_CHANGER_MIC_INIT_CFG           1
```

```
#define VOICE_CHANGER_DEFAULT_OUT_AUD      1
#define VOICE_CHANGER_AUD_INIT_CFG        1
#define VOICE_CHANGER_AUD_SINGLE_CH      1

#ifndef min
#define min(x, y)          (((x) < (y)) ? (x) : (y))
#endif

#if CONFIG_VOICE_CHANGER
#include "app_voice_changer.h"
#include "rtos_pub.h"
#include "audio_device.h"
#include "string.h"
#include "stdio.h"
#include "stdlib.h"

#define VC_BUFF_MAX_LEN      (256 * 4 * sizeof(unsigned int))
#define VC_HANDLER_INTERVAL_MS      5

beken_thread_t voice_changer_handler = NULL;
beken_timer_t vc_timer;

static void *vctimer = NULL;
static char *vcbuff = NULL;
static int g_running_flag;

#if VOICE_PCM_VC_AUD_OUTPUT_TEST
#define PCM_LENGTH          35254
extern const unsigned char acnumber_pcm[];          ///<35254
static unsigned int pc_offset = 0;
#endif

static int voice_changer_read_pcm(char*outbuf,int len)          /*读取mic数据并且存放到buffer*/
{
    int out_len = 0;
    #if VOICE_CHANGER_MIC_CFG
        out_len = audio_device_mic_read(outbuf,len);
    #endif

    #if VOICE_PCM_VC_AUD_OUTPUT_TEST
```

```
out_len = min(len,(PCM_LENGTH - pc_offset));
memcpy(outbuf,acnumber_pcm+pc_offset,out_len);

pc_offset += out_len;
if(pc_offset >= PCM_LENGTH)
{
    pc_offset = 0;
    rt_kprintf("restart\r\n");
}
#endif
return out_len;
}

static int voice_changer_write_pcm(char*outbuf,int len)          /*变声数据传送到到pcm*/
{
    int input_len = 0;

    #if VOICE_CHANGER_DEFAULT_OUT_AUD
        int bufsz;
        uint16_t* aud_buf = (uint16_t *)audio_device_get_buffer(&bufsz);
        if((bufsz == 0) || (aud_buf == NULL))
        {
            if(aud_buf)
            {
                audio_device_put_buffer(aud_buf);
            }
            rt_kprintf("vc err L%d\r\n",__LINE__);
            return input_len;
        }

        input_len = min((bufsz>>1),len);
        if(len == 0)
        {
            goto exit;
        }
        #if VOICE_CHANGER_AUD_SINGLE_CH
            int16_t *src,*dst;
            int i;
            src = outbuf;
            dst = aud_buf;
```

```
        for(i=0;i<(len/2);i++)
        {
            dst[2 * i] = src[i];
            dst[2 * i + 1] = src[i];
        }

        audio_device_write((uint8_t *)aud_buf, input_len*2);
    #else
        memcpy(aud_buf,outbuf,input_len);
        audio_device_write((uint8_t *)aud_buf, input_len);
    #endif
#endif
    return input_len;
exit:
    if(aud_buf)
    {
        audio_device_put_buffer(aud_buf);
    }
    rt_kprintf("vc L%d err\r\n",__LINE__);
    return 0;
}

static int voice_changer_shutoff(void)                                /*关闭变声功能*/
{
    g_running_flag = 0;
    if (vctimer != RT_NULL)
    {
        #if VOICE_CHANGER_HANDLER == VOICE_CHANGER_SOFT_TIMER_HANDLER
            rt_timer_stop((rt_timer_t)vctimer);
        #elif VOICE_CHANGER_HANDLER == VOICE_CHANGER_THREADT_TASK_HANDLER
            bk_rtos_delete_thread(vc_handler);
        #endif
    }

    return 0;}

static int voice_changer_launch(unsigned int freq)                  /*开启变声功能：加长声音*/
{
    if(vcbuff == NULL)
    {
```

```
        vcbuff = (char*)rt_malloc(VC_BUFF_MAX_LEN);
    }
    if(vcbuff == NULL)
    {
        rt_kprintf("vcbuff == null\r\n");
        return -1;
    }
#if (VOICE_CHANGER_MIC_CFG && VOICE_CHANGER_MIC_INIT_CFG)
    audio_device_init();

    audio_device_mic_open();
    audio_device_mic_set_channel(1);
    audio_device_mic_set_rate(freq);
#endif

#if VOICE_CHANGER_DEFAULT_OUT_AUD && VOICE_CHANGER_AUD_INIT_CFG
    audio_device_init();

    audio_device_open();
    audio_device_set_rate(freq);
    audio_device_set_volume(100);
#endif

    g_running_flag = 1;
    voice_changer_initial(freq);
    if (vctimer != RT_NULL)
    {
        #if VOICE_CHANGER_HANDLER == VOICE_CHANGER_SOFT_TIMER_HANDLER
            rt_timer_start((rt_timer_t)vctimer);
            voice_changer_start();
        #elif VOICE_CHANGER_HANDLER == VOICE_CHANGER_THREADT_TASK_HANDLER
            rt_thread_startup((rt_thread_t)vctimer);
        #endif
        rt_kprintf("vc start\r\n");
    }

    return 0;
}

static int voice_changer_handler(void)                                /*对采集的声音数据处理*/
{
```

```
unsigned char* vc_out;
int vc_out_len;
int len;

if(vcbuff == NULL)
{
    rt_kprintf("vcbuff err\r\n");
    return -1;
}

len = voice_changer_get_need_mic_data();
if(len > 0) {
    len = (len > (VC_BUFF_MAX_LEN/4))?(VC_BUFF_MAX_LEN/4) : len;
}
else if(len < 0)
{
    return -1;
}
else if(len == 0)
{
    return 0;
}

len = voice_changer_read_pcm(vcbuff,len);
if(len <= 0)
{
    rt_kprintf("origin pcm empty\r\n");
    return 0;
}

vc_out_len = voice_changer_data_handle((uint8*)vcbuff, len, &vc_out);
if(vc_out_len == 0)
{
    // no enough data for vc, so vc return 0, no need do sm_playing
    return 0;
}
else if(vc_out_len > 0)
{
    #if 1
    len = voice_changer_write_pcm((char*)vc_out,vc_out_len);
    #endif
}
```

```
#else
    voice_changer_write_pcm(vcbuff, len);
    len = vc_out_len;
#endif
    if(len > 0)
    {
        voice_changer_set_cost_data(len);
    }
}
return 0;
}

static int app_voice_changer_init(void) /* 变声功能初始化 */
{
    #if VOICE_CHANGER_HANDLER == VOICE_CHANGER_SOFT_TIMER_HANDLER
        if(vctimer == NULL)
        {
            vctimer = (void*)rt_timer_create("vc",
                                              voice_changer_timer_handler,
                                              NULL,
                                              VC_HANDLER_INTERVAL_MS,
                                              RT_TIMER_FLAG_PERIODIC |
RT_TIMER_FLAG_SOFT_TIMER);
        }
    #elif VOICE_CHANGER_HANDLER == VOICE_CHANGER_THREADT_TASK_HANDLER
        if(vctimer == NULL)
        {
            vctimer = (void*)rt_thread_create("vc",
                                              voice_changer_task_handler,
                                              NULL,
                                              4*1024,
                                              15,
                                              20);

            rt_kprintf("vctimer = %p\r\n", vctimer);
        }
    #endif
    return 0;
}

INIT_APP_EXPORT(app_voice_changer_init);
static int voice_changer_sample(int argc, char *argv[])
{

```

```
rt_err_t ret = RT_EOK;
unsigned int freq = 16000;

if(argc == 2)
{
    if(strcmp(argv[1],"launch") == 0)
    {
        rt_kprintf("voice changer freq = %d\r\n",freq);
        voice_changer_launch(freq);
        app_voice_changer_init();
    }
    else if(strcmp(argv[1],"shutoff") == 0)
    {
        rt_kprintf("voice changer shutoff\r\n");
        voice_changer_shutoff();
    }
    else if(strcmp(argv[1],"next") == 0)
    {
        rt_kprintf("voice changer set next\r\n");
        voice_changer_set_change_flag();
    }
}
else if(argc == 3)
{
    if(strcmp(argv[1],"launch") == 0)
    {
        freq = atoi(argv[2]);
        rt_kprintf("voice changer freq = %d\r\n",freq);
        voice_changer_launch(freq);
    }
}
return ret;
}

MSH_CMD_EXPORT(voice_changer_sample,vc sample);
#endif
```


26.4 操作说明

Voice changer示例代码参考\components\voice_changer\voice_changer_task.c。输入命令：voice_changer_sample launch/shutoff/next。

26.4.1 运行现象

输入命令：voice_changer_sample launch，对mic发声，可以明显发现自己的声音被拉长；输入命令：voice_changer_sample shutoff，对mic发声，可以明显发现自己的声音被拉短；输入命令：voice_config_stop 停止变声功能。

27 图像传输

27.1 图像传输简介

- a) 有高速spi-slave接口，速度高达50Mbps，可以外接其他MCU摄像头；
- b) 支持DCMI标准摄像头接口，PCLK高达24M。支持如PAS6329/6375、OV_7670、GC0328C/0308C等摄像头。
- c) 有硬件Jpeg压缩模块，目前支持最大分辨率600*800；
- d) 图像传输结构框图如下所示：

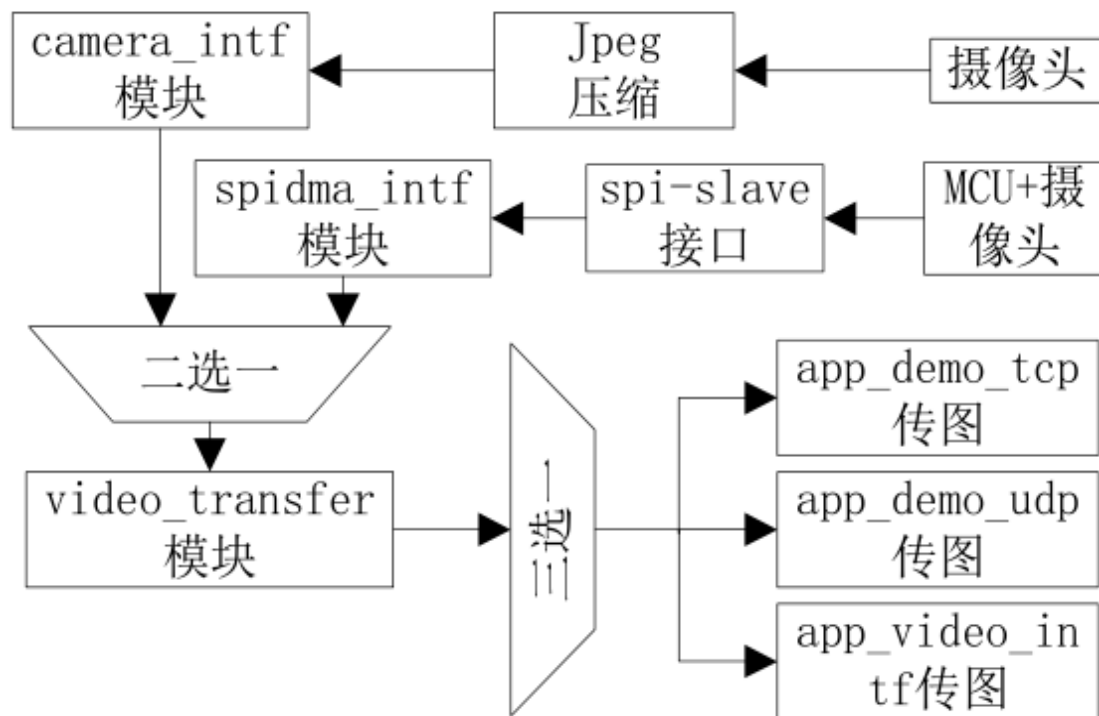


图27.1-1

如上图，图像的输入，可以选择bk7251自带的jpeg+摄像头模块，也可以通过spi-slave接口，外接MCU+摄像头模块。两种方式只能选择其中的一种。如下图，通过sys_config.h里的宏进行选择。

```
#define CFG_USE_SPIDMA          0
#define CFG_USE_CAMERA_INTF    1
```

27.2 图像传输 Related API

图像传输相关接口参考**beken378\func\video_transfer\video_transfer.h**。

函数	描述
video_transfer_init()	打开video_transfer模块
video_transfer_deinit()	关闭video_transfer模块

<code>video_transfer_set_video_param()</code>	使用DCMI接口时，设置摄像头的参数
<code>video_buffer_open()</code>	打开获取jpeg帧数据功能
<code>video_buffer_close()</code>	关闭获取jpeg帧数据功能
<code>video_buffer_read_frame()</code>	获取一帧jpeg数据，可能会挂起，直到整张jpeg收集完，并且该jpeg长度不超过目标buf的长度，才返回

27.2.1 打开video_transfer

```
int video_transfer_init(TVIDEO_SETUP_DESC_PTR setup_cfg);
```

参数	描述
<code>setup_cfg</code>	TVIDEO_SETUP_DESC_PTR类型的结构体
返回	kNoErr: 成功；其他：失败

参数类型

TVIDEO_SETUP_DESC_PTR :

<code>UINT32 send_type</code>	一般为TVIDEO_SND_TYPE枚举类型，模块里会根据send_type决定每个图像数据包的大小。
<code>send_func</code>	图像数据的发送函数，模块转发图像数据包时，通过send_func发送。
<code>start_cb</code>	模块打开spi或camera_intf后，回调此函数，用于指示传图开始。
<code>end_cb</code>	模块关闭spi或camera_intf前，回调此函数，用于指示传图结束。
<code>pkt_header_size</code>	若要在图像数据包里加入“头信息”，pkt_header_size用于指示“头信息”的大小，注意pkt_header_size的值必须是4的整数倍。如果不使用“头信息”，设成0即可。
<code>add_pkt_header</code>	添加“头信息”的回调函数，该函数会在每收到一个图像数据包时，回调，用户需实现“头信息”的具体内容。如果不使用“头信息”，设成NULL即可

27.2.2 关闭video_transfer

```
int video_transfer_deinit(void);
```

参数	描述
<code>void</code>	无
返回	kNoErr: 成功；其他：失败

27.2.3 设置摄像头的参数

```
UINT32 video_transfer_set_video_param(UINT32 ppi, UINT32 fps);
```

参数	描述
ppi	分辨率（pixer per inch），见PPI_TYPE的定义。
fps	帧率（frame per second），见FPS_TYPE的定义
返回	0：成功； 1：失败

27.2.4 打开获取jpeg帧的功能

```
int video_buffer_open (void);
```

参数	描述
void	无
返回	0：成功； 1：失败

27.2.5 关闭获取jpeg帧的功能

```
int video_buffer_close (void);
```

参数	描述
void	无
返回	0：成功； 1：失败

27.2.6 获取jpeg帧的数据

```
UINT32 video_buffer_read_frame(UINT8 *buf, UINT32 buf_len);
```

参数	描述
buf	存放jpeg数据的内存首地址
buf_len	存放jpeg数据的内存长度
返回	获取的jpeg帧的长度

27.3 图像传输的示例代码

27.3.1 关键说明

• 图像传输的宏定义

#define	CFG_USE_CAMERA_INTF	摄像头+jpeg 图传的开关宏
#define	CFG_USE_SPIDMA	High-spi-slave 图传时spidma模块的开关宏
#define	CFG_USE_HSLAVE_SPI	High-spi-slave 图传时spi接口的开关宏
#define	CFG_USE_APP_DEMO_VIDEO_TRANSFER	图传demo开关宏

• 图像传输枚举类型说明

```
typedef enum
{
    TVIDEO_SND_UDP,          /*通过UDP上传*/
    TVIDEO_SND_TCP,          /*通过TCP上传*/
    TVIDEO_SND_INTF,         /*通过其他接口上传*/
} TVIDEO_SND_TYPE;          /*通过其他接口上传*/

typedef enum {
    QVGA_320_240    = 0,
    VGA_640_480,
    PPI_MAX
} PPI_TYPE;                /*分辨率的枚举*/

typedef enum {
    TYPE_5FPS        = 0,
    TYPE_10FPS,
    TYPE_20FPS,
    FPS_MAX
} FPS_TYPE;                /*帧率的枚举*/
```

27.3.2 示例代码

1.不使用“头信息”的示例

```
/*发送函数，什么也没有做，直接返回*/
int app_video_intf_send_packet (UINT8 *data, UINT32 len)
{
    //os_printf("voide send:%p, %p\r\n", data, len);
    return len;
}

void app_video_intf_open (void)
{
    os_printf("voide open\r\n");
    /*spi接口方式 或 camera_intf 二选一*/
    #if (CFG_USE_SPIDMA || CFG_USE_CAMERA_INTF)
    TVIDEO_SETUP_DESC_ST setup;
    /* TVIDEO_SND_INTF 指示这个传送方式为 send intf*/
    setup.send_type = TVIDEO_SND_INTF;
    setup.send_func = app_video_intf_send_packet;
    /*不需要指示 图传开始或结束 */
```

```
    setup.start_cb = NULL;
    setup.end_cb = NULL;
    /*不使用 头信息*/
    setup.pkt_header_size = 0;
    setup.add_pkt_header = NULL;
    video_transfer_init(&setup);
    #endif
}

void app_video_intf_close (void)
{
    os_printf("voide close\r\n");
    #if (CFG_USE_SPIDMA || CFG_USE_CAMERA_INTF)
        video_transfer_deinit();
    #endif
}
```

2.使用“头信息”的示例

```
/*自定义 头信息 */
typedef struct tvideo_hdr_st
{
    UINT8 id;
    UINT8 is_eof;
    UINT8 pkt_cnt;
    UINT8 size;
}HDR_ST, *HDR_PTR;
/*头信息 回调函数。这个每个数据包的前4个字节都会加入 HDR_ST的头信息*/
void app_demo_add_pkt_header(TV_HDR_PARAM_PTR param)
{
    HDR_PTR elem_tvhdr = (HDR_PTR)param->ptk_ptr;
    elem_tvhdr->id = (UINT8)param->frame_id;
    elem_tvhdr->is_eof = param->is_eof;
    elem_tvhdr->pkt_cnt = param->frame_len;
    elem_tvhdr->size = 0;
}
/*发送函数，使用udp方式发送，返回发送成功的字节数 */
int app_demo_udp_send_packet (UINT8 *data, UINT32 len)
{
    int send_byte = 0;
```

```
if(!app_demo_udp_remote_connected)
    return 0;

send_byte = sendto(app_demo_udp_img_fd, data, len, MSG_DONTWAIT|MSG_MORE,
    (struct sockaddr *)app_demo_remote, sizeof(struct sockaddr_in));

if (send_byte < 0) {
    /* err */
    //APP_DEMO_UDP_PRT("send return fd:%d\r\n", send_byte);
    send_byte = 0;
}

return send_byte;
}

/*指示开始传图 */
static void app_demo_udp_app_connected(void)
{
    app_demo_softap_send_msg(DMSG_APP_CONNECTED);
}

/*指示停止传图 */
static void app_demo_udp_app_disconnected(void)
{
    app_demo_softap_send_msg(DMSG_APP_DISCONNECTED);
}

void app_video_intf_open (void)
{
    TVIDEO_SETUP_DESC_ST setup;
    setup.send_type = TVIDEO_SND_UDP;
    setup.send_func = app_demo_udp_send_packet;
    setup.start_cb = app_demo_udp_app_connected;
    setup.end_cb = app_demo_udp_app_disconnected;
    setup.pkt_header_size = sizeof(HDR_ST);
    setup.add_pkt_header = app_demo_add_pkt_header;
    video_transfer_init(&setup);
}

void app_video_intf_close (void)
{
    os_printf("voide close\r\n");
    #if (CFG_USE_SPIDMA || CFG_USE_CAMERA_INTF)
    video_transfer_deinit();
    #endif
}
```

3. 获取一帧jpeg图像以及设置摄像头参数的示例

```
/*发送串口命令 */
/*vbuf open : 打开获取一帧jpeg图像的功能 */
/*vbuf close : 关闭获取一帧jpeg图像的功能 */
/*vbuf read len_xxx: len_xxx 是读取buf的长度, 读取一帧jpeg图像, 并打印jpeg数据 */
/*vbuf setp ppi_xxx pfs_xxx : 分辨率ppi_xxx 的取值0、1, 帧率pfs_xxx的取值 0、1、2 */

void vbuf(int argc, char** argv)
{
    if(strcmp(argv[1], "open") == 0) {
        video_buffer_open();
    }
    else if(strcmp(argv[1], "read") == 0) {
        uint8_t *mybuf, i;
        uint32_t my_len;
        my_len = atoi(argv[2]);
        mybuf = os_malloc(my_len);
        if(mybuf == NULL)
        {
            rt_kprintf("vbuf test no buff\r\n");
            return;
        }
        my_len = video_buffer_read_frame(mybuf, my_len);
        rt_kprintf("frame_len: %d\r\n", my_len);
        if(1) {
            for(int i=0; i<my_len; i++)
            {
                rt_kprintf("%02x,", mybuf[i]);
                if((i+1)%32 == 0)
                    rt_kprintf("\r\n");
            }
        }
        os_free(mybuf);
    }
    else if(strcmp(argv[1], "close") == 0)
    {
        video_buffer_close();
    }
    else if(strcmp(argv[1], "setp") == 0)
    {

```



```

uint32_t ppi, pfs;
ppi = atoi(argv[2]);
pfs = atoi(argv[3]);
video_transfer_set_video_param(ppi, pfs);
}
else{
    rt_kprintf("vbuf open/read len/close/setp ppi pfs\r\n");
}
}
}
MSH_CMD_EXPORT(vbuf, vbuf);

```

27.4 操作说明

图像传输示例代码参考**beken378\app\app_demo**文件夹下，流程如下图：

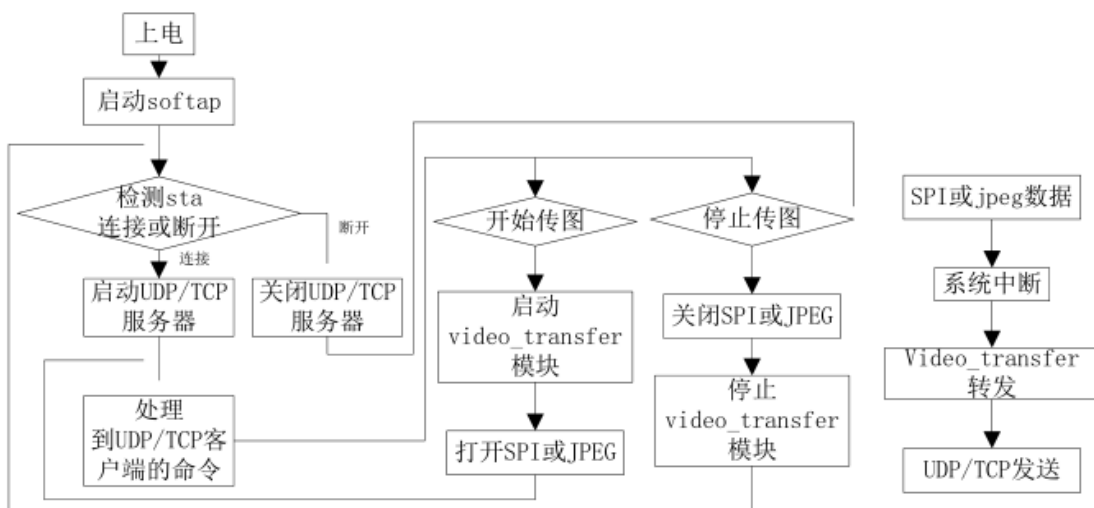


图27.4-1

如上图，上电后，会启动**softap**（启动参数可配置），然后监听**station**设备来连接（最多连接一个**station**）。

若检测到**station**连接成功，就会启动**UDP**或**TCP**服务器，等待**station**端发起**UDP**或**TCP**连接（**UDP**是无连接的，这里用**UDP**自定义的命令）。处理相关**station**发过来的数据。

若为打开图传，则打开**video_transfer**模块，**video_transfer**模块里会打开**spi**或**jpeg**。从此开始，**spi**接口或**jpeg**里的图像数据会通过系统中断的方式触发**video_transfer**的转发函数，最终通过**UDP**或**TCP**方式发送。

若为关闭图传，则关闭**video_transfer**模块，该模块里会关闭**spi**接口或**jpeg**模块，图像数据不再接收了。

若检测到**station**断线，会先关闭**UDP**或**TCP**服务，并关闭**video_transfer**模块。

27.4.1 下载PC调试工具

调试工具在SDK根目录tool\beken_wifi_camera文件夹中。

27.4.2 启动softap

调试串口输入video_demo, 开启softap, PC机(带wifi功能的笔记本), 找到BK_WIFI_000000的ssid, softap发现station连接成功后, 会启动UDP和TCP服务传输图像, 设备log下:

```
hapd_intf_sta_add:1, vif:0
rc_init: station_id=0 format_mod=0 pre_type=0 short_gi=0 max_bw=0
rc_init: nss_max=0 mcs_max=255 r_idx_min=0 r_idx_max=11 no_samples=10
sta_idx:0, pm_state:0
wpa_hostapd_no_password_connected
RW_EVT_AP_CONNECTED
ap_index:0
sta_id:0
send connected msg
app_demo_udp_init
app_demo_udp_main entry
app_demo_tcp_init
app_demo_tcp_main entry
wifi connected!
```

27.4.3 UDP传输测试

运行BK_Wifi_Camera.exe, 设置locate_ip和remote_ip后, 勾选By UDP选项框, 点击Play/Stop按钮, 会实时显示摄像头采集的图像。

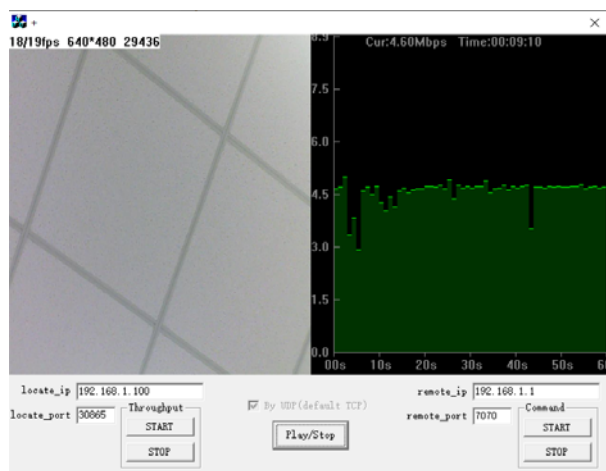


图27.4.3-1

27.4.4 TCP传输测试

运行BK_Wifi_Camera.exe, 设置locate_ip和remote_ip后, 不勾选ByUDP选项框, 点击Play/Stop按钮, 会实时显示摄像头采集的图像。

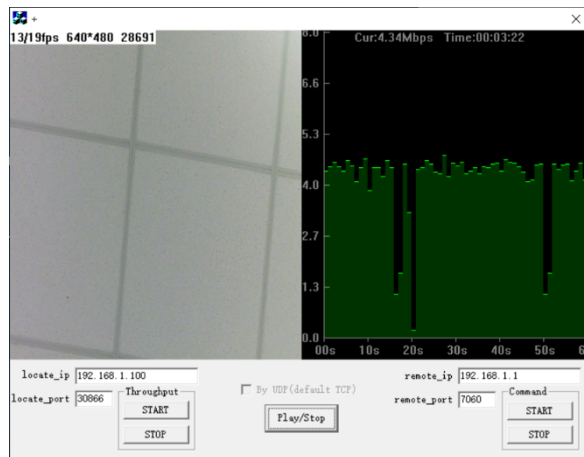


图27.4.4-1

27.4.5 web camera 浏览器实时显示视频

例程位于../test/web_camera.c,test_config.h开启宏定义

#define WEB_CAMERA_TEST /*开启web camera测试

步骤一: BK725X 与电脑端需要在同一局域网内, 方法一, BK725X设备端开启AP, 电脑连接设备AP,方法二, BK725X与电脑连接同一个路由器。

步骤二: BK725X 开启web camera ,串口命令web_jpeg_stream start。

步骤三: 电脑端打开摄像头, 输入设备端ip,即可查看摄像头视频。

```
[32:22m [I/FAL] app download beken_onchip_crc 0x00010000 0x00120000 [0m
[32:22m [I/FAL] beken_onchip 0x00143000 0x000a2600 [0m
[32:22m [I/FAL] RT-Thread Flash Abstraction Layer (VO.4.0) initialize success. [0m
tc_entity_init
ROMFS File System initialized!
msh />Enter normal mode...

app_init finished
[32m[I/player] RT-Thread lightweight player v1.2.7_Release [build Mar 3 2020][0m
set_volume 65-65
what voltage 3043
wifi ap ap test 12345678
[DRV] WLAN/drivers/wlan/drv_wlan.c L975 beken_wlan_control cmd: case WIFI_INIT!
_wifi_softap ssid:test key:12345678
rl_ap_request_enter
enter_timer-Astop
enter_timer-Astart
msh />
msh />rl_enter_handler
_ap_request_enter
rl_ap_start
Soft_AP_start
rl_init_csa_status
[saap]MM_RESET_REQ
[saap]ME_CONFIG_REQ
rw_msg_send_me_config_req ps_on is 1
[saap]ME_CHAN_CONFIG_REQ
[saap]MM_START_REQ
[csa]csa_in_progress[0:0]-clear
mm_add_if_req_handler 0
hapd_intf_add_vif,type:3, s:0, id:0
apm start with vif:0
set_beacon_int_set:100 TU
set_active_param 0
[msg]APM_STOP_CFM
update_ongoing_l_bcn_update
vif_idx:0, ch_idx:0, bcn_idx:2
```

图27.4.5-1 开启AP

28 Qspi Dcache模式

28.1 Qspi Dcache简介

BK7251的qspi dcache模式是将芯片外接的psram芯片地址映射到芯片dcache地址中，其中dcache基地址为0x03000000，通过这个基地址可以对psram进行正常的读写操作。

28.2 Qspi Dcache Related API

qspi dcache相关接口参考beken378\func\user_driver\BkDriverQspi.h，相关接口如下：

函数	描述
bk_qspi_dcache_initialize()	qspi dcache初始化
bk_qspi_start()	启动qspi功能
bk_qspi_stop ()	停止qspi功能

28.2.1 初始化qspi为dcache模式

```
OSStatus bk_qspi_dcache_initialize(qspi_dcache_drv_desc *qspi_config);
```

参数	描述
qspi_config	qspi的参数配置
返回	0: 成功; -1: 错误

参数类型

qspi_dcache_drv_desc	
mode	qspi模式
clk_set	时钟源选择分频系数设置
wr_command	写入数据命令
rd_command	读取数据命令
wr_dummy_size	写数据大小
rd_dummy_size	读数据大小

28.2.2 启动qspi功能

```
OSStatus bk_qspi_start(void);
```

参数	描述
void	空
返回	0: 成功; -1: 错误

28.2.3 停止qspi功能

```
OSStatus bk_qspi_stop(void);
```

参数	描述
void	空
返回	0: 成功; -1: 错误

28.3 Qspi Dcache示例代码

```
/*
 * 程序清单： 这是一个简单qspi dcache模式的使用例程，打开宏定义QSPI_TEST，开启测功能。
 * 命令调用格式： qspi_test
 * 程序功能： 通过qspi模块向psram写入数据并读取数据，最后比较写入和读取的数据是否有差别。
 */
#include "error.h"
#include "include.h"
#include <rthw.h>
#include <rtthread.h>
#include <rtdevice.h>
#include <stdint.h>
#include <stdlib.h>
#include <finsh.h>
#include <rtdef.h>
#include "include.h"
#include <stdio.h>
#include <string.h>
#include "typedef.h"
#include "arm_arch.h"
#include "qspi_pub.h"
#include "BkDriverQspi.h"
#include "test_config.h"

#ifdef QSPI_TEST

#define QSPI_TEST_LENGTH      ( 0x4 * 16 )

static UINT8 DataOffset;

static void qspi_psram_dcache_test(int argc, char *argv[])
```

```
{  
    UINT32 i,ret;  
    UINT32 SetLineMode;  
    qspi_dcache_drv_desc qspi_cfg;  
  
    UINT32*   p_WRData1;  
    UINT32*   p_WRData2;  
    UINT32*   p_WRData3;  
    UINT32*   p_WRData4;  
    UINT32*   p_WRData5;  
  
    UINT32*   p_RDData1;  
    UINT32*   p_RDData2;  
    UINT32*   p_RDData3;  
    UINT32*   p_RDData4;  
    UINT32*   p_RDData5;  
  
    p_WRData1 = rt_malloc(QSPI_TEST_LENGTH * sizeof(p_WRData1[0]));  
    if(p_WRData1 == RT_NULL)  
    {  
        rt_kprintf("p_WRData1 malloc failed\r\n");  
    }  
  
    p_WRData2 = rt_malloc(QSPI_TEST_LENGTH * sizeof(p_WRData2[0]));  
    if(p_WRData2 == RT_NULL)  
    {  
        rt_kprintf("p_WRData2 malloc failed\r\n");  
    }  
  
    p_WRData3 = rt_malloc(QSPI_TEST_LENGTH * sizeof(p_WRData3[0]));  
    if(p_WRData3 == RT_NULL)  
    {  
        rt_kprintf("p_WRData3 malloc failed\r\n");  
    }  
  
    p_WRData4 = rt_malloc(QSPI_TEST_LENGTH * sizeof(p_WRData4[0]));  
    if(p_WRData4 == RT_NULL)  
    {  
        rt_kprintf("p_WRData4 malloc failed\r\n");  
    }  
}
```

```
p_WRData5 = rt_malloc(QSPI_TEST_LENGTH * sizeof(p_WRData5[0]));
if(p_WRData5 == RT_NULL)
{
    rt_kprintf("p_WRData5 malloc failed\r\n");
}

p_RDData1 = rt_malloc(QSPI_TEST_LENGTH * sizeof(p_RDData1[0]));
if(p_RDData1 == RT_NULL)
{
    rt_kprintf("p_RDData1 malloc failed\r\n");
}

p_RDData2 = rt_malloc(QSPI_TEST_LENGTH * sizeof(p_RDData2[0]));
if(p_RDData2 == RT_NULL)
{
    rt_kprintf("p_RDData2 malloc failed\r\n");
}

p_RDData3 = rt_malloc(QSPI_TEST_LENGTH * sizeof(p_RDData3[0]));
if(p_RDData3 == RT_NULL)
{
    rt_kprintf("p_RDData3 malloc failed\r\n");
}

p_RDData4 = rt_malloc(QSPI_TEST_LENGTH * sizeof(p_RDData4[0]));
if(p_RDData4 == RT_NULL)
{
    rt_kprintf("p_RDData4 malloc failed\r\n");
}

p_RDData5 = rt_malloc(QSPI_TEST_LENGTH * sizeof(p_RDData5[0]));
if(p_RDData5 == RT_NULL)
{
    rt_kprintf("p_RDData5 malloc failed\r\n");
}

for(i=0; i<QSPI_TEST_LENGTH; i++)
{
```



```
p_WRData1[i]= ((i+1)<<24) | ((i+1)<<16) | ((i+1)<<8) | ((i+1)<<0) |0x70707070;
p_WRData2[i]= ((i+1)<<24) | ((i+1)<<16) | ((i+1)<<8) | ((i+1)<<0) |0x80808080;
p_WRData3[i]= ((i+1)<<24) | ((i+1)<<16) | ((i+1)<<8) | ((i+1)<<0) |0x90909090;
p_WRData4[i]= ((i+1)<<24) | ((i+1)<<16) | ((i+1)<<8) | ((i+1)<<0) |0xe0e0e0e0;
p_WRData5[i]= ((i+1)<<24) | ((i+1)<<16) | ((i+1)<<8) | ((i+1)<<0) |0xf0f0f0f0;
}

if(argc == 2)
{
    rt_kprintf("[qspi_test]:test_qspi_dcache_write_read_data\r\n");

    SetLineMode = atoi(argv[1]);

    qspi_cfg.mode = SetLineMode;      // 0: 1 line mode      3: 4 line mode
    qspi_cfg.clk_set = 0x10;
    qspi_cfg.wr_command = SetLineMode ? 0x38 : 0x02;          //write
    qspi_cfg.rd_command = SetLineMode ? 0xEB : 0x03;          //read
    qspi_cfg.wr_dummy_size = 0;
    qspi_cfg.rd_dummy_size = SetLineMode ? 0x06 : 0x00;

    bk_qspi_dcache_initialize(&qspi_cfg);
    bk_qspi_start();

    bk_qspi_dcache_write_data(0x00000, p_WRData1, QSPI_TEST_LENGTH);
    bk_qspi_dcache_write_data(0x04000, p_WRData2, QSPI_TEST_LENGTH);
    bk_qspi_dcache_write_data(0x08000, p_WRData3, QSPI_TEST_LENGTH);
    bk_qspi_dcache_write_data(0x0C000, p_WRData4, QSPI_TEST_LENGTH);
    bk_qspi_dcache_write_data(0x10000, p_WRData5, QSPI_TEST_LENGTH);

    rt_thread_delay(rt_tick_from_millisecond(100));

    bk_qspi_dcache_read_data(0x00000, p_RDData1, QSPI_TEST_LENGTH);
    bk_qspi_dcache_read_data(0x04000, p_RDData2, QSPI_TEST_LENGTH);
    bk_qspi_dcache_read_data(0x08000, p_RDData3, QSPI_TEST_LENGTH);
    bk_qspi_dcache_read_data(0x0C000, p_RDData4, QSPI_TEST_LENGTH);
    bk_qspi_dcache_read_data(0x10000, p_RDData5, QSPI_TEST_LENGTH);

    if(memcmp(p_WRData1, p_RDData1, QSPI_TEST_LENGTH*4) == 0)
    {
        rt_kprintf("[qspi_test]:qspi read data 1 pass \r\n ");
    }
}
```

```
    }
    else
    {
        rt_kprintf("[qspi_test]:qspi read data 1 error !!! \r\n ");

        for (i=0; i<QSPI_TEST_LENGTH; i++)
        {
            rt_kprintf("p_WRData[%d]=0x%lx, p_RDData[%d]=0x%lx\r\n", i, *(p_WRData1 +
i), i, *(p_RDData1 + i));
        }
    }

    if(memcmp(p_WRData2, p_RDData2, QSPI_TEST_LENGTH*4) == 0)
    {
        rt_kprintf("[qspi_test]:qspi read data 2 pass \r\n ");
    }
    else
    {
        rt_kprintf("[qspi_test]:qspi read data 2 error !!! \r\n ");

        for (i=0; i<QSPI_TEST_LENGTH; i++)
        {
            rt_kprintf("p_WRData[%d]=0x%lx, p_RDData[%d]=0x%lx\r\n", i, *(p_WRData2 +
i), i, *(p_RDData2 + i));
        }
    }

    if(memcmp(p_WRData3, p_RDData3, QSPI_TEST_LENGTH*4) == 0)
    {
        rt_kprintf("[qspi_test]:qspi read data 3 pass \r\n ");
    }
    else
    {
        rt_kprintf("[qspi_test]:qspi read data 3 error !!! \r\n ");
        for (i=0; i<QSPI_TEST_LENGTH; i++)
        {
            rt_kprintf("p_WRData[%d]=0x%lx, p_RDData[%d]=0x%lx\r\n", i, *(p_WRData3 +
i), i, *(p_RDData3 + i));
        }
    }
}
```

```
if(memcmp(p_WRData4, p_RDData4, QSPI_TEST_LENGTH*4) == 0)
{
    rt_kprintf("[qspi_test]:qspi read data 4 pass \r\n ");
}
else
{
    rt_kprintf("[qspi_test]:qspi read data 4 error !!! \r\n ");

    for (i=0; i<QSPI_TEST_LENGTH; i++)
    {
        rt_kprintf("p_WRData[%d]=0x%lx, p_RDData[%d]=0x%lx\r\n", i, *(p_WRData4 +
i), i, *(p_RDData4 + i));
    }
}

if(memcmp(p_WRData5, p_RDData5, QSPI_TEST_LENGTH*4) == 0)
{
    rt_kprintf("[qspi_test]:qspi read data 5 pass \r\n ");
}
else
{
    rt_kprintf("[qspi_test]:qspi read data 5 error !!! \r\n ");

    for (i=0; i<QSPI_TEST_LENGTH; i++)
    {
        rt_kprintf("p_WRData[%d]=0x%lx, p_RDData[%d]=0x%lx\r\n", i, *(p_WRData5 +
i), i, *(p_RDData5 + i));
    }
}

if(p_WRData1 != RT_NULL)
{
    rt_free(p_WRData1);
    p_WRData1= RT_NULL;
}
if(p_WRData2 != RT_NULL)
{
    rt_free(p_WRData2);
    p_WRData2= RT_NULL;
```

```
}  
if(p_WRData3 != RT_NULL)  
{  
    rt_free(p_WRData3);  
    p_WRData3= RT_NULL;  
}  
if(p_WRData4 != RT_NULL)  
{  
    rt_free(p_WRData4);  
    p_WRData4= RT_NULL;  
}  
if(p_WRData5 != RT_NULL)  
{  
    rt_free(p_WRData5);  
    p_WRData5= RT_NULL;  
}  
  
if(p_RDData1 != RT_NULL)  
{  
    rt_free(p_RDData1);  
    p_RDData1= RT_NULL;  
}  
if(p_RDData2 != RT_NULL)  
{  
    rt_free(p_RDData2);  
    p_RDData2= RT_NULL;  
}  
if(p_RDData3 != RT_NULL)  
{  
    rt_free(p_RDData3);  
    p_RDData3= RT_NULL;  
}  
if(p_RDData4 != RT_NULL)  
{  
    rt_free(p_RDData4);  
    p_RDData4= RT_NULL;  
}  
if(p_RDData5 != RT_NULL)  
{  
    rt_free(p_RDData5);
```

```
        p_RDData5= RT_NULL;
    }
}
else
{
    rt_kprintf("[qspi_test]:argc error!!! \r\n");
}
}

FINISH_FUNCTION_EXPORT_ALIAS(qspi_psram_dcache_test, __cmd_qspi_test, test
qspi_psram_dcache mode);
#endif
```

28.4 操作说明

示例代码参考test\qspi_test.c，打开宏定义：QSPI_TEST，开启qspi dcache模式的测试。使用qspi dcache模式需要将芯片外接psram。

28.4.1 运行现象

输入命令：qspi_test 0，可以看到log打印存取数据和读取数据都能成功，log如下：

```
msh />
msh />qspi_test 0
[qspi_test]:test_qspi_dcache_write_read_data
[qspi_test]:qspi read data 1 pass
[qspi_test]:qspi read data 2 pass
[qspi_test]:qspi read data 3 pass
[qspi_test]:qspi read data 4 pass
[qspi_test]:qspi read data 5 pass
msh />
msh />
```

图28.4.1-1

28.5 注意事项

- 测试中需要外接psram芯片。

29 BLE

29.1 BLE简介

BK7251目前只支持BLE做slave模式，master模式暂不支持。
samples_config.h开启

```
#define USING_BLE_TEST
```

29.2 BLE Related API（通用）

29.2.1 启动ble协议栈

```
void ble_activate(char *ble_name);
```

参数	描述
ble_name	可以传入NULL
返回	无

29.2.2 设置write callback

```
void ble_write_callback(write_req_t *param);
```

参数	描述
func	write操作的callback函数.函数定义: typedef void(*bk_ble_write_cb_t) (write_req_t *write_req);
返回	无

29.2.3 设置read callback

```
void ble_set_read_cb(ble_read_cb_t func);
```

参数	描述
func	read操作的callback函数。函数定义: typedef uint8_t (*bk_ble_read_cb_t) (read_req_t *read_req);
返回	无

29.2.4 设置event callback

```
void ble_set_event_cb(ble_event_cb_t func);
```

参数	描述
func	事件处理的callback函数。函数定义： typedef void (*ble_event_cb_t)(ble_event_t event, void *param);
返回	无

29.3 BLE Related API（slave）

29.3.1 开始广播

```
ble_err_t appm_start_advertising(void);
```

参数	描述
void	无
返回	ERR_SUCCESS: 成功; 其它: 失败

29.3.2 关闭广播

```
ble_err_t appm_stop_advertising(void);
```

参数	描述
void	无
返回	ERR_SUCCESS: 成功; 其它: 失败

29.3.3 发送数据

```
void ayla_wifi_send_statu_ntf_value(uint32_t len,uint8_t *buf,uint16_t seq_num);
```

参数	描述
len	数据长度（byte）
buf	数据指针
seq_num	顺序（传入0xFF即可）
返回	无

```
void ayla_wifi_send_scre_ntf_value(uint32_t len,uint8_t *buf,uint16_t seq_num);
```

参数	描述
len	数据长度（byte）

buf	数据指针
seq_num	顺序（传入0xFF即可）
返回	无

Note: 发送数据针对不同的characteristic拥有不同的接口，此处只列举了ayla service上的两个接口，其它characteristic对应的接口仅函数名不同，参数相同

29.3.4 断开连接

```
void appm_disconnect(uint8_t reason);
```

参数	描述
reason	断链原因(一般为0x13)
返回	无

29.4 BLE结构体说明

29.4.1 广播参数

adv_info_t

advData	Advertising数据(最大长度为31byte)
advDataLen	Advertising数据长度
respData	Response数据(最大长度为31byte)
respDataLen	Response数据长度
channel_map	Channel map(默认0x7: 37, 38, 39 channel都发送)
interval_min	最小interval(单位: 625us)
interval_max	最大interval(单位: 625us)

Note: 调用发送广播接口前需要先进行广播参数配置，广播参数需要配置到全局结构adv_info_t adv_info中，默认channel_map是0x7，interval_min为160，interval_max为160，若对adv_info执行了memset操作，需要重新配置channel_map，interval_min，interval_max，否则会广播参数错误

29.5 BLE示例代码

29.5.1 关键说明

• BLE枚举类型说明

```
typedef enum
{
    BLE_STACK_OK,
```



```
BLE_STACK_FAIL,
BLE_CONNECT,
BLE_DISCONNECT,
BLE_MTU_CHANGE,
BLE_CFG_NOTIFY,
BLE_CFG_INDICATE,
BLE_TX_DONE,
BLE_GEN_DH_KEY,
BLE_GET_KEY,
BLE_ATT_INFO_REQ,
BLE_CREATE_DB_OK,
BLE_CREATE_DB_FAIL,
} ble_event_t;

typedef enum
{
    ERR_SUCCESS = 0,
    ERR_STACK_FAIL,
    ERR_MEM_FAIL,
    ERR_INVALID_ADV_DATA,
    ERR_ADV_FAIL,
    ERR_STOP_ADV_FAIL,
    ERR_GATT_INDICATE_FAIL,
    ERR_GATT_NOTIFY_FAIL,
    ERR_SCAN_FAIL,
    ERR_STOP_SCAN_FAIL,
    ERR_CONN_FAIL,
    ERR_STOP_CONN_FAIL,
    ERR_DISCONN_FAIL,
    ERR_READ_FAIL,
    ERR_WRITE_FAIL,
    ERR_REQ_RF,
    /* Add more BLE error code hereafter */
} ble_err_t; //函数返回类型
```

29.5.2 示例代码

```
/*
* 程序清单： 这是一个简单ble的使用例程。
* 命令调用格式： ble_command active/start_adv/stop_adv/notify/indicate等
```

* 程序功能：通过ble命令实现开启协议栈，广播，扫描，连接等功能。

*/

```
#include <rtthread.h>
```

```
#include <finsh.h>
```

```
#include "common.h"
```

```
#include "ble_api.h"
```

```
#include "ble_pub.h"
```

```
#include "app_sdp.h"
```

```
#include "param_config.h"
```

```
#include "app_task.h"
```

```
#include "ble_cmd.h"
```

```
#include "application.h"
```

```
#include "samples_config.h"
```

```
#include "uart_pub.h"
```

```
#ifdef USING_BLE_TEST
```

```
#define BK_ATT_DECL_PRIMARY_SERVICE_128    {0x00,0x28,0,0,0,0,0,0,0,0,0,0,0,0,0,0}
```

```
#define BK_ATT_DECL_CHARACTERISTIC_128     {0x03,0x28,0,0,0,0,0,0,0,0,0,0,0,0,0,0}
```

```
#define BK_ATT_DESC_CLIENT_CHAR_CFG_128    {0x02,0x29,0,0,0,0,0,0,0,0,0,0,0,0,0,0}
```

```
#define WRITE_REQ_CHARACTERISTIC_128
```

```
{0x01,0xFF,0,0,0x34,0x56,0,0,0,0,0x28,0x37,0,0,0,0}
```

```
#define INDICATE_CHARACTERISTIC_128
```

```
{0x02,0xFF,0,0,0x34,0x56,0,0,0,0,0x28,0x37,0,0,0,0}
```

```
#define NOTIFY_CHARACTERISTIC_128
```

```
{0x03,0xFF,0,0,0x34,0x56,0,0,0,0,0x28,0x37,0,0,0,0}
```

```
static const uint8_t test_svc_uuid[16] = {0xFF,0xFF,0,0,0x34,0x56,0,0,0,0,0x28,0x37,0,0,0,0};
```

```
enum
```

```
{
```

```
    TEST_IDX_SVC,
```

```
    TEST_IDX_FF01_VAL_CHAR,
```

```
    TEST_IDX_FF01_VAL_VALUE,
```

```
    TEST_IDX_FF02_VAL_CHAR,
```

```
    TEST_IDX_FF02_VAL_VALUE,
```

```
    TEST_IDX_FF02_VAL_IND_CFG,
```

```
    TEST_IDX_FF03_VAL_CHAR,
```

```
    TEST_IDX_FF03_VAL_VALUE,
```

```
    TEST_IDX_FF03_VAL_NTF_CFG,
```

```
    TEST_IDX_NB,
```

```
};

bk_attn_desc_t test_att_db[TEST_IDX_NB] =
{
    // Service Declaration
    [TEST_IDX_SVC]          = {BK_ATT_DECL_PRIMARY_SERVICE_128,
BK_PERM_SET(RD, ENABLE), 0, 0},

    // Level Characteristic Declaration
    [TEST_IDX_FF01_VAL_CHAR] = {BK_ATT_DECL_CHARACTERISTIC_128,
BK_PERM_SET(RD, ENABLE), 0, 0},
    // Level Characteristic Value
    [TEST_IDX_FF01_VAL_VALUE] = {WRITE_REQ_CHARACTERISTIC_128,
BK_PERM_SET(WRITE_REQ, ENABLE), BK_PERM_SET(RI,
ENABLE)|BK_PERM_SET(UUID_LEN, UUID_16), 128},

    [TEST_IDX_FF02_VAL_CHAR] = {BK_ATT_DECL_CHARACTERISTIC_128,
BK_PERM_SET(RD, ENABLE), 0, 0},
    // Level Characteristic Value
    [TEST_IDX_FF02_VAL_VALUE] = {INDICATE_CHARACTERISTIC_128,
BK_PERM_SET(IND, ENABLE), BK_PERM_SET(RI, ENABLE)|BK_PERM_SET(UUID_LEN,
UUID_16), 128},

    // Level Characteristic - Client Characteristic Configuration Descriptor

    [TEST_IDX_FF02_VAL_IND_CFG] = {BK_ATT_DESC_CLIENT_CHAR_CFG_128,
BK_PERM_SET(RD, ENABLE)|BK_PERM_SET(WRITE_REQ, ENABLE), 0, 0},

    [TEST_IDX_FF03_VAL_CHAR] = {BK_ATT_DECL_CHARACTERISTIC_128,
BK_PERM_SET(RD, ENABLE), 0, 0},
    // Level Characteristic Value
    [TEST_IDX_FF03_VAL_VALUE] = {NOTIFY_CHARACTERISTIC_128,
BK_PERM_SET(NTF, ENABLE), BK_PERM_SET(RI, ENABLE)|BK_PERM_SET(UUID_LEN,
UUID_16), 128},

    // Level Characteristic - Client Characteristic Configuration Descriptor

    [TEST_IDX_FF03_VAL_NTF_CFG] = {BK_ATT_DESC_CLIENT_CHAR_CFG_128,
BK_PERM_SET(RD, ENABLE)|BK_PERM_SET(WRITE_REQ, ENABLE), 0, 0},
};
```

```
ble_err_t bk_ble_init(void)
{
    ble_err_t status;
    struct bk_ble_db_cfg ble_db_cfg;

    ble_db_cfg.att_db = test_att_db;
    ble_db_cfg.att_db_nb = TEST_IDX_NB;
    ble_db_cfg.prf_task_id = 0;
    ble_db_cfg.start_hdl = 0;
    ble_db_cfg.svc_perm = BK_PERM_SET(SVC_UUID_LEN, UUID_16);
    memcpy(&(ble_db_cfg.uuid[0]), &test_svc_uuid[0], 16);

    status = bk_ble_create_db(&ble_db_cfg);

    return status;
}

void appm_adv_data_decode(uint8_t len, const uint8_t *data)
{
    uint8_t index;
    uint8_t i;
    for(index = 0; index < len;)
    {
        switch(data[index + 1])
        {
            case 0x01:
            {
                bk_printf("AD_TYPE : ");
                for(i = 0; i < data[index] - 1; i++)
                {
                    bk_printf("%02x ", data[index + 2 + i]);
                }
                bk_printf("\r\n");
                index += (data[index] + 1);
            }
            break;
            case 0x08:
            case 0x09:
            {
```

```
        bk_printf("ADV_NAME : ");
        for(i = 0; i < data[index] - 1; i++)
        {
            bk_printf("%c",data[index + 2 + i]);
        }
        bk_printf("\r\n");
        index +=(data[index] + 1);
    }
    break;
case 0x02:
    {
        bk_printf("UUID : ");
        for(i = 0; i < data[index] - 1;)
        {
            bk_printf("%02x%02x  ",data[index + 2 + i],data[index + 3 + i]);
            i+=2;
        }
        bk_printf("\r\n");
        index +=(data[index] + 1);
    }
    break;
default:
    {
        index +=(data[index] + 1);
    }
    break;
}
return ;
}

void ble_write_callback(write_req_t *write_req)
{
    bk_printf("write_cb[prf_id:%d, att_idx:%d, len:%d]\r\n", write_req->prf_id, write_req->att_idx,
write_req->len);
    for(int i = 0; i < write_req->len; i++)
    {
        bk_printf("0x%x ", write_req->value[i]);
    }
    bk_printf("\r\n");
}
```

```
}

uint8_t ble_read_callback(read_req_t *read_req)
{
    bk_printf("read_cb[prf_id:%d, att_idx:%d]\r\n", read_req->prf_id, read_req->att_idx);
    read_req->value[0] = 0x10;
    read_req->value[1] = 0x20;
    read_req->value[2] = 0x30;
    return 3;
}

void ble_event_callback(ble_event_t event, void *param)
{
    switch(event)
    {
        case BLE_STACK_OK:
        {
            os_printf("STACK INIT OK\r\n");
            bk_ble_init();
        }
        break;
        case BLE_STACK_FAIL:
        {
            os_printf("STACK INIT FAIL\r\n");
        }
        break;
        case BLE_CONNECT:
        {
            os_printf("BLE CONNECT\r\n");
        }
        break;
        case BLE_DISCONNECT:
        {
            os_printf("BLE DISCONNECT\r\n");
        }
        break;
        case BLE_MTU_CHANGE:
        {
            os_printf("BLE_MTU_CHANGE:%d\r\n", *(uint16_t *)param);
        }
    }
}
```

```
break;
case BLE_TX_DONE:
{
    os_printf("BLE_TX_DONE\r\n");
}
break;
case BLE_GEN_DH_KEY:
{
    os_printf("BLE_GEN_DH_KEY\r\n");
    os_printf("key_len:%d\r\n", ((struct ble_gen_dh_key_ind *)param)->len);
    for(int i = 0; i < ((struct ble_gen_dh_key_ind *)param)->len; i++)
    {
        os_printf("%02x ", ((struct ble_gen_dh_key_ind *)param)->result[i]);
    }
    os_printf("\r\n");
}
break;
case BLE_GET_KEY:
{
    os_printf("BLE_GET_KEY\r\n");
    os_printf("pri_len:%d\r\n", ((struct ble_get_key_ind *)param)->pri_len);
    for(int i = 0; i < ((struct ble_get_key_ind *)param)->pri_len; i++)
    {
        os_printf("%02x ", ((struct ble_get_key_ind *)param)->pri_key[i]);
    }
    os_printf("\r\n");
}
break;
case BLE_CREATE_DB_OK:
{
    os_printf("CREATE DB SUCCESS\r\n");
}
break;
default:
    os_printf("UNKNOWN EVENT\r\n");
break;
}
}

static void ble_command_usage(void)
```

```
{
    os_printf("ble help          - Help information\r\n");
    os_printf("ble active        - Active ble to with BK7231BTxxx\r\n");
    os_printf("ble start_adv     - Start advertising as a slave device\r\n");
    os_printf("ble stop_adv      - Stop advertising as a slave device\r\n");
    os_printf("ble notify prf_id att_id value\r\n");
    os_printf("                  - Send ntf value to master\r\n");
    os_printf("ble indicate prf_id att_id value\r\n");
    os_printf("                  - Send ind value to master\r\n");

    os_printf("ble disc          - Disconnect\r\n");
}

static void ble_get_info_handler(void)
{
    UINT8 *ble_mac;
    os_printf("\r\n***** ble information *****\r\n");

    if (ble_is_start() == 0) {
        os_printf("no ble startup      \r\n");
        return;
    }

    ble_mac = ble_get_mac_addr();
    os_printf("* name: %s          \r\n", ble_get_name());
    os_printf("* mac:%02x-%02x-%02x-%02x-%02x-%02x\r\n", ble_mac[0],
ble_mac[1],ble_mac[2],ble_mac[3],ble_mac[4],ble_mac[5]);
    os_printf("***** end *****\r\n");
}

typedef adv_info_t ble_adv_param_t;

static void ble_advertise(void)
{
    UINT8 mac[6];
    char ble_name[20];
    UINT8 adv_idx, adv_name_len;

    wifi_get_mac_address((char *)mac, CONFIG_ROLE_STA);
    adv_name_len = snprintf(ble_name, sizeof(ble_name), "bk72xx-%02x%02x", mac[4], mac[5]);
```



```
memset(&adv_info, 0x00, sizeof(adv_info));

adv_info.channel_map = 7;
adv_info.interval_min = 160;
adv_info.interval_max = 160;

adv_idx = 0;
adv_info.advData[adv_idx] = 0x02; adv_idx++;
adv_info.advData[adv_idx] = 0x01; adv_idx++;
adv_info.advData[adv_idx] = 0x06; adv_idx++;

adv_info.advData[adv_idx] = adv_name_len + 1; adv_idx +=1;
adv_info.advData[adv_idx] = 0x09; adv_idx +=1; //name
memcpy(&adv_info.advData[adv_idx], ble_name, adv_name_len); adv_idx +=adv_name_len;

adv_info.advDataLen = adv_idx;

adv_idx = 0;

adv_info.respData[adv_idx] = adv_name_len + 1; adv_idx +=1;
adv_info.respData[adv_idx] = 0x08; adv_idx +=1; //name
memcpy(&adv_info.respData[adv_idx], ble_name, adv_name_len); adv_idx +=adv_name_len;
adv_info.respDataLen = adv_idx;

if (ERR_SUCCESS != appm_start_advertising())
{
    os_printf("ERROR\r\n");
}

static void ble_command(int argc, char **argv)
{
    ble_adv_param_t adv_param;

    if ((argc < 2) || (os_strcmp(argv[1], "help") == 0))
    {
        ble_command_usage();
        return;
    }
}
```

```
if (os_strcmp(argv[1], "active") == 0)
{

    ble_set_write_cb(ble_write_callback);
    ble_set_read_cb(ble_read_callback);
    ble_set_event_cb(ble_event_callback);ble_activate(NULL);
}
else if(os_strcmp(argv[1], "start_adv") == 0)
{
    ble_advertise();
}
else if(os_strcmp(argv[1], "stop_adv") == 0)
{
    if(ERR_SUCCESS != appm_stop_advertising())
    {
        os_printf("ERROR\r\n");
    }
}
else if(os_strcmp(argv[1], "notify") == 0)
{
    uint8 len;
    uint16 prf_id;
    uint16 att_id;
    uint8 write_buffer[20];

    if(argc != 5)
    {
        ble_command_usage();
        return;
    }

    len = os_strlen(argv[4]);
    if(len % 2 != 0)
    {
        os_printf("ERROR\r\n");
        return;
    }
    hexstr2bin(argv[4], write_buffer, len/2);

    prf_id = atoi(argv[2]);
```

```
att_id = atoi(argv[3]);

if(ERR_SUCCESS != bk_ble_send_ntf_value(len / 2, write_buffer, prf_id, att_id))
{
    os_printf("ERROR\r\n");
}
}
else if(os_strcmp(argv[1], "indicate") == 0)
{
    uint8 len;
    uint16 prf_id;
    uint16 att_id;
    uint8 write_buffer[20];

    if(argc != 5)
    {
        ble_command_usage();
        return;
    }

    len = os_strlen(argv[4]);
    if(len % 2 != 0)
    {
        os_printf("ERROR\r\n");
        return;
    }
    hexstr2bin(argv[4], write_buffer, len/2);

    prf_id = atoi(argv[2]);
    att_id = atoi(argv[3]);

    if(ERR_SUCCESS != bk_ble_send_ind_value(len / 2, write_buffer, prf_id, att_id))
    {
        os_printf("ERROR\r\n");
    }
}
else if(os_strcmp(argv[1], "disc") == 0)
{
    appm_disconnect();
}
```

```
}  
MSH_CMD_EXPORT(ble_command,ble_command);  
  
#endif
```

29.6 操作说明

29.6.1 数据交互

• 下载调试工具

android应用商店搜索Ble调试工具，下载后打开，界面如下图所示：



图29.6.1-1



图29.6.1-2

• 开启广播

设备上电后，调试串口输入**ble_command active**启动协议栈,然后输入**ble_command start_adv**开启广播，设备log如下：

```
msh />ble_command active  
ble start no ble name  
ble name:BK7231BT-01, c9:47:8c:8b:22:48  
-----rw_main task init----
```

```
-----rw_main  start-----  
gapm_cmp_evt_handler operation = 0x1, status = 0x0  
gapm_cmp_evt_handler operation = 0x3, status = 0x0  
STACK INIT OK  
ble create new db  
ble_env->start_hdl = 0x7  
gapm_cmp_evt_handler operation = 0x1b, status = 0x0  
BLE_CREATE_DB_OK  
msh />  
msh />ble_command start_adv  
channel_map=7,interval=[160,160]  
appm start advertising
```

• 数据通信

手机打开BLE调试APP，点击扫描所有设备并显示，点击bk-2248,连接BLE设备，结果如图29.6.1-3和图29.6.1-4所示。



图29.6.1-3



图29.6.1-4

APP成功连接BLE设备，设备log如下：

```
msh />ble start_adv  
channel_map=7,interval=[160,160]  
appm start advertising  
gapc_connection_req_ind_handler  
BLE CONNECT
```

```
gapm_cmp_evt_handler operation = 0xd, status = 0x0
```

手机点进服务(4)，发送0x40给BLE设备，会立刻收到返回值，如下图所示：



图29.6.1-5

设备收到数据为64和APP下发的0x40一致，log如下所示：

```
msh />
msh />---vbat voltage:3055---
---vbat voltage:3055---
gapc_connection_req_ind_handler
BLE CONNECT
gapm_cmp_evt_handler operation = 0xd, status = 0x0
---vbat voltage:3053---
ble_write_callback prf_id:0,att_idx:4
value[0]:64
```

30 USB充电模式

30.1 USB充电模式简介

BK7251的充电电路分为外部充电和内部充电，外部充电硬件需要额外添加PNP管8550。充电程序需要用到充电参数和vbat adc检测参数，其中vbat adc校准需要两步，一共四个字节，例如1c 10 7d 89。不同芯片间的参数存在微小，所以需要先校准后使用，默认出厂后校准参数存储在efuse的16–19字节，无需再次校准。如果efuse中的参数没有读取到，用户可以自己校准一次后存储在TLV中，供充电使用。

充电方式可以用宏CFG_CHARGE_MODE选择，内部充电可以使用CHARGE_INTERNAL_HW_MODE或CHARGE_INTERNAL_SW_MODE，外部充电可以使用CHARGE_EXTERNAL_HW_MODE或CHARGE_EXTERNAL_SW_MODE。

30.2 USB充电模式 Related API

充电功能的相关接口参考beken378\func\usb_plug\usb_plug.c，相关接口如下：

函数	描述
vbat_adc_cal_step1()	vbat adc 校准第一步
usb_charger_calibration()	充电参数校准
usb_charge_start()	充电开始函数
usb_charge_stop()	充电停止函数

30.2.1 vbat adc 校准第一步

vbat adc校准第一步调用之前：需要VUSB外接5v电压，VBAT不加电压。先调用本函数再调用usb_charger_calibration。

```
int vbat_adc_cal_step1(void);
```

参数	描述
void	空
返回	0：成功；-1：错误

30.2.2 充电参数校准

充电参数调用之前：需要VUSB外接5v电压，VBAT外接4.2v电压。usb_charger_calibration中会包含vbat adc校准第二步，所以需要在vbat_adc_cal_step1之后调用，完成后会自动将结果存储在TLV中。

```
void usb_charger_calibration(void);
```

参数	描述
void	空
返回	空

30.2.3 开始充电模式

```
void usb_charge_start(CHARGE_STEP step, UINT32 charge_elect);
```

参数	描述
step	充电步骤的枚举类型
charge_elect	在软件控制下需要设置的充电电流大小
返回	空

充电开始函数时设置充电步骤:

```
typedef enum
{
    STEP_STOP = 0,          /*停止*/
    STEP_START = 1,         /*开始*/
    STEP_TRICKLE = 2,       /*小电流模式， 电池低于3v使用*/
    STEP_EXTER_CC = 3,      /*外部恒流模式*/
    STEP_INTER_CC = 4,      /*内部恒流模式*/
    STEP_INTER_CV = 5,      /*内部恒压模式*/
} CHARGE_STEP;
```

30.2.4 停止充电模式

```
void usb_charge_stop(void);
```

参数	描述
void	空
返回	空

30.3 示例代码

30.3.1 关键说明

- USB充电模式的宏定义:

在使用USB充电功能必须开启宏定义：CFG_USE_USB_CHARGE。

#define	CFG_USE_USB_CHARGE	使用USB充电功能
#define	CFG_CHARGE_MODE	选择使用充电方式(四种充电方式)
#define	CHARGE_INTERNAL_HW_MODE 0	硬件内部充电
#define	CHARGE_INTERNAL_SW_MODE 1	软件内部充电
#define	CHARGE_EXTERNAL_HW_MODE 2	硬件外部充电
#define	CHARGE_EXTERNAL_SW_MODE 3	软件外部充电

30.4 操作说明

目前默认且最高效的充电方式是硬件外部充电，下载好开启充电功能的bin文件，串口接到uart1，充电的电池设备电源连接vbat引脚，接地的引脚连接到usb的GND,对demo板供电设备：+5V的电压连接到VUSB引脚，连接和log信息如下图所示：



图30.4-1

30.4.1 运行现象

上电后，keep_count一直计数，表示正在充电，如果充电完成会打印charger_full，打印log如下：

```
cb4:5 40 keep_count:290
vbat 4208
cb4:5 40 keep_count:291
vbat 4208
cb4:5 40 keep_count:292
vbat 4208
cb4:5 40 keep_count:293
```



vbat 4208
cb4:5 40 keep_count:294
vbat 4208
cb4:5 40 keep_count:295
vbat 4208
cb4:5 40 keep_count:296
vbat 4208
cb4:5 40 keep_count:297
vbat 4208
cb4:5 40 keep_count:298
vbat 4208
cb4:5 40 keep_count:299
vbat 4209
cb4:5 40 keep_count:300
vbat 4209
OK,charger_full
vbat 4210
OK,charger_full
vbat 4210
OK,charger_full
vbat 4209
OK,charger_full

31 EZ_CONFIG配网

31.1 EZ_CONFIG简介

EZ_config配网是利用组播技术，通过对MAC地址编码，设备端及手机端统一协议，进行配网信息传输，为设备进行网络配置。

31.2 EZ_CONFIG Related API

EZ_config相关接口参考\samples\ezconfig\ezconfig.h。ezconfig依赖tinycrypt软件包，在rtconfig.h开启宏定义，

```
#define      PKG_USING_TINYCRYPT
#define      TINY_CRYPT_MD5
#define      TINY_CRYPT_AES
#define      TINY_CRYPT_AES_ROM_TABLES
#define      TINY_CRYPT_BASE64
```

开启ezconfig配网例程，sample_config.h 打开宏定义

```
#define      EZ_CONFIG_SAMPLE
```

相关接口如下：

31.2.1 开始EZ_CONFIG

```
void bk_ezconfig_init(void);
```

参数	描述
void	空
返回	0:成功；其他：失败

31.2.2 获取EZ_CONFIG状态

```
uint32_t bk_ezconfig_get_status(void);
```

参数	描述
void	空
返回	ezconfig状态 EZCONFIG_STATUS_CONTINUE = 0, EZCONFIG_STATUS_CHANNEL_LOCKED = 1, EZCONFIG_STATUS_COMPLETE = 2

31.2.3 获取EZ_CONFIG解码结果

```
EZconfig_result_t ezconfig_get_result(void);
```

参数	描述
void	为ezconfig库分配的内存
返回	EZconfig_result_t 地址

EZconfig_result_t:

char *passwd	wifi密码
char *ssid	wifi ssid
char ip[4]	手机端ip
unsigned char passwd_len;	wifi密码长度
unsigned char ssid_len;	wifi ssid长度
unsigned char ip_len;	ip地址长度
unsigned char reserved;	保留值

31.3 使用说明

手机需安装配网APP进行配网，配网APP位于../tool/ezconfig。
设备端开启ezconfig配网，log如图31.3-1所示：

```

ROMFS File System initialized!
Enter normal mode...

app_init finished
[32m[I/player] RT-Thread lightweight player v1.2.7_Release [build Mar  3 2020][0m
set volume 65-65
sh />start_ezconfig
ezconfig start
[DRV_WLAN]set monitor callback
[DRV_WLAN]start monitor
bk_wlan_stop-ap
bk_wlan_stop-sta
Soft AP_start
rl_init_csa_status
[saap]MM_RESET_REQ
[saap]ME_CONFIG_REQ
rv_msg_send me_config_req ps_on is 1
[saap]ME_CHAN_CONFIG_REQ
[saap]MM_START_REQ
mm_add_if_req_handler:0
apm start with vif:0
    beacon_int_set:100 TU
set_active_param 0
[msg]AFM_STOP_CFM
update_ongoing_l_bcn_update
mm_set_vif_state_req_handler
mm-next-timer_null
no_enter_monitor_in_the_remove-if-routine
mm_enter_monitor_mode
hal_machw_enter_monitor_mode
Switch channel 2
Switch channel 3
Switch channel 4
Switch channel 5
Switch channel 6
Switch channel 7
Switch channel 8
Switch channel 9

```

图31.3-1

32 BLE 配网

32.1 BLE 配网简介

BK725X开启ble slave,手机端通过与设备端进行ble连接，将wifi ssid和密码发送给设备，设备端获取配网信息后连接路由器完成配网。

32.2 BLE 配网 Related API

BLE配网相关接口参考samples\ble_netconfig.h。开启ble配网例程，sample_config.h 打开宏定义：

```
#define BLE_CONFIG_SMAPLE
```

相关接口如下：

函数	描述
bk_ble_netconfig_start ()	开启BLE配网
bk_ble_netconfig_stop ()	停止BLE配网
get_ble_netconfig_state ()	获取BLE 配网状态
result_cb(char *ssid, char *password,char *ble_get_openid, void *user_data, void *userdata_len)	获取配网信息

32.3 使用说明

扫描下方二维码或微信搜索小雅慧读公众号，进入配网微信小程序，设备端发送串口命令开启BLE配网，log如图32.3-1所示：

```
ble_netconfig_sample
msh />
msh />ble start no ble name
ble name:BK7231BT-01, c9:47:8c:38:ce:b3
-----rw_main task init-----
-----rw_main start-----
gapm_cmp_evt_handler operation = 0x1, status = 0x0
gapm_cmp_evt_handler operation = 0x3, status = 0x0
STACK INIT OK
ble create new db
ble_env->start_hdl = 0x7
gapm_cmp_evt_handler operation = 0x1b, status = 0x0
BLE_CREATE_DB_OK
appm start advertising
```

图32.3-1

33 AP 配网

33.1 AP 配网简介

AP配网即设备热点配网，设备处于AP模式，手机用于STA模式，手机连接到处于AP模式的设备后组成局域网，手机发送需要连接路由的ssid和password至设备，设备接收后找到对应的路由器主动去连接路由器，完成配网。

33.2 AP 配网 Related A

Apconfig示例代码\samples\apconfig\network_ap_config.c。apconfig依赖webnet软件包，在rtconfig.h开启宏定义，

```
#define      PKG_USING_WEBNET
#define      WEBNET_USING_CGI
```

开启apconfig配网例程，sample_config.h 打开宏定义

```
#define      AP_CONFIG_SAMPLE
```

33.3 使用说明

BK725X设备端开启AP模式，电脑或者手机端连接设备AP后打开浏览器，输入网址 http://192.168.169.1/cgi-bin/web_ap_config，输入wifi ssid password, 设备收到配网信息后开始连接路由器，完成配网。

```
app_init finished
[32m[I/player] RT-Thread lightweight player v1.2.7_Release [build Mar  3 2020][0m
set_volume 65-65
---bat voltage:3043---
wifi ap ap test 12345678
[DRV_WLAN]drivers\wlan\drv_wlan.c L975 beken_wlan_control cmd: case WIFI_INIT!
_wifi_softap: ssid:test key:12345678
rl_ap_request_enter
enter_timer-Astop
enter_timer-Astart
msh />
msh />rl_enter_handler
_ap_request_enter
rl_ap_start
Soft_AP_start
rl_init_csa_status
[saap]MM_RESET_REQ
[saap]ME_CONFIG_REQ
rw_msg_send_me_config_req ps_on is 1
[saap]ME_CHAN_CONFIG_REQ
[saap]MM_START_REQ
[csa]csa_in_progress[0:0]-clear
mm_add_if_req_handler:0
hapd_intf_add_vif,type:3, s:0, id:0
apm start with vif:0
---beacon_int_set:100 TU
set_active param 0
[msg]APM_STOP_CFM
update_ongoing_1_bcn_update
vif_idx:0, ch_idx:0, bmc_idx:2
```

图33.3-1 开启AP

```
ifconfig
network interface: ap
MTU: 1500
MAC: c8 47 8c 22 c8 b3
FLAGS: UP LINK_UP ETHARP BROADCAST IGMP
ip address: 192.168.169.1
gw address: 192.168.169.1
net mask : 255.255.255.0
network interface: w0 (Default)
MTU: 1500
MAC: c8 47 8c 22 c8 b2
FLAGS: UP LINK_DOWN ETHARP BROADCAST IGMP
ip address: 0.0.0.0
gw address: 0.0.0.0
net mask : 0.0.0.0
dns server #0: 0.0.0.0
dns server #1: 0.0.0.0
msh />
```

图33.3-2 查看设备ip



图33.3-3 浏览器打开链接

```
sta_ip_start
configuring interface wlan (with DHCP client)
dhcp_check_status_init_timer
[DHCP] [dhcp_server_recv:305] ap recv 308
[DHCP] dhcp_server_recv: recv_netif w0 skip
rl_launch_handler
rl_launch_handler
bad_channel vif_idx=0
rl_launch_handler
mm_switch_channel
mm_csa_set_channel
mm_csa_beacon_after_change
update_ongoing_1_bcn_update
rl_set_csa_switched
gen:300000
rx_hw_error
phy_interface_underrun
update_ongoing_1_bcn_update
rl_launch_handler
[DHCP] [dhcp_server_recv:305] ap recv 308
[DHCP] dhcp_server_recv: recv_netif w0 skip
IP UP: 192.168.1.102
[ip_up] start tick = 0, ip_up tick = 142500, total = 142500
[Flash]ENV isn't initialize OK
[Flash]ENV isn't initialize OK
[Flash]ENV isn't initialize OK
[Flash]ENV isn't initialize OK
[Flash]ENV isn't initialize OK
[Flash]ENV isn't initialize OK
write new profile to flash 0x001FF000 72 byte!
[Flash]ENV isn't initialize OK
[Flash]ENV isn't initialize OK
[Flash]ENV isn't initialize OK
[Flash]ENV isn't initialize OK
[Flash]ENV isn't initialize OK
[Flash]ENV isn't initialize OK
```

图33.3-4 设备配网成功