BK7251 RTOS SDK

API Reference



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](#_Toc35440305)

[1 ADC 18](#_Toc35440306)

[1.1 ADC简介 18](#_Toc35440307)

[1.2 ADC Related API 18](#_Toc35440308)

[1.2.1 adc通用结构体说明 18](#_Toc35440309)

[1.2.2 创建adc检测线程 18](#_Toc35440310)

[1.2.3 配置adc检测通道及回调函数 19](#_Toc35440311)

[1.2.4 启动adc 19](#_Toc35440312)

[1.2.5 关闭adc 19](#_Toc35440313)

[1.3 ADC示例代码 19](#_Toc35440314)

[2 PWM 22](#_Toc35440315)

[2.1 PWM简介 22](#_Toc35440316)

[2.2 PWM Related API 22](#_Toc35440317)

[2.2.1 pwm枚举类型说明 22](#_Toc35440318)

[2.2.2 初始化pwm 22](#_Toc35440319)

[2.2.3 启动pwm功能 23](#_Toc35440320)

[2.2.4 停止pwm功能 23](#_Toc35440321)

[2.3 PWM示例代码 23](#_Toc35440322)

[2.4 操作说明 24](#_Toc35440323)

[2.4.1 打开配置 24](#_Toc35440324)

[2.4.2 运行现象 24](#_Toc35440325)

[2.5 注意事项 25](#_Toc35440326)

[3 Audio 26](#_Toc35440327)

[3.1 Audio简介 26](#_Toc35440328)

[3.2 Audio Related API 26](#_Toc35440329)

[3.2.1 audio通用结构体说明 26](#_Toc35440330)

[3.2.2 audio设备初始化 26](#_Toc35440331)

[3.2.3 打开mic 27](#_Toc35440332)

[3.2.4 设置mic数据采集通道 27](#_Toc35440333)

[3.2.5 设置mic采样率 27](#_Toc35440334)

[3.2.6 采集mic声音数据 27](#_Toc35440335)

[3.2.7 关闭mic 28](#_Toc35440336)

[3.2.8 打开audio设备 28](#_Toc35440337)

[3.2.9 设置dac采样率 28](#_Toc35440338)

[3.2.10 设置dac音量 28](#_Toc35440339)

[3.2.11 获取dac数据，播放音频 28](#_Toc35440340)

[3.2.12 关闭dac 29](#_Toc35440341)

[3.3 Audio示例代码 29](#_Toc35440342)

[3.3.1 关键说明 29](#_Toc35440343)

[3.3.2 示例代码 29](#_Toc35440344)

[4 Button 34](#_Toc35440345)

[4.1 Button简介 34](#_Toc35440346)

[4.2 Button Realated API 34](#_Toc35440347)

[4.2.1 button初始化 34](#_Toc35440348)

[4.2.2 配置回调函数 34](#_Toc35440349)

[4.2.3 开始button工作 35](#_Toc35440350)

[4.2.4 结束button工作 35](#_Toc35440351)

[4.3 Button示例代码 35](#_Toc35440352)

[4.3.1 关键说明 35](#_Toc35440353)

[4.3.2 示例代码 35](#_Toc35440354)

[4.4 操作说明 38](#_Toc35440355)

[4.4.1 打开配置 38](#_Toc35440356)

[4.4.2 按键测试 38](#_Toc35440357)

[5 I2C总线 40](#_Toc35440358)

[5.1 I2C简介 40](#_Toc35440359)

[5.2 模拟I2C Related API 40](#_Toc35440360)

[5.2.1 寻找总线获取设备句柄 40](#_Toc35440361)

[5.2.2 对从设备的读写数据 40](#_Toc35440362)

[5.3 模拟I2C总线示例代码 41](#_Toc35440363)

[5.3.1 关键说明 41](#_Toc35440364)

[5.3.2 对从设备的读写数据 41](#_Toc35440365)

[5.4 硬件I2C总线示例代码 45](#_Toc35440366)

[5.4.1 硬件I2C接口说明 45](#_Toc35440367)

[5.4.2 对从设备的读写数据 45](#_Toc35440368)

[6 I2S总线 50](#_Toc35440369)

[6.1 I2S简介 50](#_Toc35440370)

[6.2 I2S Related API 50](#_Toc35440371)

[6.2.1 i2s通用结构体说明 50](#_Toc35440372)

[6.2.2 i2s模块参数设置 50](#_Toc35440373)

[6.2.3 i2s主从设备发送/接收数据 51](#_Toc35440374)

[6.3 I2S示例代码 51](#_Toc35440375)

[6.3.1 关键说明 51](#_Toc35440376)

[6.3.2 示例代码 52](#_Toc35440377)

[7 通用SPI 57](#_Toc35440378)

[7.1 通用SPI简介 57](#_Toc35440379)

[7.2 通用SPI Related API 57](#_Toc35440380)

[7.2.1 spi通用结构体说明 57](#_Toc35440381)

[7.2.2 spi模块配置 57](#_Toc35440382)

[7.2.3 spi发送数据 58](#_Toc35440383)

[7.2.4 spi接收数据 58](#_Toc35440384)

[7.3 通用SPI示例代码 58](#_Toc35440385)

[7.3.1 关键说明 59](#_Toc35440386)

[7.3.2 示例代码 59](#_Toc35440387)

[7.4 操作说明 62](#_Toc35440388)

[7.4.1 打开配置 62](#_Toc35440389)

[7.4.2 运行现象 62](#_Toc35440390)

[7.5 注意事项 63](#_Toc35440391)

[8 通用SPI FLASH设备 64](#_Toc35440392)

[8.1 通用SPI FLASH简介 64](#_Toc35440393)

[8.2 通用SPI FLASH Related API 64](#_Toc35440394)

[8.2.1 控制设备 64](#_Toc35440395)

[8.3 通用SPI FLASH示例代码 64](#_Toc35440396)

[8.3.1 关键说明 64](#_Toc35440397)

[8.3.2 示例代码 64](#_Toc35440398)

[8.4 操作说明 67](#_Toc35440399)

[8.4.1 运行现象 67](#_Toc35440400)

[8.5 注意事项 68](#_Toc35440401)

[9 通用SPI PSRAM设备 69](#_Toc35440402)

[9.1 通用SPI PSRAM简介 69](#_Toc35440403)

[9.2 通用SPI PSRAM related API 69](#_Toc35440404)

[9.3 通用SPI PSRAM示例代码 69](#_Toc35440406)

[9.3.1 关键说明 69](#_Toc35440407)

[9.3.2 示例代码 69](#_Toc35440408)

[9.4 操作说明 71](#_Toc35440409)

[9.4.1 打开配置 71](#_Toc35440410)

[9.4.2 运行现象 71](#_Toc35440411)

[9.5 注意事项 71](#_Toc35440412)

[10 高速SPI从设备 72](#_Toc35440413)

[10.1 高速SPI从设备简介 72](#_Toc35440414)

[10.2 高速SPI从设备Related API 72](#_Toc35440415)

[10.3 高速SPI从示例代码 72](#_Toc35440416)

[10.3.1 关键说明 72](#_Toc35440417)

[10.3.2 示例代码 72](#_Toc35440418)

[10.4 操作说明 75](#_Toc35440419)

[10.4.1 打开配置 75](#_Toc35440420)

[10.4.2 运行现象 75](#_Toc35440421)

[10.5 注意事项 75](#_Toc35440422)

[11 GPIO 76](#_Toc35440423)

[11.1 GPIO简介 76](#_Toc35440424)

[11.2 GPIO Related API 76](#_Toc35440425)

[11.2.1 设置引脚模式 76](#_Toc35440426)

[11.2.2 设置引脚电平 76](#_Toc35440427)

[11.2.3 读取引脚电平 77](#_Toc35440428)

[11.2.4 绑定引脚中断回调函数 77](#_Toc35440429)

[11.2.5 使能引脚中断 77](#_Toc35440430)

[11.2.6 脱离引脚中断回调函数 77](#_Toc35440431)

[11.3 GPIO示例代码 78](#_Toc35440432)

[11.3.1 关键说明 78](#_Toc35440433)

[11.3.2 示例代码 78](#_Toc35440434)

[11.4 操作说明 79](#_Toc35440435)

[11.4.1 打开配置 80](#_Toc35440436)

[11.4.2 运行现象 80](#_Toc35440437)

[12 UART 81](#_Toc35440438)

[12.1 UART简介 81](#_Toc35440439)

[12.2 UART Related API 81](#_Toc35440440)

[12.2.1 uart通用结构体说明 81](#_Toc35440441)

[12.2.2 控制串口设备 81](#_Toc35440442)

[12.3 UART示例代码 82](#_Toc35440443)

[12.3.1 关键说明 82](#_Toc35440444)

[12.3.2 示例代码 83](#_Toc35440445)

[12.4 操作说明 85](#_Toc35440446)

[12.4.1 打开配置 86](#_Toc35440447)

[12.4.2 运行现象 86](#_Toc35440448)

[12.5 注意事项 86](#_Toc35440449)

[13 Player 87](#_Toc35440450)

[13.1 Player简介 87](#_Toc35440451)

[13.2 Player Related API 87](#_Toc35440452)

[13.2.1 初始化播放器 87](#_Toc35440453)

[13.2.2 开始/恢复播放 87](#_Toc35440454)

[13.2.3 停止播放 88](#_Toc35440455)

[13.2.4 暂停播放 88](#_Toc35440456)

[13.2.5 设置当前歌曲播放的位置 88](#_Toc35440457)

[13.2.6 设置播放歌曲的uri 88](#_Toc35440458)

[13.2.7 获取当前歌曲的uri 88](#_Toc35440459)

[13.2.8 获取当前播放器的状态 89](#_Toc35440460)

[13.2.9 准备开始播放用户音频数据流 89](#_Toc35440461)

[13.2.10 写入用户音频数据到播放器 89](#_Toc35440462)

[13.2.11 设置用户音频数据的总长度 89](#_Toc35440463)

[13.2.12 设置播放层事件的回调函数 89](#_Toc35440464)

[13.2.13 设置播放器音量 90](#_Toc35440465)

[13.2.14 设置播放器音量 90](#_Toc35440466)

[13.2.15 获取当前歌曲的持续时间 90](#_Toc35440467)

[13.2.16 获取当前歌曲播放的位置 90](#_Toc35440468)

[13.3 Player示例代码 90](#_Toc35440469)

[13.4 操作说明 98](#_Toc35440470)

[13.4.1 运行现象 98](#_Toc35440471)

[14 网络接口 100](#_Toc35440472)

[14.1 网络接口简介 100](#_Toc35440473)

[14.2 网络接口 Related API 100](#_Toc35440474)

[14.2.1 启动网络 100](#_Toc35440475)

[14.2.2 启动STATION快速连接 101](#_Toc35440476)

[14.2.3 关闭网络 102](#_Toc35440477)

[14.2.4 启动scan 102](#_Toc35440478)

[14.2.5 注册scan结束后的回调函数 102](#_Toc35440479)

[14.2.6 scan特定的网络 102](#_Toc35440480)

[14.2.7 启动监听模式 102](#_Toc35440481)

[14.2.8 关闭监听模式 103](#_Toc35440482)

[14.2.9 注册监听回调函数 103](#_Toc35440483)

[14.2.10 获取当前的网络状态 103](#_Toc35440484)

[14.2.11 获取当前的连接状态 104](#_Toc35440485)

[14.2.12 获取当前的信道 104](#_Toc35440486)

[14.2.13 设置信道 105](#_Toc35440487)

[14.3 网络接口使用示例 105](#_Toc35440488)

[14.3.1 关键说明 105](#_Toc35440489)

[14.3.2 代码示例 105](#_Toc35440490)

[14.4 操作说明 115](#_Toc35440491)

[14.4.1 启动STATION连接 115](#_Toc35440492)

[14.4.2 启动STATION快速连接 116](#_Toc35440493)

[14.4.3 STATION模式获取状态 117](#_Toc35440494)

[14.4.4 启动AP 117](#_Toc35440495)

[14.4.5 AP模式获取状态 117](#_Toc35440496)

[14.4.6 启动SCAN 118](#_Toc35440497)

[14.4.7 启动混杂包监听 119](#_Toc35440498)

[15 RTOS接口 120](#_Toc35440499)

[15.1 RTOS接口简介 120](#_Toc35440500)

[15.2 RTOS Related APIs 120](#_Toc35440501)

[15.2.1 RTOS结构体说明 120](#_Toc35440502)

[15.2.2 创建一个新的线程 121](#_Toc35440503)

[15.2.3 删除一个使用结束的线程 122](#_Toc35440504)

[15.2.4 使当前线程挂起，等待另一个线程终止 122](#_Toc35440505)

[15.2.5 使一个线程挂起一段时间 122](#_Toc35440506)

[15.2.6 初始化一个信号量 122](#_Toc35440507)

[15.2.7 发出信号量 122](#_Toc35440508)

[15.2.8 获取一个信号量，并提供超时机制 123](#_Toc35440509)

[15.2.9 销毁一个信号量 123](#_Toc35440510)

[15.2.10 初始化一个互斥锁 123](#_Toc35440511)

[15.2.11 获得一个互斥锁 123](#_Toc35440512)

[15.2.12 释放一个互斥锁 123](#_Toc35440513)

[15.2.13 销毁一个互斥锁 124](#_Toc35440514)

[15.2.14 初始化一个消息队列 124](#_Toc35440515)

[15.2.15 将一个数据对象推入消息队列 124](#_Toc35440516)

[15.2.16 从消息队列中取出一个数据对象 124](#_Toc35440517)

[15.2.17 销毁一个消息队列 125](#_Toc35440518)

[15.2.18 查询一个队列是否为空 125](#_Toc35440519)

[15.2.19 查询一个队列是否已满 125](#_Toc35440520)

[15.2.20 初始化一个时钟，并传入回调函数 125](#_Toc35440521)

[15.2.21 启动一个时钟 126](#_Toc35440522)

[15.2.22 停止一个时钟 126](#_Toc35440523)

[15.2.23 重新加载一个过期的时钟 126](#_Toc35440524)

[15.2.24 销毁一个时钟 126](#_Toc35440525)

[15.2.25 获取一个时钟是否正在运行 126](#_Toc35440526)

[15.3 RTOS关键说明 127](#_Toc35440527)

[16 OTA 128](#_Toc35440528)

[16.1 OTA简介 128](#_Toc35440529)

[16.2 OTA Related API 128](#_Toc35440530)

[16.2.1 fal初始化 128](#_Toc35440531)

[16.2.2 远程下载固件 128](#_Toc35440532)

[16.3 OTA示例代码 129](#_Toc35440533)

[16.4 操作说明 130](#_Toc35440534)

[16.4.1 生成rbl升级文件 130](#_Toc35440535)

[16.4.2 搭建本地HTTP Server环境 130](#_Toc35440536)

[16.4.3 运行现象 131](#_Toc35440537)

[17 Bootloader 133](#_Toc35440538)

[17.1 Bootloader简介 133](#_Toc35440539)

[17.2 分区表的设置 133](#_Toc35440540)

[17.2.1 Bootloader分区 133](#_Toc35440541)

[17.2.2 App分区 133](#_Toc35440542)

[17.2.3 Download分区 133](#_Toc35440543)

[17.3 L\_boot 134](#_Toc35440544)

[17.4 UP\_boot 134](#_Toc35440545)

[17.5 获取bootloader.bin文件 134](#_Toc35440546)

[17.6 生成all.bin文件 134](#_Toc35440547)

[17.7 Bootloader示例代码 135](#_Toc35440548)

[17.7.1 2M分区表信息配置文件partition\_audio\_2M.json示例 135](#_Toc35440549)

[17.7.2 UP\_boot示例 136](#_Toc35440550)

[17.7.3 生成all.bin的配置文件config\_sample.json示例 139](#_Toc35440551)

[18 低功耗 140](#_Toc35440552)

[18.1 低功耗简介 140](#_Toc35440553)

[18.2 低功耗 Related API 140](#_Toc35440554)

[18.2.1 进入低功耗模式 140](#_Toc35440555)

[18.2.2 deep\_sleep 模式 140](#_Toc35440556)

[18.3 低功耗示例代码 141](#_Toc35440557)

[18.3.1 关键说明 141](#_Toc35440558)

[18.3.2 示例代码 141](#_Toc35440559)

[18.4 操作说明 144](#_Toc35440560)

[18.4.1 连接万用表 144](#_Toc35440561)

[18.4.2 运行现象 145](#_Toc35440562)

[19 混音 146](#_Toc35440563)

[19.1 混音简介 146](#_Toc35440564)

[19.2 混音 Related API 146](#_Toc35440565)

[19.2.1 混音初始化 146](#_Toc35440566)

[19.2.2 暂停背景音播放 146](#_Toc35440567)

[19.2.3 重新播放背景音 146](#_Toc35440568)

[19.3 混音示例代码 147](#_Toc35440569)

[19.3.1 关键说明 147](#_Toc35440570)

[19.3.2 示例代码 147](#_Toc35440571)

[19.4 操作说明 148](#_Toc35440572)

[19.4.1 打开配置 148](#_Toc35440573)

[19.4.2 运行现象 148](#_Toc35440574)

[20 Airkiss 配网 149](#_Toc35440575)

[20.1 Airkiss简介 149](#_Toc35440576)

[20.2 Airkiss Related API 149](#_Toc35440577)

[20.2.1 开始airkiss 149](#_Toc35440578)

[20.2.2 获取airkiss状态 149](#_Toc35440579)

[20.2.3 获取airkiss解码结果 149](#_Toc35440580)

[20.3 Airkiss示例代码 150](#_Toc35440581)

[20.3.1 关键说明 150](#_Toc35440582)

[20.3.2 示例代码 150](#_Toc35440583)

[20.4 操作说明 152](#_Toc35440584)

[20.4.1 扫描微信airkiss配网二维码或下载Airkiss调试工具 152](#_Toc35440585)

[32.4.2 运行现象 152](#_Toc35440586)

[20.5 注意事项 154](#_Toc35440587)

[21 声波配网 155](#_Toc35440588)

[21.1 声波配网简介 155](#_Toc35440589)

[21.2 声波配网 Related API 155](#_Toc35440590)

[21.2.1 声波配网开始 155](#_Toc35440591)

[21.2.2 用户提前终止声波配网 156](#_Toc35440592)

[21.2.3 获取版本号 156](#_Toc35440593)

[21.3 声波配网示例代码 156](#_Toc35440594)

[21.4 操作说明 161](#_Toc35440595)

[21.4.1 打开配置 161](#_Toc35440596)

[21.4.2 运行现象 161](#_Toc35440597)

[22 Vad 164](#_Toc35440598)

[22.1 Vad自动语音检测简介 164](#_Toc35440599)

[22.2 Vad Related API 164](#_Toc35440600)

[22.2.1 进入vad检测模式 164](#_Toc35440601)

[22.2.2 获取帧的长度 164](#_Toc35440602)

[22.2.3 vad入口函数 164](#_Toc35440603)

[22.2.4 关闭vad 165](#_Toc35440604)

[22.3 Vad示例代码 165](#_Toc35440605)

[22.4 操作说明 168](#_Toc35440606)

[22.4.1 打开配置 168](#_Toc35440607)

[22.4.2 运行现象 168](#_Toc35440608)

[23 AMR编码器 170](#_Toc35440609)

[23.1 AMR编码器简介 170](#_Toc35440610)

[23.2 AMR编码器 Related API 170](#_Toc35440611)

[23.2.1 AMR-NB编码器初始化 170](#_Toc35440612)

[23.2.2 AMR-NB编码 170](#_Toc35440613)

[23.2.3 释放AMR-NB编码 171](#_Toc35440614)

[23.3 AMR编码器示例代码 171](#_Toc35440615)

[23.3.1 关键说明 171](#_Toc35440616)

[23.3.2 示例代码 171](#_Toc35440617)

[23.4 操作说明 184](#_Toc35440618)

[23.4.1 下载AMR Player工具 184](#_Toc35440619)

[23.4.2 网络调试助手设置 184](#_Toc35440620)

[23.4.3 打开配置 184](#_Toc35440621)

[23.4.4 运行现象 184](#_Toc35440622)

[24 Opus编码器 185](#_Toc35440623)

[24.1 Opus编码器简介 185](#_Toc35440624)

[24.2 Opus编码器 Related API 185](#_Toc35440625)

[24.2.1 创建opus编码器 185](#_Toc35440626)

[24.2.2 返回opus编码器所需内存的大小 185](#_Toc35440627)

[24.2.3 修改opus编码器的复杂度 186](#_Toc35440628)

[24.2.4 获取opus编码器的比特率 186](#_Toc35440629)

[24.2.5 获取opus编码器的最终状态 186](#_Toc35440630)

[24.2.6 opus编码 186](#_Toc35440631)

[24.2.7 释放opus编码器对象 187](#_Toc35440632)

[24.3 Opus编码器示例代码 187](#_Toc35440633)

[24.3.1 关键说明 187](#_Toc35440634)

[24.3.2 示例代码 187](#_Toc35440635)

[24.4 操作说明 199](#_Toc35440636)

[24.4.1 下载Cool Edit Pro工具 199](#_Toc35440637)

[24.4.2 网络调试助手设置 200](#_Toc35440638)

[24.4.3 打开配置 200](#_Toc35440639)

[24.4.4 运行现象 200](#_Toc35440640)

[25 EasyFlash 201](#_Toc35440641)

[25.1 EasyFlash简介 201](#_Toc35440642)

[25.2 EasyFlash Related API 201](#_Toc35440643)

[25.2.1 easyflash初始化 201](#_Toc35440644)

[25.2.2 获得easyflash环境变量 201](#_Toc35440645)

[25.2.3 将数据写入到环境变量中 201](#_Toc35440646)

[25.2.4 保存数据到flash 202](#_Toc35440647)

[25.3 EasyFlash示例代码 202](#_Toc35440648)

[25.3.1 关键说明 202](#_Toc35440649)

[25.3.2 示例代码 202](#_Toc35440650)

[25.4 操作说明 205](#_Toc35440651)

[25.4.1 打开配置 205](#_Toc35440652)

[25.4.2 运行现象 206](#_Toc35440653)

[26 Voice Changer 207](#_Toc35440654)

[26.1 Voice Changer简介 207](#_Toc35440655)

[26.2 Voice Changer Related API 207](#_Toc35440656)

[26.2.1 voice changer初始化 207](#_Toc35440657)

[26.2.2 退出voice changer 207](#_Toc35440658)

[26.2.3 开始voice changer 207](#_Toc35440659)

[26.2.4 停止voice changer 208](#_Toc35440660)

[26.2.5 设置voice changer变声功能标志 208](#_Toc35440661)

[26.2.6 voice changer获取mic数据 208](#_Toc35440662)

[26.2.7 设置消耗数据的长度 208](#_Toc35440663)

[26.2.8 处理数据 208](#_Toc35440664)

[26.3 Voice Changer示例代码 209](#_Toc35440665)

[26.3.1 关键说明 209](#_Toc35440666)

[26.3.2 示例代码 209](#_Toc35440667)

[26.4 操作说明 217](#_Toc35440668)

[26.4.1 运行现象 217](#_Toc35440669)

[27 图像传输 218](#_Toc35440670)

[27.1 图像传输简介 218](#_Toc35440671)

[27.2 图像传输 Related API 218](#_Toc35440672)

[27.2.1 打开video\_transfer 219](#_Toc35440673)

[27.2.2 关闭video\_transfer 219](#_Toc35440674)

[27.2.3 设置摄像头的参数 219](#_Toc35440675)

[27.2.4 打开获取jpeg帧的功能 220](#_Toc35440676)

[27.2.5 关闭获取jpeg帧的功能 220](#_Toc35440677)

[27.2.6 获取jpeg帧的数据 220](#_Toc35440678)

[27.3 图像传输的示例代码 220](#_Toc35440679)

[27.3.1 关键说明 220](#_Toc35440680)

[27.3.2 示例代码 221](#_Toc35440681)

[27.4 操作说明 225](#_Toc35440682)

[27.4.1 下载PC调试工具 226](#_Toc35440683)

[27.4.2 启动softap 226](#_Toc35440684)

[27.4.3 UDP传输测试 226](#_Toc35440685)

[27.4.4 TCP传输测试 227](#_Toc35440686)

[27.4.5 web camera 浏览器实时显示视频 227](#_Toc35440687)

[28 Qspi Dcache模式 229](#_Toc35440688)

[28.1 Qspi Dcache简介 229](#_Toc35440689)

[28.2 Qspi Dcache Related API 229](#_Toc35440690)

[28.2.1 初始化qspi为dcache模式 229](#_Toc35440691)

[28.2.2 启动qspi功能 229](#_Toc35440692)

[28.2.3 停止qspi功能 230](#_Toc35440693)

[28.3 Qspi Dcache示例代码 230](#_Toc35440694)

[28.4 操作说明 237](#_Toc35440695)

[28.4.1 运行现象 237](#_Toc35440696)

[28.5 注意事项 237](#_Toc35440697)

[29 BLE 238](#_Toc35440698)

[29.1 BLE简介 238](#_Toc35440699)

[29.2 BLE Related API（通用） 238](#_Toc35440700)

[29.2.1 启动ble协议栈 238](#_Toc35440701)

[29.2.2 设置write callback 238](#_Toc35440702)

[29.2.3 设置read callback 238](#_Toc35440703)

[29.2.4 设置event callback 238](#_Toc35440704)

[29.3 BLE Related API（slave） 239](#_Toc35440705)

[29.3.1 开始广播 239](#_Toc35440706)

[29.3.2 关闭广播 239](#_Toc35440707)

[29.3.3 发送数据 239](#_Toc35440708)

[29.3.4 断开连接 240](#_Toc35440709)

[29.4 BLE结构体说明 240](#_Toc35440710)

[29.4.1 广播参数 240](#_Toc35440711)

[29.5 BLE示例代码 240](#_Toc35440712)

[29.5.1 关键说明 240](#_Toc35440713)

[29.5.2 示例代码 241](#_Toc35440714)

[29.6 操作说明 252](#_Toc35440715)

[29.6.1 数据交互 252](#_Toc35440716)

[30 USB充电模式 255](#_Toc35440717)

[30.1 USB充电模式简介 255](#_Toc35440718)

[30.2 USB充电模式 Related API 255](#_Toc35440719)

[30.2.1 vbat adc 校准第一步 255](#_Toc35440720)

[30.2.2 充电参数校准 255](#_Toc35440721)

[30.2.3 开始充电模式 256](#_Toc35440722)

[30.2.4 停止充电模式 256](#_Toc35440723)

[30.3 示例代码 256](#_Toc35440724)

[30.3.1 关键说明 256](#_Toc35440725)

[30.4 操作说明 257](#_Toc35440726)

[30.4.1 运行现象 257](#_Toc35440727)

[31 EZ\_CONFIG配网 259](#_Toc35440728)

[31.1 EZ\_CONFIG简介 259](#_Toc35440729)

[31.2 EZ\_CONFIG Related API 259](#_Toc35440730)

[31.2.1 开始EZ\_CONFIG 259](#_Toc35440731)

[31.2.2 获取EZ\_CONFIG状态 259](#_Toc35440732)

[31.2.3 获取EZ\_CONFIG解码结果 260](#_Toc35440733)

[31.3 使用说明 260](#_Toc35440734)

[32 BLE 配网 261](#_Toc35440735)

[32.1 BLE 配网简介 261](#_Toc35440736)

[32.2 BLE 配网 Related API 261](#_Toc35440737)

[32.3 使用说明 261](#_Toc35440738)

[33 AP 配网 262](#_Toc35440739)

[33.1 AP 配网简介 262](#_Toc35440740)

[33.2 AP 配网 Related A 262](#_Toc35440741)

[33.3 使用说明 262](#_Toc35440742)

# 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 <rtdef.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 micphone(mic1)  1: two micphones( mic1 && mic2 ) -- use two micphone data  2: two micphones( mic1 && mic2 ) -- only use mic1 data  3: two micphones( 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 errorr\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 Realated 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\_RRESS\_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 |
| **rt\_i2c\_bus\_device\_ops\* ops** | i2c操作 |
| **rt\_uint16\_t addr** | 从设备地址 |
| **rt\_uint16\_t flags** | 读写标志位 |
| **rt\_mutex lock** | 锁定结构体 |
| **rt\_uint32\_t timeput** | 超时标志 |
| **rt\_uint32\_t retries** | 重复次数 |
| **void \*pirv** | 设备私有数据 |

**rt\_i2c\_msg：**

|  |  |
| --- | --- |
| **rt\_uint16\_t addr** | 地址 |
| **rt\_uint16\_t flags** | 读写标志 |
| **rt\_uint16\_t len** | buffer 长度 |
| **rt\_uint8\_t \*buf** | 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**

|  |  |
| --- | --- |
| **#define RT\_USING\_I2C** | 使用MCU的I2C设备 |
| **#define RT\_USING\_I2C\_BITOPS** | MCU的GPIO模拟I2C总线 |
| **#define BEKEN\_USING\_IIC** | 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 justifie d、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：

1. 数据交换长度可配，常以byte为单位，MSB先发，LSB后发；
2. 支持主机模式，时钟可配置，最大速率30MHZ；
3. 支持从机模式，能承受的最大速率10MHZ；
4. 时针极性（CPOL）和时针相位（CPHA）可配置；
5. 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()** | 通过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!\r\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\r\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\r\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\_recv(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("gspi\_test master/slave tx/rx rate len\r\n");  }  }  MSH\_CMD\_EXPORT(gspi\_test, gspi\_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。

gspi\_test master tx 1000000 128, 1M速率发送128字节数据

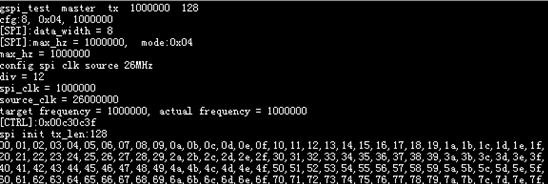


图7.4.1

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

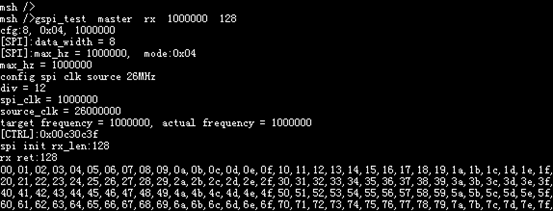


图7.4.2

## 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：

1. 使用SPI四线主模式；
2. 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** |
| **rt\_device\_control()** | 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** |
| **dev** | SPI FLASH设备的指针 |
| **cmd** | 设备定义的操作命令，具体见下面说明。 |
| **\*arg** | 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:**

|  |  |
| --- | --- |
| **#define BK\_SPI\_FLASH\_ERASE\_CMD** | 擦除命令 |
| **#define BK\_SPI\_FLASH\_PROTECT\_CMD** | falsh加写保护 |
| **#define BK\_SPI\_FLASH\_UNPROTECT\_CMD** | 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\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：

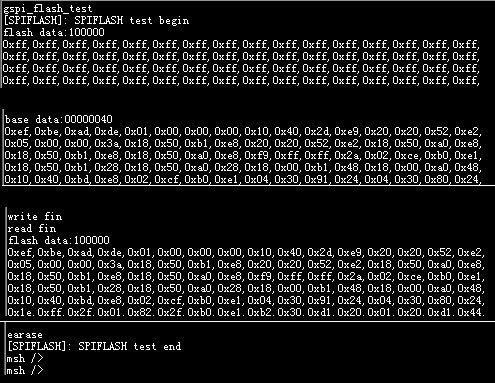


图8.4.1-1

## 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：

1. 使用SPI四线主模式；
2. 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 gspi psram need 'CFG\_USE\_SPI\_MASTER' and 'CFG\_USE\_SPI\_MST\_PSRAM'"  #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：

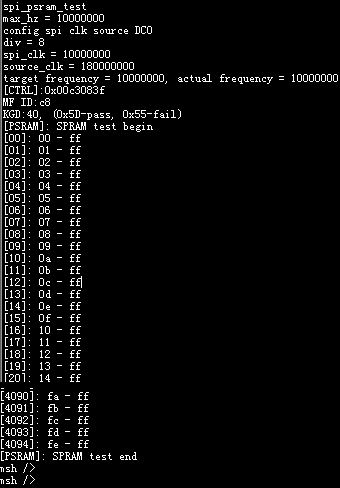


图9.4.2-1

## 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：

1. 支持四线全双工、三线半双工模块；
2. 支持MSB、LSB可配置；
3. 支持DMA传输；
4. 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!\r\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 rx 100，接收100字节数据

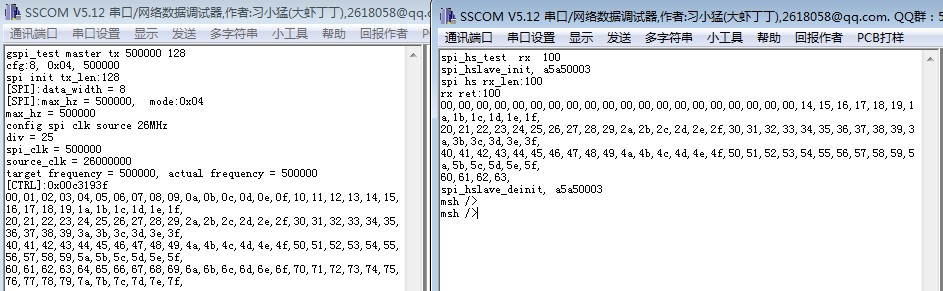


图10.4.2-1

## 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。

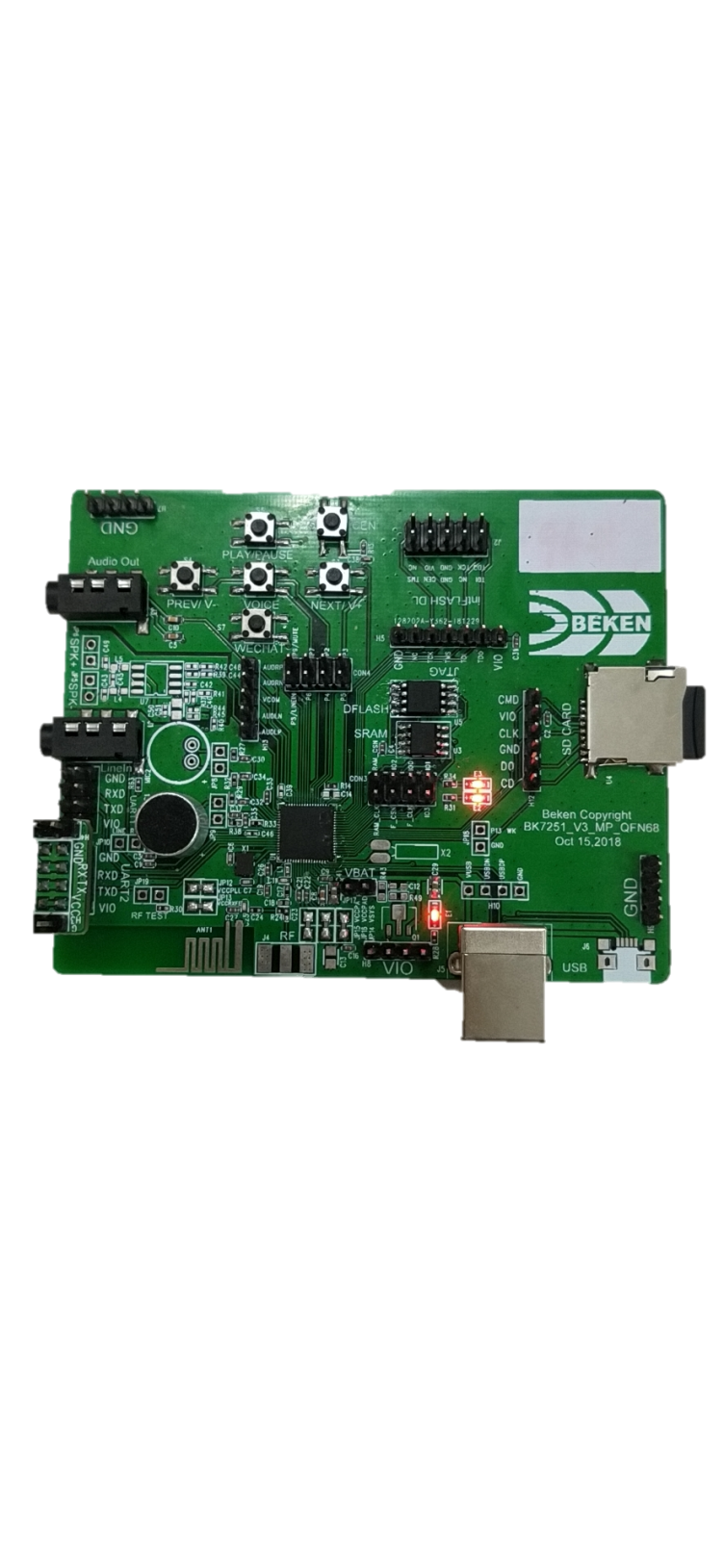


图11.4.1-1

### 11.4.2 运行现象

按下S3按键，LED2熄灭：

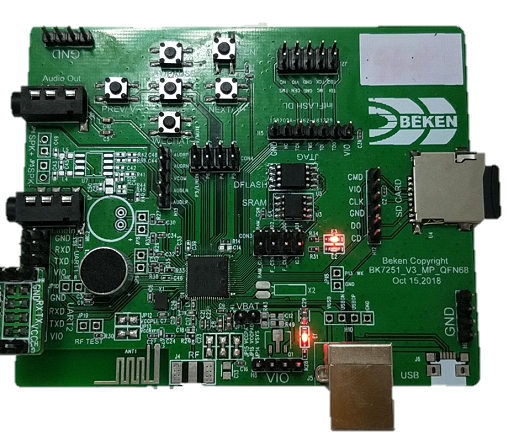


图11.4.1-2

# 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，否则通信将无法正常完成。

## 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** |
| **rt\_device\_control()** | 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** |
| **rt\_uint32\_t baud\_rate** | 波特率设置：一般为115200 |
| **rt\_uint32\_t data\_bits** | 数据位：一般为8bit |
| **rt\_uint32\_t stop\_bits** | 停止位:一般为1 |
| **rt\_uint32\_t parity** | 奇偶校验位：无校验位 |
| **rt\_uint32\_t bit\_order** | 大小端：一般为小端 |
| **rt\_uint32\_t invert** | 模式转化：不转换 |
| **rt\_uint32\_t bufsz** | 接收buffer大小 |
| **rt\_uint32\_t reserved** | 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>  #ifdef 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：

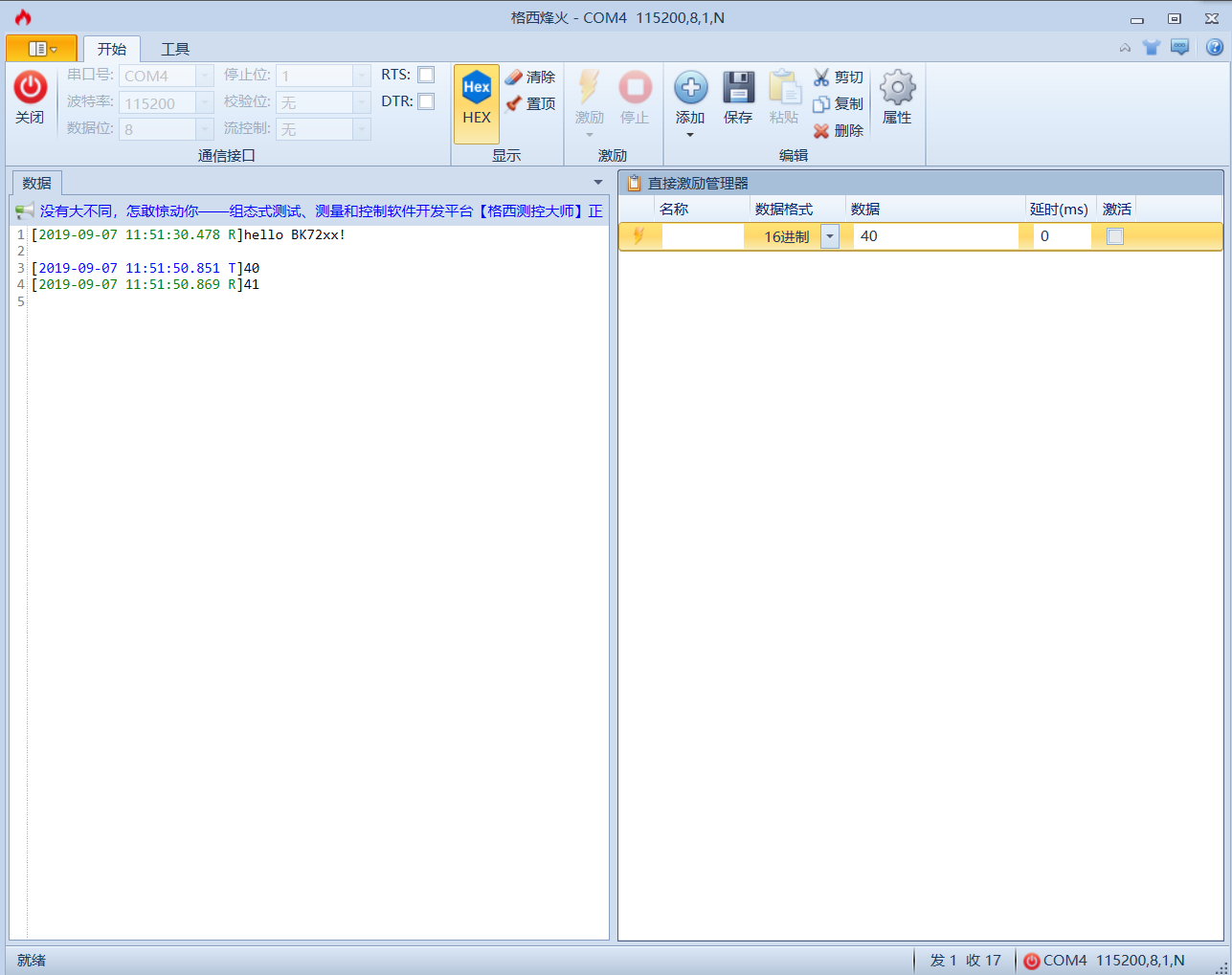


图12.4.2-1

## 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(PLAYER\_ENABLE\_NET\_STREAM)  #include "netstream.h"  #include "stream\_pipe.h"  #include "netstream\_buffer.h"  #endif  #if defined(PLAYER\_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(PLAYER\_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);  }  #if defined(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：

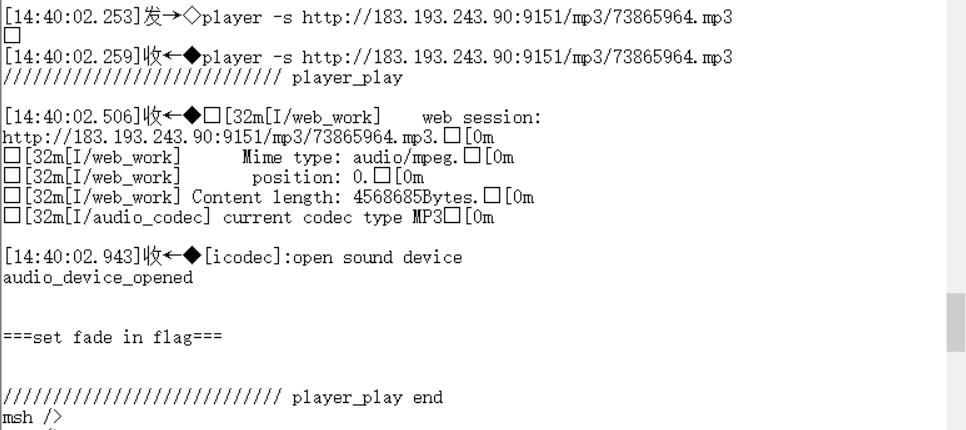


图13.4.1-1

# 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, /\*未任何连接状态\*/  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 "\r\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\r\n",sta\_ip\_is\_start(),uap\_ip\_is\_start());  #else  bk\_printf("sta: %d, softap: %d, b/g\r\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\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;  }  }  /\*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\_recv\_ret;  airkiss\_recv\_ret = airkiss\_recv(ak\_contex, 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 mlan (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：



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：

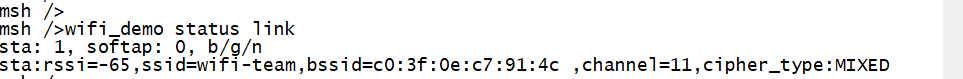


图14.4.3-2

### 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：

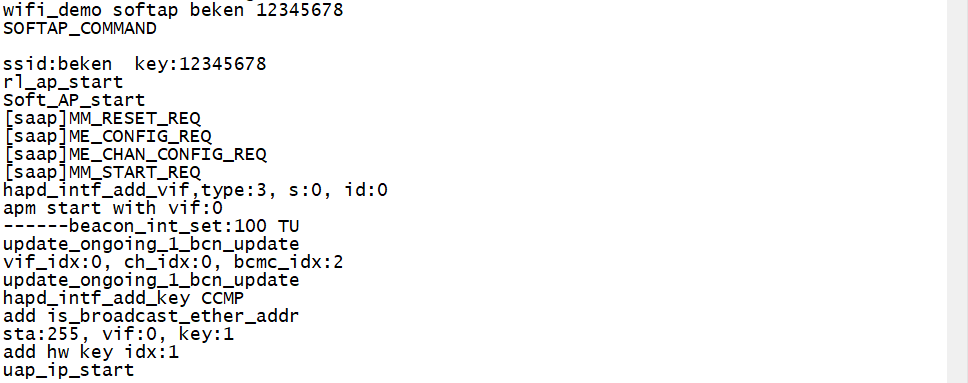


图14.4.4-1

### 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：



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：

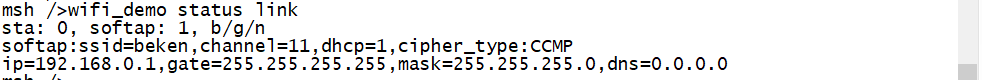


图14.4.5-2

### 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：



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：



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\rttos\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 |

### 15.2.13 销毁一个互斥锁

|  |
| --- |
| **OSStatus bk\_rtos\_deinit\_mutex( beken\_mutex\_t\* mutex );** |

|  |  |
| --- | --- |
| **参数** | **描述** |
| **mutex** | 指向要销毁的互斥锁的句柄的指针。 |
| **返回** | kNoErr：成功；其他：失败 |

### 15.2.14 初始化一个消息队列

|  |
| --- |
| **OSStatus bk\_rtos\_init\_queue( beken\_queue\_t\* queue,**  **const char\* name,**  **uint32\_t message\_size,**  **uint32\_t number\_of\_messages );** |

|  |  |
| --- | --- |
| **参数** | **描述** |
| **queue** | 指向要创建的消息队列的句柄的指针。 |
| **name** | 队列的名字 |
| **message\_size** | 将要进入队列对象的最大字节数 |
| **number\_of\_messages** | 队列的深度，即队列中对象的最大数量 |
| **返回** | kNoErr：成功；其他：失败 |

### 15.2.15 将一个数据对象推入消息队列

|  |
| --- |
| **OSStatus bk\_rtos\_push\_to\_queue( beken\_queue\_t\* queue,**  **void\* message,**  **uint32\_t timeout\_ms );** |

|  |  |
| --- | --- |
| **参数** | **描述** |
| **queue** | 指向要推入数据对象的消息队列的句柄的指针。 |
| **message** | 推入队列的对象，对象大小在队列初始化rtos\_init\_queue中已指定。 |
| **timeout\_ms** | 返回前等待的毫秒数。 |
| **返回** | kNoErr：成功；其他：失败 |

### 15.2.16 从消息队列中取出一个数据对象

|  |
| --- |
| **OSStatus bk\_rtos\_pop\_from\_queue( beken\_queue\_t\* queue,**  **void\* message,**  **uint32\_t timeout\_ms );** |

|  |  |
| --- | --- |
| **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)** | /\*执行成功\*/ |
| **#define RTOS\_FAILURE (0)** | /\*执行失败\*/ |

RTOS优先级配置：

|  |  |
| --- | --- |
| **#define BEKEN\_DEFAULT\_WORKER\_PRIORITY (6)** | /\*默认优先级为6\*/ |
| **#define BEKEN\_APPLICATION\_PRIORITY (7)** | /\*应用优先级为7\*/ |

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** |
| **fal\_init()** | fal初始化 |
| **http\_ota\_fw\_download()** | 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** |
| **void** | 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** |
| **url** | 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，加密密匙与加密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，

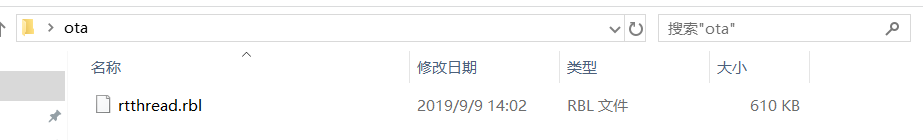


图16.4.1-2



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

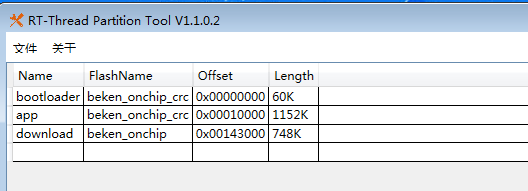
打开rt\_partition\_tool软件，加载原始的bootloader.bin，然后导入分区表partition\_audio\_2M\_sd.json，最后把分区表保存到bootloader.bin中，操作完成后该bootloader.bin即可和应用代码一起通过打包工具beken\_packager生成最终的bin文件。

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能完整的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 && !strncmp(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, containning 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 (!strncmp(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");  }  }  FINSH\_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：

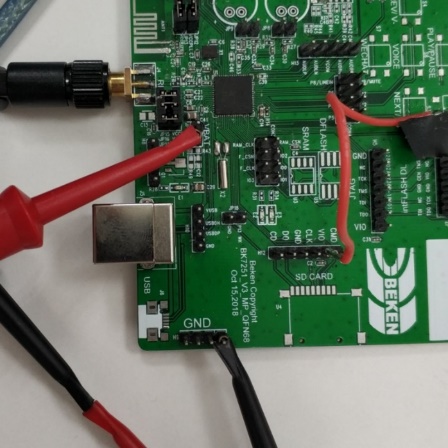
****

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\fuction\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** |
| **mixer\_init()** | 混音初始化 |
| **mixer\_pause()** | 暂停背景音乐，录音的时候必须暂停 |
| **mixer\_replay()** | 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** |
| **void** | 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** |
| **void** | 无 |
| **返回** | void |

### 19.2.3 重新播放背景音

|  |
| --- |
| **void mixer\_replay(void)；** |

|  |  |
| --- | --- |
| **参数** | **描述** |
| **void** | 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

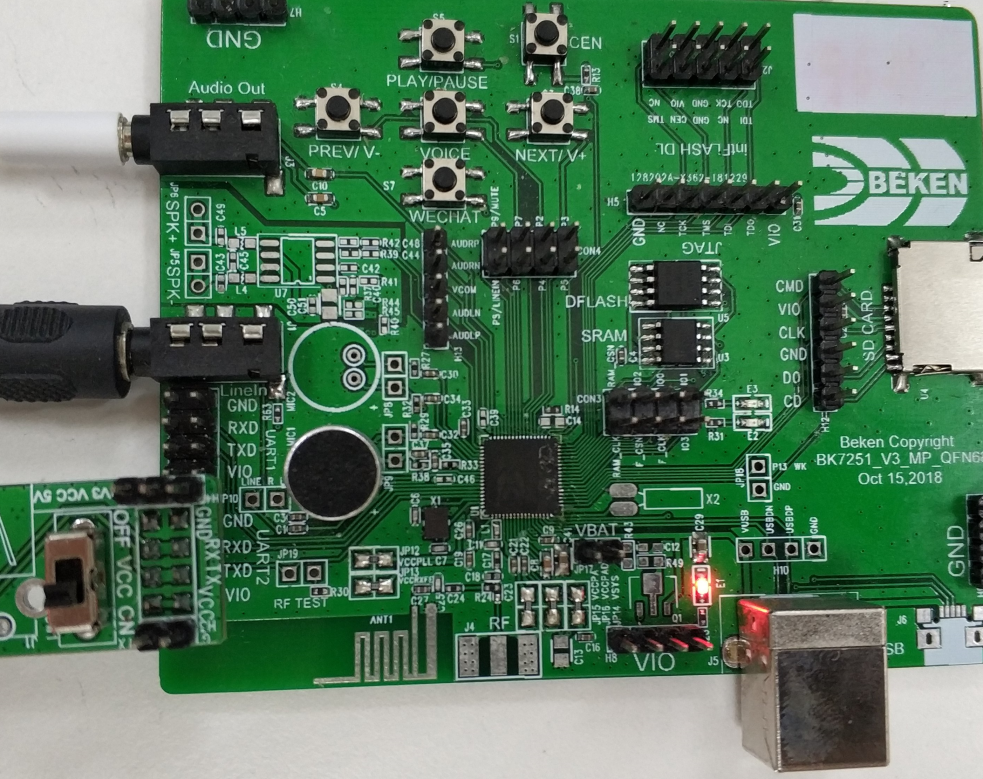
### 19.4.1 打开配置

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：



audio out

line in

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的一种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** |
| **airkiss()** | 开始airkiss |
| **airkiss\_get\_status()** | 获取aikiss状态 |
| **airkiss\_get\_result()** | 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** |
| **void** | 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** |
| **void** | 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** |
| **airkiss\_context\_t\* context,** | 为airkiss库分配的内存 |
| **airkiss\_result\_t \*result** | 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\_ariksss);  #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](https://iot.weixin.qq.com/wiki/new/index.html?page=6-1)

进入下载页面后，下载下图所示工具：



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：

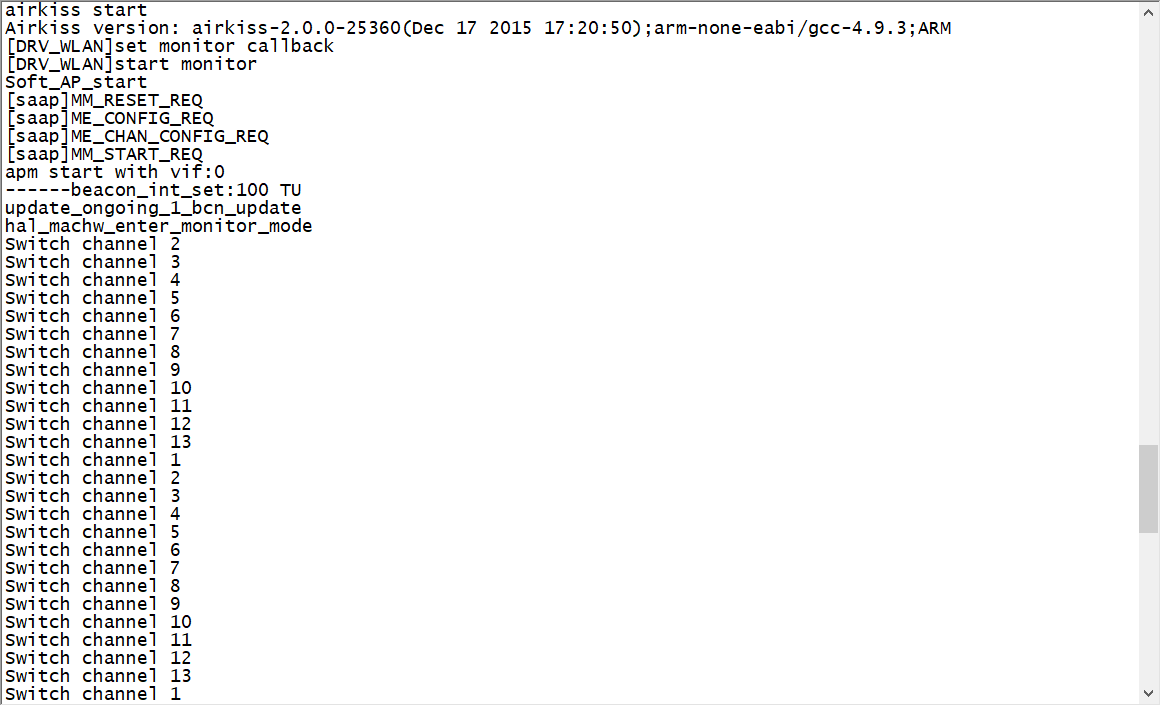


图20.4.2-1

**• 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：



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)  {  #ifdef 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：

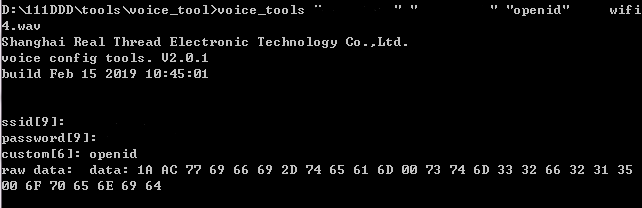


图21.4.2-1

**• 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：

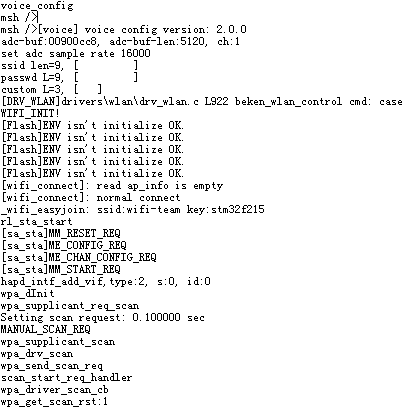


图21.4.2-2

网络连接成功log如下：

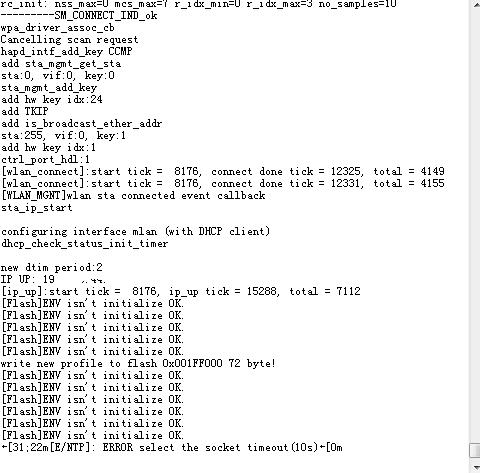


图21.4.2-3

# 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** |
| **amrnb\_encoder\_init()** | amr编码初始化 |
| **amrnb\_encoder\_encode()** | amr编码 |
| **amrnb\_encoder\_deinit()** | 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** |
| **amrnb** | AMR-NB编码器的指针 |
| **dtx** | 0：连续传输数据  1：不连续传输数据 |
| **pmalloc** | malloc函数指针 |
| **pfree** | 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** |
| **amrnb** | AMR-NB编码器的指针 |
| **mode** | amr编码模式 |
| **in** | 输入的语音 |
| **out** | 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\_recv(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\r\n");  record\_help();  return;  }  rec\_encoder = rt\_calloc(1, sizeof(struct rec\_encoder\_manager));  if(NULL == rec\_encoder)  {  rt\_kprintf("rec\_enoder\_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](http://www.amrplayer.com)

### 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** |
| **opus\_encoder\_create()** | 创建opus编码 |
| **opus\_encoder\_get\_size()** | 返回编码器所需内存的大小 |
| **opus\_encoder\_set\_complexity()** | 修改编码器复杂度 |
| **opus\_encoder\_get\_bitrate()** | 获取编码器的比特率 |
| **opus\_encoder\_get\_final\_range()** | 获取编码器最终状态 |
| **opus\_encode()** | opus编码 |
| **opus\_encoder\_destroy()** | 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\_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\_recv(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\r\n");  record\_help();  return;  }  rec\_encoder = rt\_calloc(1, sizeof(struct rec\_encoder\_manager));  if(NULL == rec\_encoder)  {  rt\_kprintf("rec\_enoder\_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); |

## 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](http://cool-edit-pro.brothersoft.com/)

### 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** |
| **easyflash\_init()** | easyflash初始化 |
| **ef\_get\_env()** | 获得easyflash环境变量 |
| **ef\_set\_env()** | 写数据到easyflash中 |
| **ef\_save\_env()** | 保存数据到flash |

### 25.2.1 easyflash初始化

|  |
| --- |
| **EfErrCode easyflash\_init(void);** |

|  |  |
| --- | --- |
| **参数** | **描述** |
| **void** | 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** |
| **key** | 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** |
| **key** | 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

Vocie 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bufffer\*/  {  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);  #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

1. 有高速spi-slave接口，速度高达50Mbps，可以外接其他MCU摄像头；
2. 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。
3. 有硬件Jpeg压缩模块，目前支持最大分辨率600\*800；
4. 图像传输结构框图如下所示：

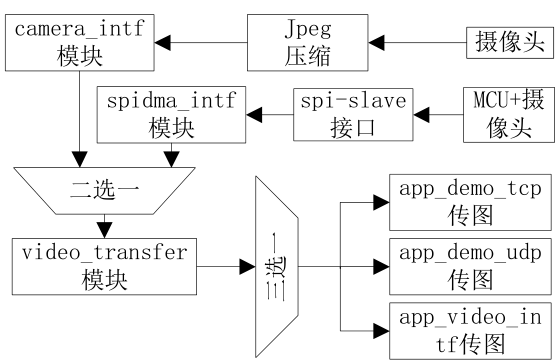


图27.1-1

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 |
| **video\_transfer\_set\_video\_param()** | 使用DCMI接口时，设置摄像头的参数 |
| **video\_buffer\_open()** | 打开获取jpeg帧数据功能 |
| **video\_buffer\_close()** | 关闭获取jpeg帧数据功能 |
| **video\_buffer\_read\_frame()** | 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** |
| **setup\_cfg** | 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 ：**

|  |  |
| --- | --- |
| **UINT32 send\_type** | 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。 |
| **send\_func** | 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。 |
| **start\_cb** | 模块打开spi或camera\_intf后，回调此函数，用于指示传图开始。 |
| **end\_cb** | 模块关闭spi或camera\_intf前，回调此函数，用于指示传图结束。 |
| **pkt\_header\_size** | 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。 |
| **add\_pkt\_header** | 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** |
| **void** | 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 seccond），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 |

|  |  |
| --- | --- |
| **#deine 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\_romote\_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\_CONECTED);  }  /\*指示停止传图 \*/  static void app\_demo\_udp\_app\_disconnected(void)  {  app\_demo\_softap\_send\_msg(DMSG\_APP\_DISCONECTED);  }  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：

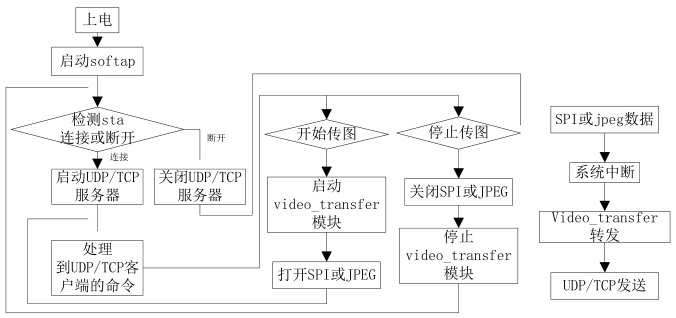


图27.4-1

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，这里时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。

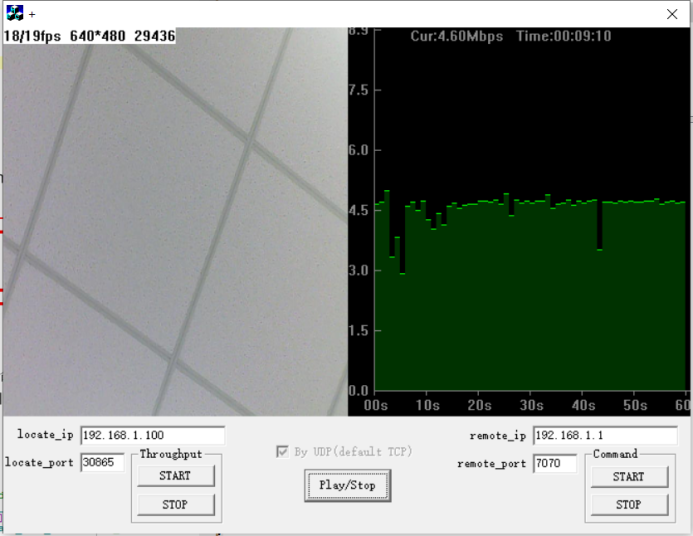


图27.4.3-1

### 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。

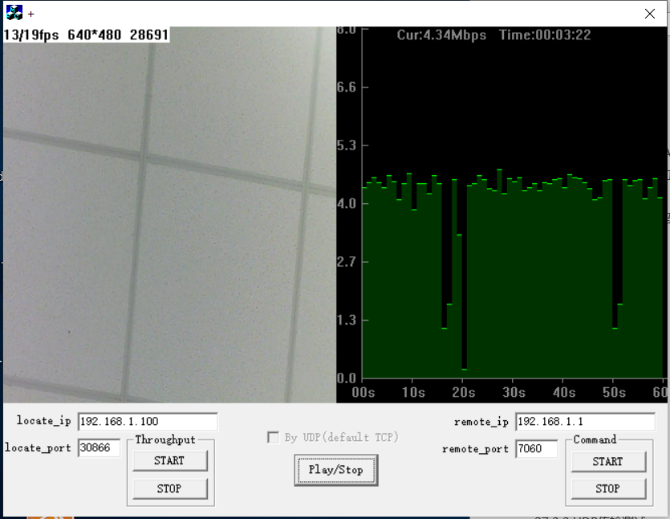


图27.4.4-1

### 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。

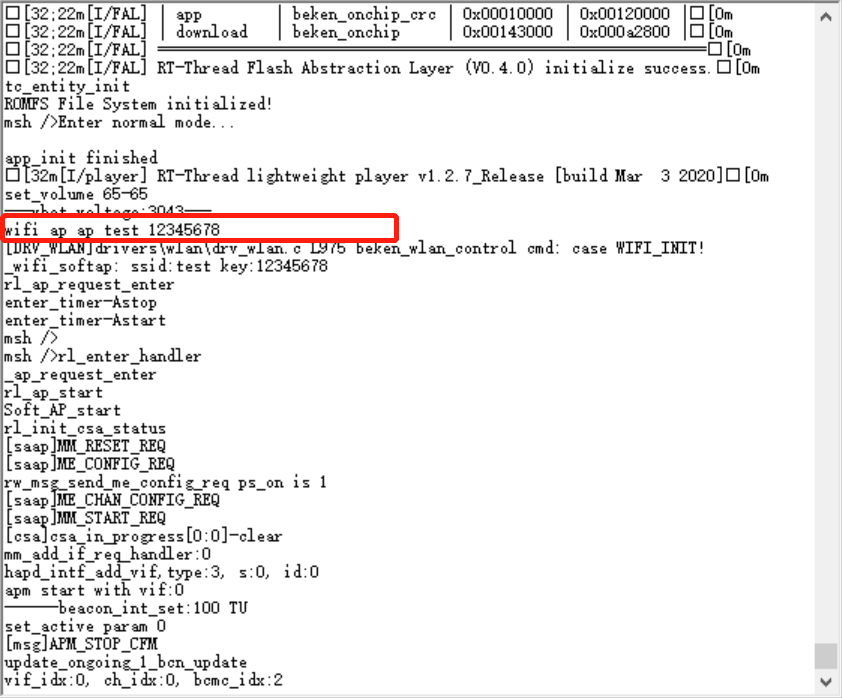
步骤三：电脑端打开摄像头，输入设备端ip,即可查看摄像头视频。

图27.4.5-1 开启AP

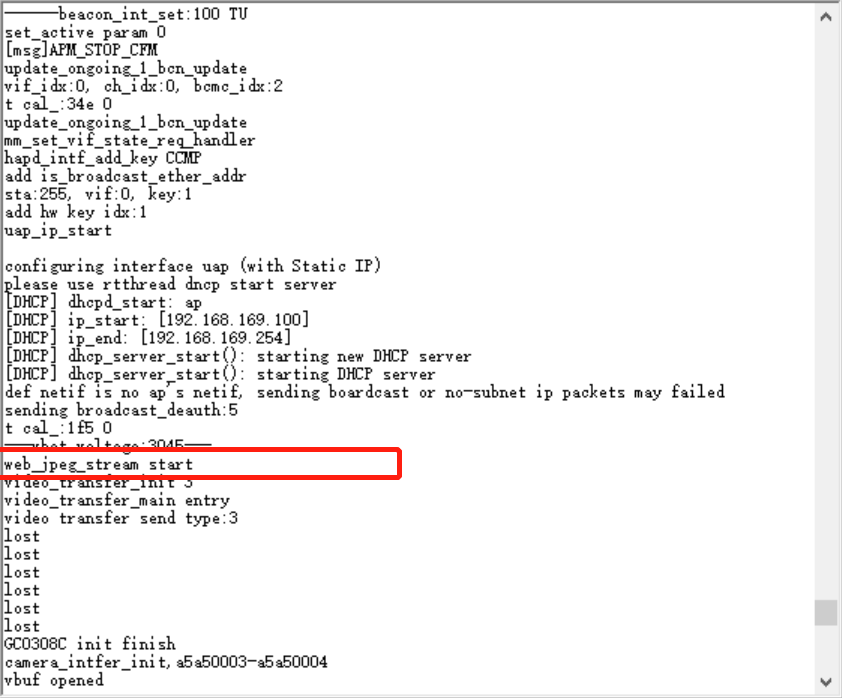


图27.4.5-2 开启web camera

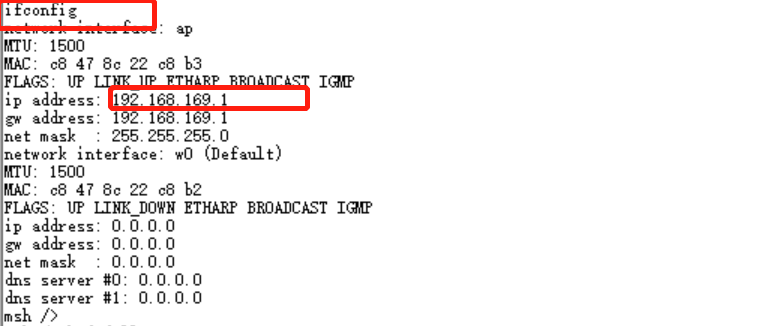


图27.4.5-3 查看设备ip

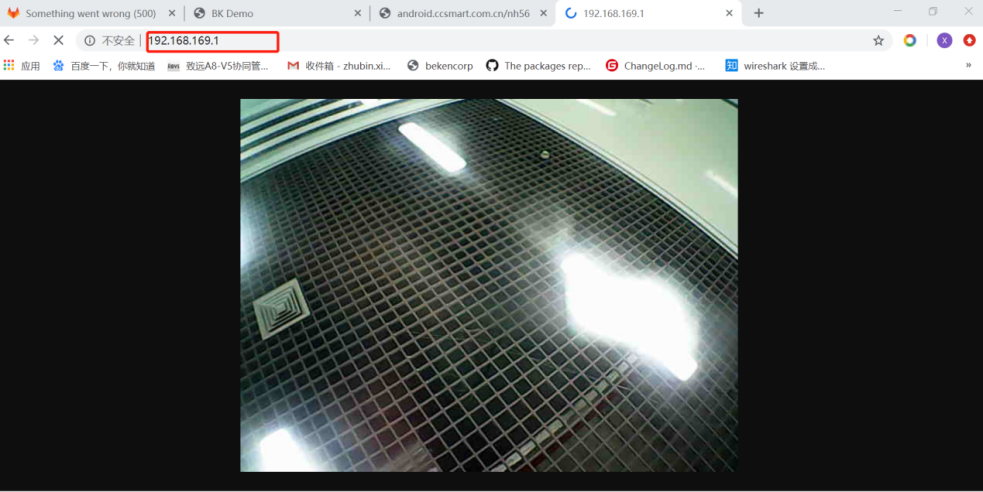


图27.4.5-4 浏览器查看视频

# 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");  }  }  FINSH\_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：

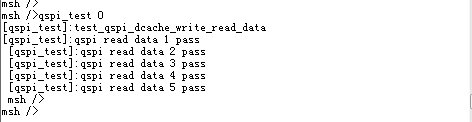


图28.4.1-1

## 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

|  |
| --- |
| **ble\_err\_t bk\_ble\_send\_ntf\_value(uint32\_t len, uint8\_t \*buf, uint16\_t prf\_id, uint16\_t att\_idx)** |

|  |  |
| --- | --- |
| **参数** | **描述** |
| **len** | 数据长度（byte） |
| **buf** | 数据指针 |
| **prf\_id** | Profile id |
| **att\_idx** | Attribute id |
| **返回** | ERR\_SUCCESS:成功  ERR\_PROFILE：失败 |

|  |
| --- |
| **ble\_err\_t bk\_ble\_send\_ind\_value(uint32\_t len, uint8\_t \*buf, uint16\_t prf\_id, uint16\_t att\_idx)** |

|  |  |
| --- | --- |
| **参数** | **描述** |
| **len** | 数据长度（byte） |
| **buf** | 数据指针 |
| **prf\_id** | Profile id |
| **att\_idx** | Attribute id |
| **返回** | ERR\_SUCCESS:成功  ERR\_PROFILE：失败 |

### 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 |

### 29.3.5 关闭BLE

|  |
| --- |
| **void ble\_stop(void)** |

|  |  |
| --- | --- |
| **参数** | **描述** |
| **无** | 关闭BLE协议栈 |
| **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\_attm\_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("UNKNOW 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**

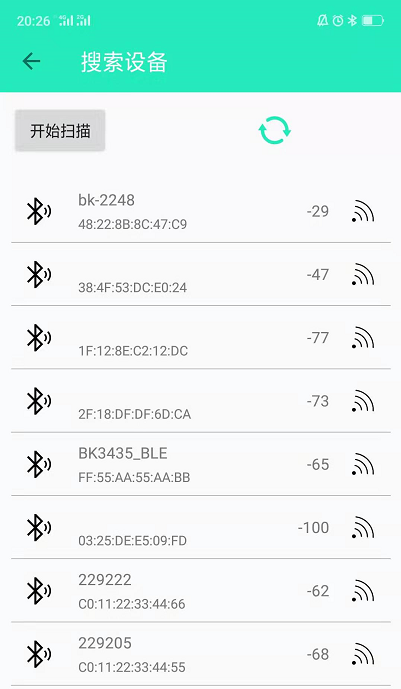
手机打开BLE调试APP，点击扫描所有设备并显示，点击bk-2248,连接BLE设备，结果如图29.6.1-3和图29.6.1-4所示。

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：

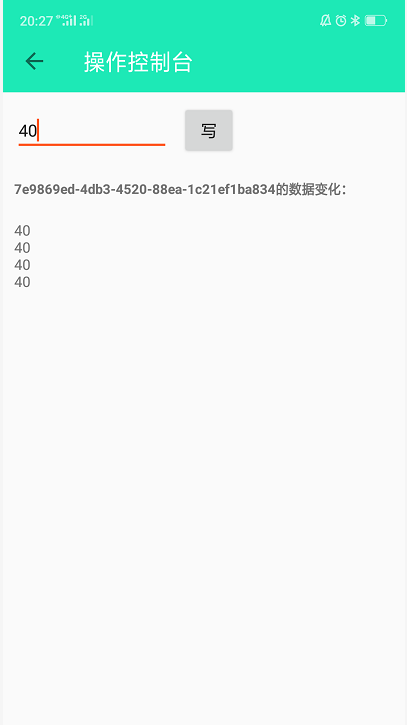


图29.6.1-5

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 |
| **nsigned 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。

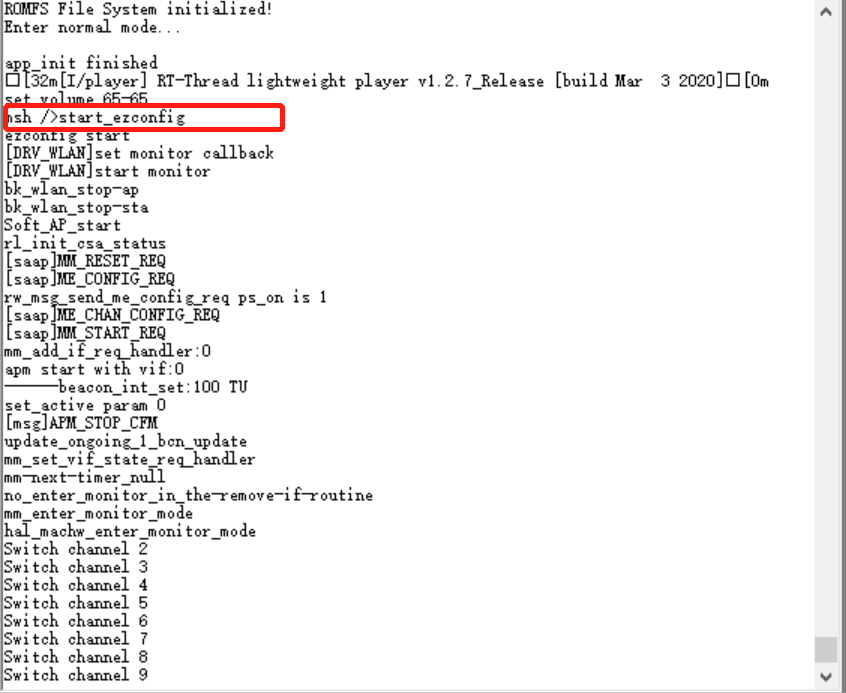
设备端开启ezconifg配网，log如图31.3-1所示:

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password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

扫描下方配网二维码或微信搜索小雅慧读公众号，点击更多进入蓝牙配网微信小程序，设备端发送串口命令开启BLE配网，log如图32.3-1所示:

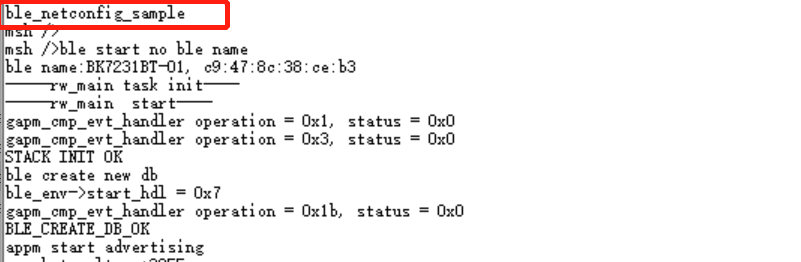




图32.3-1 图32.3-2 小雅公众号

# 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。

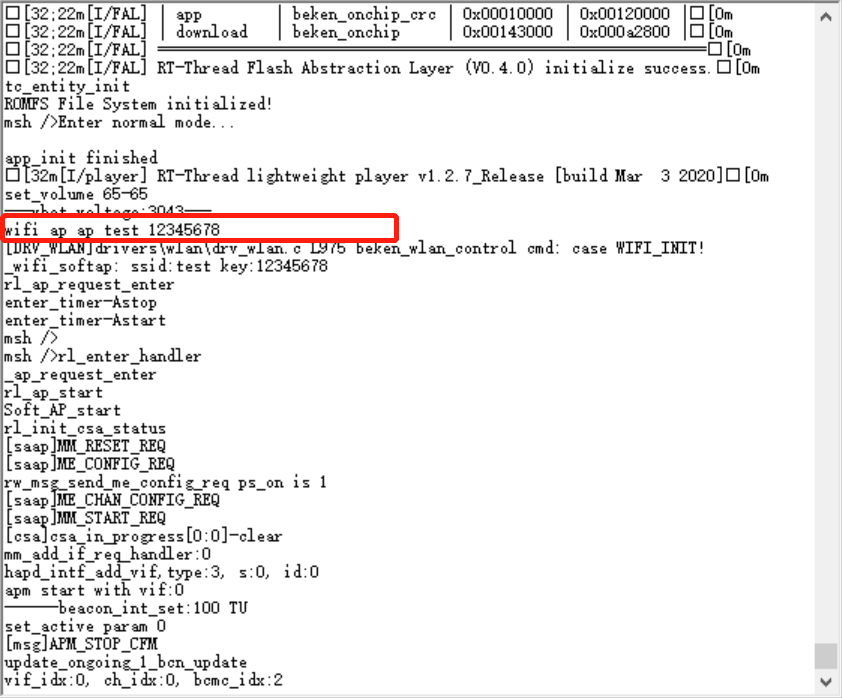


图33.3-1 开启AP

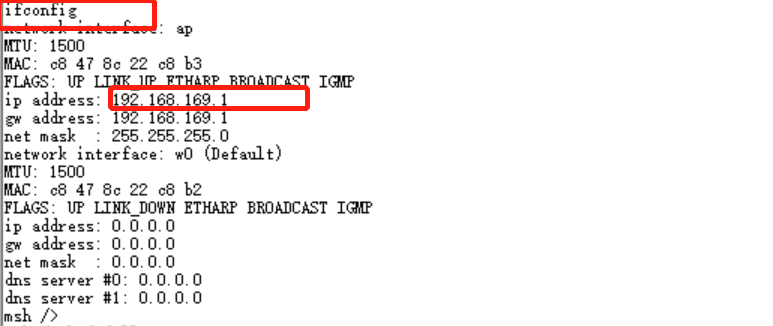


图33.3-2 查看设备ip

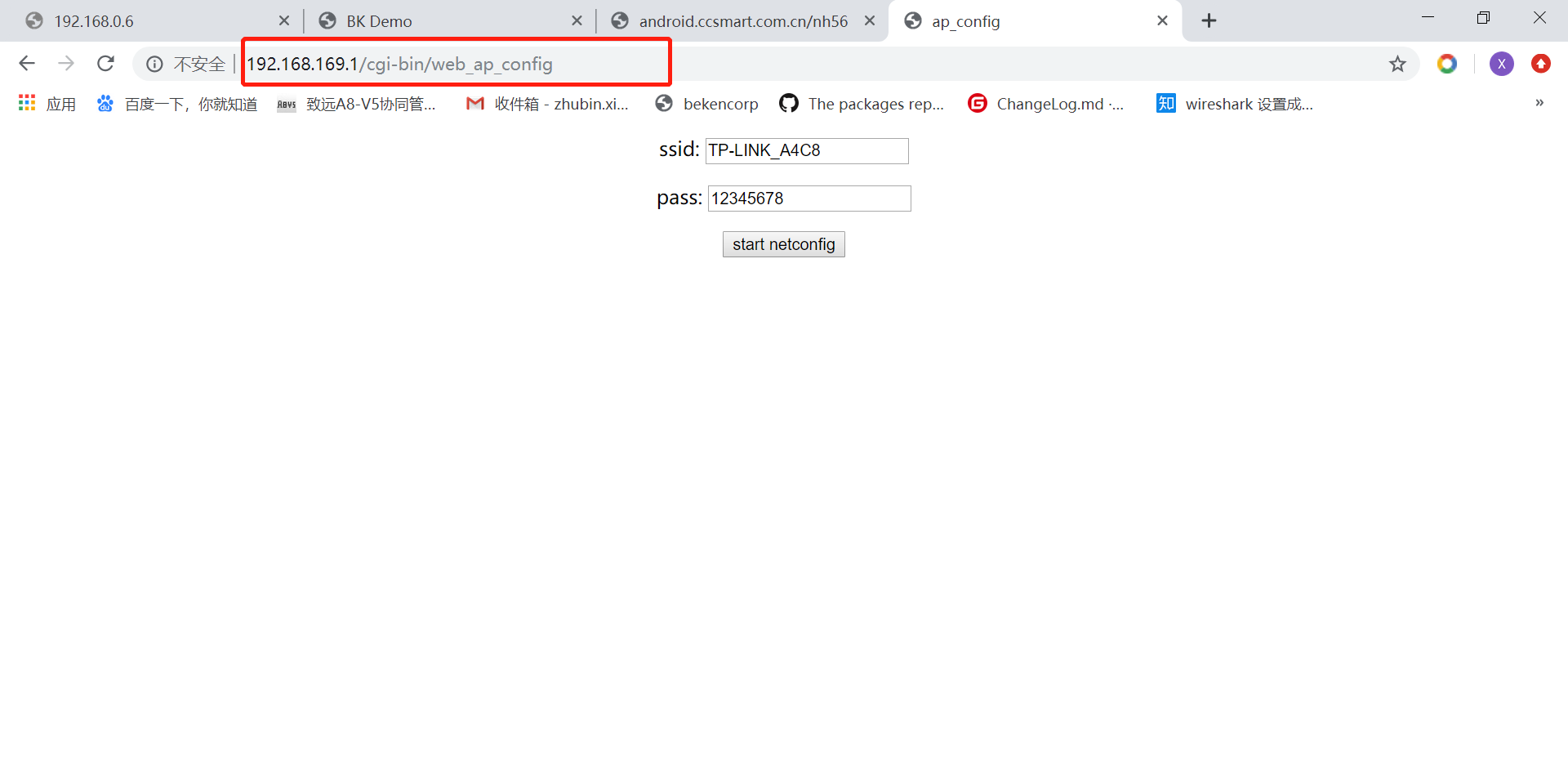


图33.3-3 浏览器打开链接

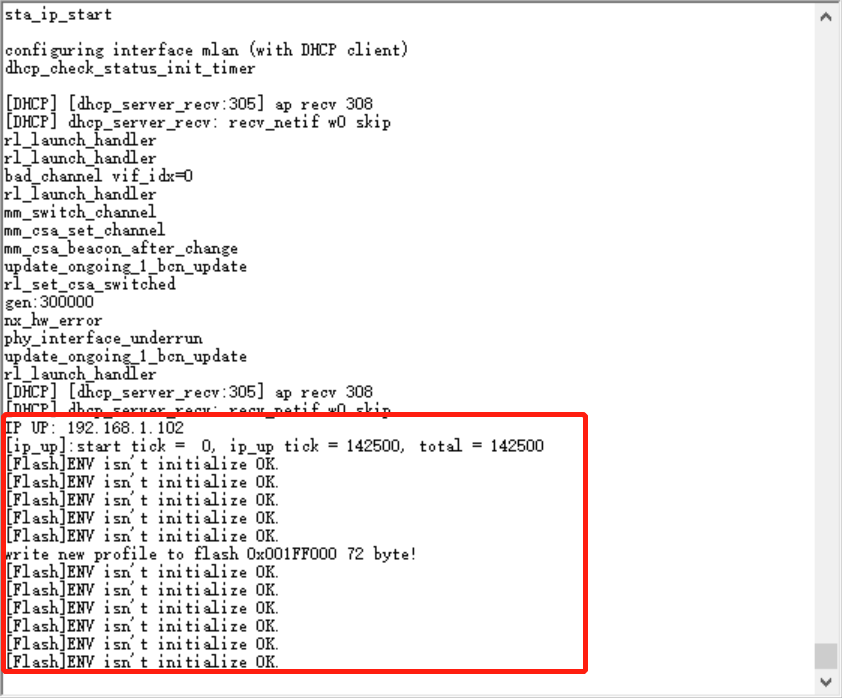


图33.3-4 设备配网成功