0 aic8800 sdk 用户指南

本文档为 aic8800 系列芯片 sdk 配套的软件开发指南
(适用芯片：AIC8800M/AIC8800A/AIC8800F/AIC8800MC/AIC8800FC/AIC8800M40B/AIC8800M80)

目录内容：

版本信息：

Date	Version	Notes
2021-01-08	1.1.00	初始版本
2021-01-15	1.1.01	增加bin文件烧录
2021-01-16	1.1.02	更新test case介绍
2021-01-22	1.1.03	更新bin文件烧录
2021-03-01	1.1.04	更新test case & wifi application介绍
2021-05-28	1.1.05	增加AIC8800A EVB & 问题定位章节
2021-10-28	1.1.06	更新modules & 增加BT相关介绍
2021-11-18	1.1.07	增加GPIO功能选择
2021-11-26	1.1.08	增加VCORE09配置方法
2021-12-07	1.1.09	增加芯片系统架构及内存外设mapping
2021-12-14	1.1.10	更新芯片系统架构图
2022-02-09	1.1.11	增加PWRKEY时序图
2022-04-19	1.1.12	修改BT/BLE应用层相关介绍
2023-02-06	1.1.13	更新源码编译与WiFi应用介绍
2023-03-14	1.2.00	增加AIC8800MC/FC系列芯片相关内容
2023-04-02	1.2.01	更新bin文件烧录
2023-07-27	1.2.02	增加AIC8800M40B系列芯片相关内容
2023-10-17	1.2.03	增加AIC8800M80系列芯片相关内容

1 sdk 概述

1.1 目录结构

sdk 结构如下所示：

```
aic8800-sdk/
├── applications
|   ├── blesmartconfigwifi
|   ├── command_common
|   ├── consolewifi
|   ├── csiwifi
|   ├── examplewifi
|   ├── fhostifwifi
|   ├── hostifwifi
|   ├── http_otawifi
|   ├── http_servwifi
|   ├── micwifi
|   ├── rawdatawifi
|   ├── tls_examplewifi
|   ├── uartwifi
|   └── whwifi
├── audio
|   ├── app
|   ├── asio
|   ├── bt
|   ├── common
|   ├── config
|   ├── device
|   ├── lib
|   ├── process
|   └── wifi
├── btdm
|   ├── ble
|   ├── bt
|   ├── config
|   └── lib
├── config
|   ├── aic8800
|   |   ├── target_btdm
|   |   ├── target_test
|   |   ├── target_wifi
|   |   └── target_wifi_fhostif
|   ├── aic8800mc
|   |   ├── target_ble
|   |   ├── target_test
|   |   ├── target_wifi
|   |   └── target_wifi_fhostif
|   └── aic8800m40
|       ├── target_ble
|       ├── target_test
|       ├── target_wifi
|       └── target_ble_wifi_fhostif
├── freertos
|   ├── config
|   ├── FreeRTOS
```

```
|   ├── rtos_al
|   └── rtos_port
├── lwip
|   ├── config
|   ├── dhcps
|   ├── lwip-STABLE-2_0_2_RELEASE_VER
|   └── net_al
├── modules
|   ├── aon
|   ├── apps
|   ├── atcmd
|   ├── ble_task
|   ├── bt_task
|   ├── common
|   ├── console
|   ├── dbg
|   ├── fatfs
|   ├── gsensor
|   ├── key
|   ├── led
|   ├── letter-shell-v3
|   ├── light_sensor
|   ├── m2d
|   ├── ota
|   ├── rtos
|   ├── screen
|   ├── softwdg
|   └── xyzmodem
├── plf
|   ├── aic8800
|   ├── aic8800mc
|   └── aic8800m40
├── tests
|   ├── config
|   └── src
├── tools
├── utils
├── wifi
|   ├── config
|   ├── fhost
|   ├── hostif
|   ├── lib
|   ├── LinuxDriver
|   ├── macif
|   ├── rawdata
|   └── wlan
└── wpa_supplicant
```

1.2 功能介绍

1) 平台相关：plf/

包含 CPU 启动，系统初始化，外设驱动等。当前支持的平台有三个大类： aic8800，aic8800mc和aic8800m40。

Platform	ChipName	Notes
aic8800	AIC8800M AIC8800A AIC8800F	内置Flash，支持USB/SDIO 内置Flash，支持SDIO 外置Flash，支持USB
aic8800mc	AIC8800MC AIC8800FC	内置Flash，支持USB/SDIO 外置Flash，支持USB
aic8800m40	AIC8800M40B AIC8800M80	内置Flash，支持USB/SDIO 内置Flash，支持USB/SDIO

2) 实时系统相关：freertos/

当前 freertos 版本为 V10.2.1，其中rtos_al目录下API提供了应用程序易用的接口。

3) 软件模块相关：modules/

包含一系列相对独立的软件模块，实现特定的功能。

4) 音频相关：audio/

实现音频编解码、录音和播放等功能。

5) WiFi 协议栈相关：lwip/, wifi/, wpa_supplicant/

实现 802.11 协议栈和 TCP/IP 网络功能，当前 lwip 版本为 V2.0.2。

6) 编译脚本相关: config/, tools/

依赖于 Python 2.7，推荐使用编译器 arm-none-eabi-gcc 9.2.1。
以aic8800平台为例，不同应用场景下的编译脚本及库文件分列在目录 config/aic8800/target_***/，具体编译方法可参照2.3节。

7) 测试用例相关：tests/src/

包含各外设或模块的测试 case，用于快速验证特定功能。

8) WiFi 应用相关：applications/

基于 WiFi 和 TCP/IP 的应用层程序，可参考其中对协议栈 API 的使用。

9) BTDM 协议栈相关: btdm/

实现基础HCI 、L2CAP 、 RFCOMM 、 SDP、 GAVDP(v1.3) 、 HFP(v1.7)、 AVDTP(v1.3)/A2DP(v1.3) 、AVCTP(v1.4)/AVRCP(v1.4) 、SPP(v1.2) 、HID(v1.0);
ble profile :GATT 、 (暂时只开放HOGP 、 BAS); 以及配合SDK wifi协议线自定义开发的ble配网协议 smartconfig、还有自定义profile audtransmit client/server等。

10) BT/BLE 应用相关：modules/apps/

基于 BT 和 BLE 协议栈的应用层程序，其中根据单BT、单BLE以及BR/BLE combo不同组合，加载 bt_task或者ble_task配合apps运行。

2 编译环境配置

2.1 Windows

1) 安装 Git/Python2.7

使用git-bash作为命令行工具，确保python加入到PATH中：

```
$ export PATH=/c/Python27:/c/Python27/Scripts:$PATH
```

2) 配置编译器

下载gcc-arm 9.2.1 win32版本，https://armkeil.blob.core.windows.net/developer/Files/downloads/gnu-rm/9-2019q4/gcc-arm-none-eabi-9-2019-q4-major-win32.zip ，解压到非中文路径，如：/d/software/gcc-arm-none-eabi-9_2_1，设置环境变量：

```
$ export PATH=/d/software/gcc-arm-none-eabi-9_2_1/bin:$PATH
$ export GNUARM_4_8_LIB=/d/software/gcc-arm-none-eabi-9_2_1/lib/gcc/arm-none-eabi/9.2.1
```

2.2 Linux

1) 安装Python2.7

一般Linux发行版都会自带Python，如果没有请根据所使用的发行版自行安装，这里不再详述。

2) 配置编译器

下载gcc-arm 9.2.1 linux版本，https://armkeil.blob.core.windows.net/developer/Files/downloads/gnu-rm/9-2019q4/gcc-arm-none-eabi-9-2019-q4-major-x86_64-linux.tar.bz2 ，解压到特定目录，如：/opt/gcc_arm/gcc-arm-none-eabi-9_2_1，设置环境变量：

```
$ export PATH=/opt/gcc_arm/gcc-arm-none-eabi-9_2_1/bin:$PATH
$ export GNUARM_4_8_LIB=/opt/gcc_arm/gcc-arm-none-eabi-9_2_1/lib/gcc/arm-none-eabi/9.2.1
```

2.3 源码编译

1) 编译脚本路径

config目录结构如下，目前依据芯片类型，分为aic8800,aic8800mc和aic8800m40三个子文件夹，在每个子文件夹下按照使用功能进行划分，将实现相同功能类型的编译脚本归类在同一个target文件夹下，如 target_test。同时，在该target文件夹下，依据具体的细分功能划分不同的编译脚本。

实际使用时，编译相应功能的脚本，均会在SDK根目录build/路径下生成二进制文件，具体位置如：aic8800-sdk/build/***-aic8800/host_wb.bin

```
config/
├── aic8800
|   ├── target_test
|   |   ├── config
|   |   ├── lib
|   |   ├── res
```

```
|   |   ├── tgt_cfg
|   |   ├── tgt_cfg_hw
|   |   ├── build_bootloader.sh
|   |   ├── build_test_case.sh
|   |   ├── build_test_case_dsp.sh
|   |   └── [other-test-case.sh]
|   ├── target_wifi
|   ├── target_bt
|   ├── target_ble
|   └── [other-target-module]
├── aic8800mc
|   ├── target_test
|   |   ├── config
|   |   ├── lib
|   |   ├── res
|   |   ├── tgt_cfg
|   |   ├── tgt_cfg_hw
|   |   ├── build_test_case.sh
|   |   ├── build_test_case_sleep.sh
|   |   └── [other-test-case.sh]
|   ├── target_wifi
|   ├── target_wifi_fhostif
|   └── [other-target-module]
├── aic8800m40
|   ├── target_test
|   |   ├── config
|   |   ├── lib
|   |   ├── res
|   |   ├── tgt_cfg
|   |   ├── tgt_cfg_hw
|   |   ├── build_test_case.sh
|   |   └── [other-test-case.sh]
|   ├── target_wifi
|   ├── target_ble_wifi_fhostif
|   └── [other-target-module]
└── [some-tool-scripts]
```

2) 编译选项设置

SDK有众多编译选项，在aic8800-sdk/config/scutils.py里，已经设置了各个选项的默认值。如在scutils.py中，

```
# Console
env['CONSOLE'] = ARGUMENTS.get('CONSOLE', 'on') 意味着默认console是打开的。
```

在编译脚本中(config目录结构下的.sh文件)，用户可以修改选项的值，脚本的优先级大于scutils.py的优先级。
如在脚本中加入CONSOLE=off，则不管scutils.py中env['CONSOLE']是'on'还是'off'，都会关闭console。

```
./build_test_case.sh CONSOLE=off -j4
```

3) 添加用户代码

SDK预留了 `user/` 目录，结构如下：

```
user/
|-- config
|   |-- includelist.txt
|   |-- sourcelist.txt
|   `-- sourcelist_lib.txt
|-- lib
|   |-- aic8800
|   |   `-- armgcc_4_8
|   |       `-- libuser.a
|   |-- aic8800mc
|   |   `-- armgcc_4_8
|   |       `-- libuser.a
|   `-- aic8800m40
|       `-- armgcc_4_8
|           `-- libuser.a
`-- src
    |-- priv
    |   |-- demo_lib.c
    |   `-- demo_lib.h
    `-- pub
        |-- demo_src.c
        `-- demo_src.h
```

建议用户将自己代码加入其中，有如下原因：

- 代码保持独立，方便后续移植到新版本SDK
- 该目录支持将部分源码编译成库，以提供给其他人使用

具体使用方法：编译选项 `USER_CODE=src` 是把 `user/` 目录下所有源文件按照源码编译，并且将 `sourcelist_lib.txt` 中指定的文件打包成库文件，文件生成地址：`aic8800-sdk/build/xxx/armgcc_4_8/libuser.a`，之后需要手动替换到 `user/lib/aic8800xx/armgcc_4_8/libuser.a`；编译选项 `USER_CODE=lib` 是编译时自动链接 `user/lib/` 下的库文件。

3 软件模块 module

3.1 模块列表

目录 modules/ 下有软件模块：

```
modules/
|   ├── aon
|   ├── apps
|   ├── atcmd
|   ├── ble_task
|   ├── bt_task
|   ├── common
|   ├── console
|   ├── dbg
|   ├── fatfs
|   ├── gsensor
|   ├── key
|   ├── led
|   ├── letter-shell-v3
|   ├── light_sensor
|   ├── ota
|   ├── rtos
|   ├── screen
|   └── xyzmodem
```

3.2 模块介绍

1) AT 命令 atcmd

实现若干 AT 命令操作。

2) 串口控制台 console, letter-shell

基于串口的命令交互平台，可用于特定程序的验证、调试等。

3) 文件系统 fatfs

当前 fatfs 版本为 R0.14。

4) common部分

其中包含了基于rtos的公共co_task，co_event, co_timer等，以及可使用的list,pool,math等基础定义。

5) 蓝牙应用apps

经典蓝牙profile对应的应用程序使用案例，eg：a2dp source/sink、 hfp hf/ag、 spp、avrcp、空中升级等；单ble场景下的使用案例；以及TWS等蓝牙耳机/音响使用案例。

6) bt_task/ble_task

作为基础的task驱动不同的蓝牙应用使用案例场景。

7) aon

使用蓝牙场景时，系统睡眠前后应用层需要保存和恢复的内存处理。

8) key

基于gpio的通用按键功能。

9) led

基于pwm的通用led功能。

10) gsensor

基于硬件I2C或软件模拟I2C接口的gsensor使用案例。

11) light_sensor

基于gpio的光感元器件使用案例。

12) ota

flash升级软件基础功能。可结合wifi或bt空中升级案例。

13) screen

基础LCD相关驱动案例。

14) xyzmodem

基于串口的xyzmodem文件传输协议基础案例。

4 测试用例 test case

4.1 测试用例列表

目录 tests/ 下包含各外设或模块的test case

```
tests/
└── src
    ├── test_aes.c
    ├── test_asio.c
    ├── test_bootloader.c
    ├── test_ce.c
    ├── test_clksw.c
    ├── test_dma.c
    ├── test_dsp.c
    ├── test_fft512.c
    ├── test_fir.c
    ├── test_flash.c
    ├── test_gpadc.c
    ├── test_gpio.c
    ├── test_gpio_irq.c
    ├── test_hci.c
    ├── test_i2cm0.c
    ├── test_mp3.c
    ├── test_multadd.c
    ├── test_psram.c
    ├── test_pwm.c
    ├── test_screen.c
    ├── test_sdcard.c
    ├── test_sleep.c
    ├── test_spi_lcd_4wires.c
    ├── test_spi0.c
    ├── test_ticker.c
    ├── test_trng.c
    ├── test_uart.c
    ├── test_upgrade.c
    ├── test_usb_host.c
    ├── test_usbh_video.c
    └── test_wdt.c
```

4.2 测试用例介绍

主要是外设驱动的测试代码实现，包括：

1) 基础外设： gpio, uart, spi, i2c master, timer(ticker), flash, wdt（GPIO功能选择如下图）

AIC8800M/A/F iomux

Bank	PAD Name	Function Mode								
		Function 0	Function 1	Function 2	Function 3	Function 4	Function 5	Function 6	Function 8	Function 9
GPIOA (1.8V/3.3V)	GPIOA2	GPIOA2	uart0_rx	uart1_rx	tx_en_5g	pcm_din		i2s_dat_in_0		spi_di
	GPIOA3	GPIOA3	uart0_tx	uart1_tx	rx_en_5g	pcm_dout		i2s_dat_out_0		spi_do
	GPIOA4	GPIOA4	uart0_cts	uart1_cts	uart1_rx			codec_mclk		
	GPIOA5	GPIOA5	uart0_rts	uart1_rts	uart1_tx					
	GPIOA6	GPIOA6	i2cm_scl	uart2_rx	uart1_cts		pwm_a0			
	GPIOA7	GPIOA7	i2cm_sda	uart2_tx	uart1_rts		pwm_a1		i2s_bck_0	
	GPIOA8	uart0_rx	GPIOA8	uart2_cts			pwm_a2		i2s_lrck_0	
	GPIOA9	uart0_tx	GPIOA9	uart2_rts					i2s_dat_out_0	
	GPIOA10	GPIOA10	uart1_rx	sdio_s_data_1	spi_sck			sdhost_data_2	i2s_bck_1	uart2_rx
	GPIOA11	GPIOA11	uart1_tx	sdio_s_data_0	spi_csn			sdhost_data_3	i2s_lrck_1	uart2_tx
	GPIOA12	GPIOA12	uart1_cts	sdio_s_clk	spi_di	pwm_a0	pcm_fsync	sdhost_cmd	i2s_dat_out_1	uart2_cts
	GPIOA13	GPIOA13	uart1_rts	sdio_s_cmd	spi_do	pwm_a1	pcm_clk	sdhost_clk	i2s_dat_in_0	uart2_rts
	GPIOA14	GPIOA14	i2cm_scl	sdio_s_data_3		pwm_a2	pcm_din	sdhost_data_0	i2s_dat_in_1	
	GPIOA15	GPIOA15	i2cm_sda	sdio_s_data_2			pcm_dout	sdhost_data_1	codec_mclk	

Bank	PAD Name	Function Mode								
		ANA_Function 0	ANA_Function 1	ANA_Function 2	Function 0	Function 1	Function 2	Function 3	Function 4	
GPIOB (AIC8800M) (1.8v/3.3V/5V)	GPIOB0	led0	adc(0-1.1v)			GPIOB0(5V)	uart1_rx			
	GPIOB1		adc(0-1.1v)	TOUCH		GPIOB1	uart1_tx			
	GPIOB2	led1	adc(0-1.1v)		GPIOB2	uart1_rx				
	GPIOB3	led2	adc(0-1.1v)		GPIOB3	uart1_tx		pwm_b0		
	GPIOB4		adc(0-1.1v)	TOUCH	GPIOB4(AON)			pwm_b1	mclk_out	
	GPIOB5		adc(0-1.1v)	TOUCH	GPIOB5(AON)			pwm_b2		
	GPIOB6		adc(0-1.1v)	TOUCH	GPIOB6(AON)	uart0_rx				
	GPIOB7		adc(0-1.1v)	TOUCH	GPIOB7(AON)	uart0_tx				
	GPIOB8		adc(0-1.1v)	TOUCH	GPIOB8					
	GPIOB9				GPIOB9					
	GPIOB10				GPIOB10					
	GPIOB11				GPIOB11					
	GPIOB12				GPIOB12					
	GPIOB13				GPIOB13	pwm_b3				
	GPIOB14				GPIOB14	pwm_b4				
	GPIOB15				GPIOB15	pwm_b5				

Module	PAD Name	Analog Func0	Analog Func1	Function Mode						
				Function 0	Function 1	Function 2	Function 3			
GPIOB (AIC8800A) (1.8v/3.3V)	GPIOB_0	adc(0-1.1v)	TOUCH		GPIOB_0		pwm_b0			
	GPIOB_1	adc(0-1.1v)	TOUCH		GPIOB_1		pwm_b1			
	GPIOB_2	adc(0-1.1v)	TOUCH	GPIOB_2	pwm_b0					
	GPIOB_3	adc(0-1.1v)	TOUCH	GPIOB_3	pwm_b1					
	GPIOB_4	adc(0-1.1v)		GPIOB_4	dmic_clk					
	GPIOB_6	HP_RP		GPIOB_6	dmic_dat_1		pwm_b0			
	GPIOB_7	HP_RN		GPIOB_7	dmic_dat_2		pwm_b1			
	GPIOB_8	HP_LN		GPIOB_8	GPIOB_0	pwm_b0				
	GPIOB_9	HP_LP		GPIOB_9	GPIOB_1	pwm_b1				
	GPIOB_11	MIC3_P		GPIOB_11	GPIOB_3	dmic_dat_0	dmic_clk			
	GPIOB_13	MIC2_P		GPIOB_13	GPIOB_5	dmic_dat_1	dmic_clk			
	GPIOB_15	MIC1_P		GPIOB_15	GPIOB_7	dmic_dat_2	dmic_clk			

AIC8800MC/FC iomux

Module	PAD Name	Ext. Func	Function Mode																															
			Function 0	I/O	PULL	IDV	Function 1	I/O	PULL	IDV	Function 2	I/O	PULL	IDV	Function 3	I/O	PULL	IDV	Function 4	I/O	PULL	IDV	Function 5	I/O	PULL	IDV	Function 6	I/O	PULL	IDV					
FLASH	FLS_CLK		spi_flash_clk	O	OFF																														
	FLS_CS		spi_flash_cs	O	UP																														
	FLS_SIO_0		spi_flash_sio_0	I/O	DN	0																													
	FLS_SIO_1		spi_flash_sio_1	I/O	DN	0																													
	FLS_SIO_2		spi_flash_sio_2	I/O	UP	1																													
	FLS_SIO_3		spi_flash_sio_3	I/O	UP	1																													
GPIOA	GPIOA_0		cpu_p_swclk	I	UP	1	gpioa_0	I/O	DN	0	i2cm_scl	I/O	UP	1	debug_clk	O	OFF		i2s_lrck_0	I/O	OFF	0	pcm_fsync	I/O	DN	0	spi_lcd_sck	I/O	DN	0					
	GPIOA_1		cpu_p_swd	I/O	UP	1	gpioa_1	I/O	DN	0	i2cm_sda	I/O	UP	1	aon_uart_tx	O	OFF		i2s_bck_0	I/O	OFF	0	pcm_clk	I/O	DN	0	spi_lcd_csn_0	I/O	UP	1					
	GPIOA_2		gpioa_2	I/O	DN	0	uart0_rx	I	UP	1	uart1_rx	I	UP	1	i2cm_scl	I/O	UP	1	i2s_dat_in_0	I	OFF	0	pcm_din	I	DN	0	spi_lcd_di	I/O	DN	0					
	GPIOA_3		gpioa_3	I/O	DN	0	uart0_tx	O	OFF		uart1_tx	O	OFF		i2cm_sda	I/O	UP	1	i2s_dat_out_0	O	OFF		pcm_dout	O	OFF		spi_lcd_do	I/O	DN	0					
	GPIOA_4		gpioa_4	I/O	DN	0	uart0_cts	I	UP	1	uart1_cts	I	UP	1	uart1_rx	I	UP	1	codec_mclk	I	OFF	0	pwm_0	O	OFF		spi_lcd_cd	O	OFF						
	GPIOA_5		gpioa_5	I/O	DN	0	uart0_rts	O	OFF		uart1_rts	O	OFF		uart1_tx	O	OFF		debug_clk	O	OFF		pwm_1	O	OFF		spi_lcd_fmark	I	DN	0					
	GPIOA_6		gpioa_6	I/O	DN	0	i2cm_scl	I/O	UP	1	uart2_rx	I	UP	1	uart1_cts	I	UP	1	aon_pwm_0	O	OFF		i2s_bck_0	I/O	OFF	0	spi_lcd_csn_1	I/O	UP	1					
	GPIOA_7		gpioa_7	I/O	DN	0	i2cm_sda	I/O	UP	1	uart2_tx	O	OFF		uart1_rts	O	OFF		aon_pwm_1	O	OFF		i2s_lrck_0	I/O	OFF	0	spi_lcd_csn_2	I/O	UP	1					
	GPIOA_8		uart0_rx	I	UP	1	gpioa_8	I/O	DN	0	uart2_cts	I	UP	1																					
	GPIOA_9		uart0_tx	O	OFF		gpioa_9	I/O	DN	0	uart2_rts	O	OFF						aon_pwm_2	O	OFF		i2s_dat_out_0	O	OFF		spi_lcd_csn_3	I/O	UP	1					
	GPIOA_10	sdio_data_1	gpioa_10	I/O	DN	0	uart1_rx	I	UP	1	spi_lcd_sck	I/O	DN	0	psi_m_scl	O	DN	0	sdmmc_data_2	I/O	UP	1	i2s_bck_1	I/O	OFF	0	uart2_rx	I	UP	1					
	GPIOA_11	sdio_data_0	gpioa_11	I/O	DN	0	uart1_tx	O	OFF		spi_lcd_csn_0	I/O	UP	1	psi_m_sda	I/O	DN	0	sdmmc_data_3	I/O	UP	1	i2s_lrck_1	I/O	OFF	0	uart2_tx	O	OFF						
	GPIOA_12	sdio_clk	gpioa_12	I/O	DN	0	uart1_cts	I	UP	1	spi_lcd_di	I/O	DN	0	pcm_fsync	I/O	DN	0	sdmmc_cmd	I/O	UP	1	i2s_dat_in_1	I	OFF	0	uart2_cts	I	UP	1					
	GPIOA_13	sdio_cmd	gpioa_13	I/O	DN	0	uart1_rts	O	OFF		spi_lcd_do	I/O	DN	0	pcm_clk	I/O	DN	0	sdmmc_clk	O	UP	1	codec_mclk	I	OFF	0	uart2_rts	O	OFF						
	GPIOA_14	sdio_data_3	gpioa_14	I/O	DN	0	i2cm_scl	I/O	UP	1	spi_lcd_cd	O	OFF		pcm_din	I	DN	0	sdmmc_data_0	I/O	UP	1	i2s_dat_out_1	O	OFF		pwm_2	O	OFF						
	GPIOA_15	sdio_data_2	gpioa_15	I/O	DN	0	i2cm_sda	I/O	UP	1	spi_lcd_fmark	I	DN	0	pcm_dout	O	OFF		sdmmc_data_1	I/O	UP	1	i2s_dat_in_0	I	OFF	0	debug_clk	O	OFF						
GPIOB	GPIOB_0	host_wake_wl	gpioa_16	I/O	UP	0	pcm_fsync	I/O	DN	0	tports0	O	OFF		uart1_rx	I	UP	1	pwm_0	O	OFF		spi_lcd_csn_1	I/O	UP	1									
	GPIOB_1	wl_wake_host	gpioa_17	I/O	UP	0	pcm_clk	I/O	DN	0	tports1	O	OFF		uart1_tx	O	OFF		pwm_1	O	OFF		spi_lcd_csn_2	I/O	UP	1									
	GPIOB_2	bt_wake_host	gpioa_18	I/O	DN	0	pcm_din	I	DN	0	tports2	O	OFF		uart1_cts	I	UP	1	pwm_2	O	OFF		spi_lcd_csn_3	I/O	UP	1									
	GPIOB_3	host_wake_bt	gpioa_19	I/O	DN	0	pcm_dout	O	OFF		tports3	O	OFF		uart1_rts	O	OFF																		
USB	USB_DP	GPIOA_20																																
	USB_DM	GPIOA_21																																

AIC8800M40B/M80 iomux

Module	PAD Name	Ext. Func	Function Mode												
			Function 0	Function 1	Function 2	Function 3	Function 4	Function 5	Function 6	Function 7	Function 8	Function 9	Function 10	Function 11	Function 12	Function 13
FLASH	FLS_CLK		spi_flash_clk													
	FLS_CS		spi_flash_cs													
	FLS_SIO_0		spi_flash_sio_0													
	FLS_SIO_1		spi_flash_sio_1													
	FLS_SIO_2		spi_flash_sio_2													
	FLS_SIO_3		spi_flash_sio_3													
GPIOA	GPIOA_0		cpu_p_swclk	gpioa_0	i2cm_scl	wf_ext_pa_ctrl_0	pcm_fsync	pta_ant_sw_0	i2s_lrck_0		spdif_in	spi_lcd_sck	pwm_0	pcm_dout	pcm_clk	bt_uart_cts
	GPIOA_1		cpu_p_swd	gpioa_1	i2cm_sda	wf_ext_pa_ctrl_1	pcm_clk	pta_ant_sw_1	i2s_bck_0		spdif_out	spi_lcd_csn_0	pwm_1	pcm_din	pcm_dout	bt_uart_rts
	GPIOA_2		gpioa_2	uart0_rx	uart1_rx	wf_ext_pa_ctrl_2	pcm_din	debug_clk	i2s_dat_in_0		pta_ant_sw_0	spi_lcd_di	pwm_2	pcm_fsync	pcm_din	bt_uart_rx
	GPIOA_3		gpioa_3	uart0_tx	uart1_tx	wf_ext_pa_ctrl_3	pcm_dout	aon_uart_tx	i2s_dat_out_0		pta_ant_sw_1	spi_lcd_do		pcm_clk	pcm_fsync	bt_uart_tx
	GPIOA_4		gpioa_4	uart0_cts	uart1_cts	uart1_rx	bt_uart_rx	aud_dac_sync	codec_mclk		i2s_bck_0	spi_lcd_cd				pcm_fsync
	GPIOA_5		gpioa_5	uart0_rts	uart1_rts	uart1_tx	bt_uart_tx	aon_uart_tx	debug_clk		i2s_lrck_0	spi_lcd_fmark				pcm_din
	GPIOA_6		gpioa_6	i2cm_scl	uart2_rx	uart1_cts	bt_uart_cts	psi_s_scl	bt_uart_rx			spi_lcd_csn_1				pcm_dout
	GPIOA_7		gpioa_7	i2cm_sda	uart2_tx	uart1_rts	bt_uart_rts	psi_s_sda	bt_uart_tx		aon_pwm_0	spi_lcd_csn_2				pcm_clk
	GPIOA_8		uart0_rx	gpioa_8	uart2_cts	spdif_in			bt_uart_cts							
	GPIOA_9		uart0_tx	gpioa_9	uart2_rts	spdif_out	aon_pwm_1		bt_uart_rts		i2s_dat_out_0	spi_lcd_csn_3				
	GPIOA_10	sdio_data_1	gpioa_10	uart1_rx	bt_uart_rx	spi_lcd_sck	bt_uart_cts		sdmmc_data_2		i2s_bck_1	uart2_rx	bt_uart_cts	bt_uart_rts		
	GPIOA_11	sdio_data_0	gpioa_11	uart1_tx	bt_uart_tx	spi_lcd_csn_0	bt_uart_rts	debug_clk	sdmmc_data_3		i2s_lrck_1	uart2_tx	bt_uart_rts	bt_uart_rx		
	GPIOA_12	sdio_clk	gpioa_12	uart1_cts	bt_uart_cts	spi_lcd_di	aon_pwm_2	pcm_fsync	sdmmc_cmd		i2s_dat_in_1	uart2_cts	bt_uart_rx	bt_uart_tx		
	GPIOA_13	sdio_cmd	gpioa_13	uart1_rts	bt_uart_rts	spi_lcd_do	pwm_0	pcm_clk	sdmmc_clk		codec_mclk	uart2_rts	bt_uart_tx	bt_uart_cts		
	GPIOA_14	sdio_data_3	gpioa_14	i2cm_scl	spdif_in	spi_lcd_cd	pwm_1	pcm_din	sdmmc_data_0		i2s_dat_out_1	pta_ant_sw_0	uart0_rx			
	GPIOA_15	sdio_data_2	gpioa_15	i2cm_sda	spdif_out	spi_lcd_fmark	pwm_2	pcm_dout	sdmmc_data_1		i2s_dat_in_0	pta_ant_sw_1	uart0_tx			
GPIOB	GPIOB_0		gpiob_0	pcm_fsync	i2cm_scl	spi_lcd_sck	aon_pwm_0	wf_ext_pa_ctrl_0								
	GPIOB_1		gpiob_1	pcm_clk	i2cm_sda	spi_lcd_csn_0	aon_pwm_1	wf_ext_pa_ctrl_1								
	GPIOB_2		gpiob_2	pcm_din	psi_m_scl	spi_lcd_di	aon_pwm_2	wf_ext_pa_ctrl_2								
	GPIOB_3		gpiob_3	pcm_dout	psi_m_sda	spi_lcd_do	spdif_in	wf_ext_pa_ctrl_3								
	GPIOB_4		gpiob_4	pwm_0	i2s_lrck_0	bt_uart_rx	spdif_out									
	GPIOB_5		gpiob_5	pwm_1	i2s_bck_0	bt_uart_tx										
	GPIOB_6		gpiob_6	pwm_2	i2s_dat_in_0	bt_uart_cts										
	GPIOB_7		gpiob_7	aon_pwm_0	i2s_dat_out_0	bt_uart_rts										
USB	USB_DP	gpioa_16														
	USB_DM	gpioa_17														

2. 文件系统： sdcard, usb_host

3. 音频相关： asio, mp3

4. 蓝牙相关： hci
5. 硬件加速： aes, dma, fft512, trng
6. 8800M spi, i2c master Feature 说明

I2C Feature	GPIOA I2C	GPIOB I2C
支持AMBA 2.0 APB总线	支持	支持
支持标准模式（100 Kb/s）和快速模式（400 Kb/s）协议	支持	支持
支持可编程主模式	不支持	不支持
支持7位和10位寻址模式	支持7bit	支持7bit
支持自动时钟延展	不支持	不支持
支持可编程时钟与数据时序	支持	支持
支持DMA	支持	不支持
支持通用呼叫地址	不支持	不支持
SPI Feature	**GPIOA SPI**	**GPIOB SPI**
支持AMBA 2.0 APB总线	支持	支持
支持标准3线或4线	支持	支持
支持时钟频率配置	支持	支持
支持读写时钟相位调整	支持	支持
支持DMA	支持	不支持
支持LCD spi 3线4线模式	支持	不支持

4.3 编译指令

可通过编译选项指定case： TEST=[CASE_NAME]，例如 config/build_test_case.sh 可以这样写：

```
python ../tools/scons.py . PRODUCT=basic-rtos PLF=aic8800 BT=armgcc_4_8
PMIC_VER=lite USE_LIB_DRV=on TEST=gpio_irq
或:
python ../tools/scons.py . PRODUCT=basic-rtos PLF=aic8800 BT=armgcc_4_8
PMIC_VER=lite USE_LIB_DRV=on TEST=i2cm0
```

5 WiFi 应用 application

5.1 应用程序列表

目录 applications/ 下包含应用程序：

```
applications/
├── consolewifi
├── examplewifi
├── http_servwifi
├── uartwifi
├── fhostifwifi
└── voicewifi
```

5.2 应用程序介绍

1. examplewifi

   实现基本的 softap 启动和 station 关联等功能。

2. consolewifi

   基于 console module 的命令控制台，包含若干 WiFi 基础操作流程，各命令接口参考文档 WiFiConsole.pdf。

3. uartwifi

   基于 atcmd module 的 AT 命令交互接口，适用于 WiFi 透传模块控制，各命令接口参考文档 WiFiATCommand.pdf。

4. fhostifwifi

   基于虚拟网卡驱动 netdrv，实现嵌入式主控与aic8800系列芯片的系统低功耗功能，开发文档参考 Fhostif_UserGuide.pdf、CustomMsg_UserGuide.pdf 以及 FhostifOTA_UserGuide.pdf。

5. http_servwifi

   基于 lwip 实现的 http server，结合文件系统使用时，可通过 web 访问 U 盘或 TF 卡的文件。

6. voicewifi

   基于 lwip + audio 实现的无线语音系统。

6 btdm 接口及应用

6.1 应用程序列表

目录 btdm/ 下包含应用程序：

```
btdm/
├── bt
│   └── include
│       ├── aic_adp_api.h
│       ├── aic_adp_type.h
│       ├── aic_host_cfg.h
│       ├── bt_aon_sram.h
│       ├── bt_types_def.h
│       ├── co_errors.h
│       ├── co_types_def.h
│       └── interface
│           ├── aic_adp_a2dp.h
│           ├── aic_adp_avrcp.h
│           ├── aic_adp_dev.h
│           ├── aic_adp_hfp.h
│           ├── aic_adp_hid.h
│           ├── aic_adp_hsp.h
│           ├── aic_adp_mgr.h
│           ├── aic_adp_spp.h
│           ├── aic_adp_test.h
│           ├── aic_adp_tws.h
│           └── aic_bt_msg.h
├── ble
│   ├── ble_adp
│   │   └── aic_ble_adp_api.h
│   ├── ble_app
│   │   ├── app_audtransmitc
│   │   │   ├── app_audtransmitc.c
│   │   │   └── app_audtransmitc.h
│   │   ├── app_audtransmits
│   │   │   ├── app_audtransmits.c
│   │   │   └── app_audtransmits.h
│   │   ├── app_batt
│   │   │   ├── app_batt.c
│   │   │   └── app_batt.h
│   │   ├── app_hid
│   │   │   ├── app_hid.c
│   │   │   └── app_hid.h
│   │   ├── app_main
│   │   │   ├── app.c
│   │   │   ├── app.h
│   │   │   ├── app_task.c
│   │   │   └── app_task.h
│   │   ├── app_sec
│   │   │   └── app_sec.h
│   │   └── app_smartconfig
│   │       ├── app_smartconfig.c
│   │       └── app_smartconfig.h
│   ├── ble_dbg
│   │   ├── aicble_dbg.c
```

```
|   |   └── aicble_dbg.h
|   ├── ble_profiles
|   |   ├── audtransmit
|   |   |   ├── audtransmitc
|   |   |   |   ├── api
|   |   |   |   |   └── audtransmitc_task.h
|   |   |   |   ├── sourcelist.txt
|   |   |   |   └── src
|   |   |   |       ├── audtransmitc.c
|   |   |   |       ├── audtransmitc.h
|   |   |   |       └── audtransmitc_task.c
|   |   |   └── audtransmits
|   |   |       ├── api
|   |   |       |   └── audtransmits_task.h
|   |   |       ├── sourcelist.txt
|   |   |       └── src
|   |   |           ├── audtransmits.c
|   |   |           ├── audtransmits.h
|   |   |           └── audtransmits_task.c
|   |   ├── bas
|   |   |   └── bass
|   |   |       ├── api
|   |   |       |   └── bass_task.h
|   |   |       └── src
|   |   |           └── bass.h
|   |   ├── hogp
|   |   |   ├── hogp_common.h
|   |   |   └── hogpd
|   |   |       ├── api
|   |   |       |   └── hogpd_task.h
|   |   |       └── src
|   |   |           └── hogpd.h
|   |   ├── prf
|   |   |   └── prf_user.c
|   |   └── smartconfig
|   |       └── smartconfigs
|   |           ├── api
|   |           |   └── smartconfig_task.h
|   |           └── src
|   |               ├── smartconfig.c
|   |               ├── smartconfig.h
|   |               └── smartconfig_task.c
|   └── ble_stack
|       ├── ble_ip
|       |   ├── aicble.h
|       |   ├── rwapp_config.h
|       |   ├── rwble_hl_config.h
|       |   ├── rwip.h
|       |   ├── rwip_config.h
|       |   ├── rwip_task.h
|       |   └── rwprf_config.h
|       ├── common
|       |   └── api
|       |       ├── ble_utils.h
|       |       ├── co_bt.h
|       |       ├── co_bt_defines.h
|       |       ├── co_error.h
|       |       └── co_hci.h
```

```
|       ├── hl
|       |   ├── api
|       |   |   ├── att.h
|       |   |   ├── gap.h
|       |   |   ├── gapc_task.h
|       |   |   ├── gapm_task.h
|       |   |   ├── gattc_task.h
|       |   |   ├── prf_types.h
|       |   |   └── rwble_hl_error.h
|       |   └── inc
|       |       ├── attm.h
|       |       ├── gapc.h
|       |       ├── gapm.h
|       |       ├── prf.h
|       |       ├── prf_utils.h
|       |       └── smpc.h
|       └── ke
|           └── api
|               ├── ke_mem.h
|               ├── ke_msg.h
|               ├── ke_task.h
|               └── ke_timer.h
├── config
|   ├── includelist.txt
|   └── sourcelist.txt
└── lib
    └── armgcc_4_8
        ├── libbleonly.a
        ├── libbt.a
        ├── libbtdm.a
        └── libbttws.a
```

6.2 协议栈接口介绍

1) bt

bt/include：bt协议栈基本接口以及定义

bt/include/interface：bt协议栈profile接口以及定义

可参考modules/apps/src/bt中使用方法。

```
        ├── app_a2dp.c
        ├── app_a2dp_source.c
        ├── app_avrcp.c
        ├── app_bt.c
        ├── app_bt_queue.c
        ├── app_console.c
        ├── app_hfg.c
        ├── app_hfp.c
        ├── app_hid.c
        ├── app_hsp.c
        ├── app_ota_box.c
        ├── app_spp.c
        ├── app_test.c
        └── app_tws.c
```

app_bt.c：

为经典蓝牙应用层主文件，包含各个profile的注册以及消息处理函数注册、按键消息函数注册等基本接口。

app_bt_init()函数为主入口，在bt_task.c中task执行起来后被调用。

app_bt_queue.c：

为bt应用层消息队列，其他task想使用bt协议栈API，需创建消息发送之bt task再执行。

app_console.c：

为bt协议栈应用层对应console接口创建。

其余均为profile应用。

2) ble

ble/ble_stack: ble协议栈基本接口以及定义

ble/ble_profiles: ble协议栈profile接口以及定义

ble/ble_dbg: ble协议栈打印level修改

ble/ble_app: ble协议栈profile定义app的接口以及定义

可参考modules/apps/src/ble中使用方法。

```
        ├── app_ble_audtransmit.c
        ├── app_ble_console.c
        ├── app_ble_only.c
        ├── app_ble_queue.c
        └── app_m2d.c
```

app_ble_only.c

为低功耗蓝牙应用层主文件，包含各个profile的注册以及消息处理函数注册等基本接口。

app_ble_init()函数为主入口，在ble_task.c中task执行起来后被调用。需要注意的是，当执行Bt/Ble Combo运行

时，ble被合并再bt_task中与经典蓝牙一起执行，app_ble_init()在bt_task中被调用。

app_ble_queue.c：

为ble应用层消息队列，其他task想使用协议栈API，在单ble或者bt/ble combo时，需创建消息发送之ble_task或bt_task（combo）再执行。

app_ble_console.c：

为ble协议栈应用层对应console接口创建。

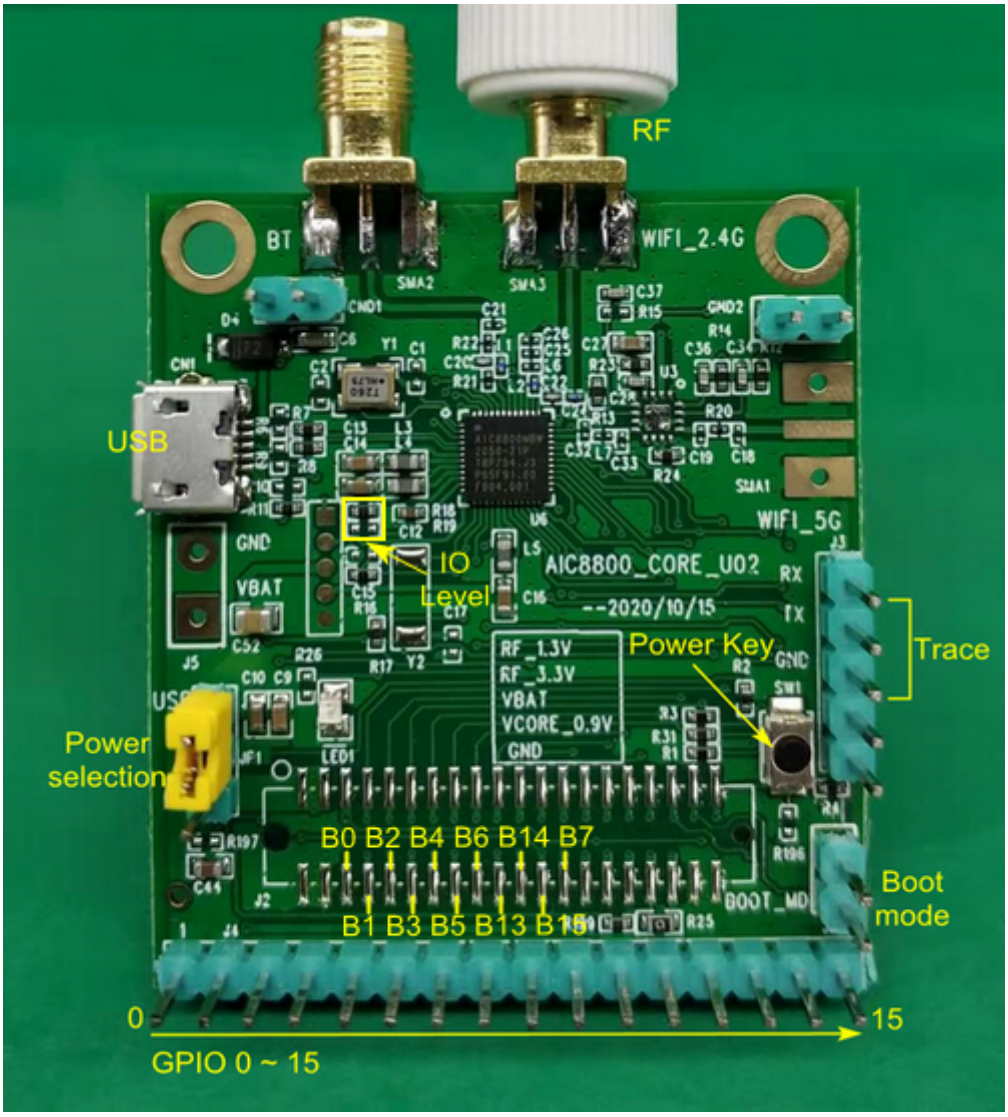
其余均为profile应用。

7 使用 demo 板卡

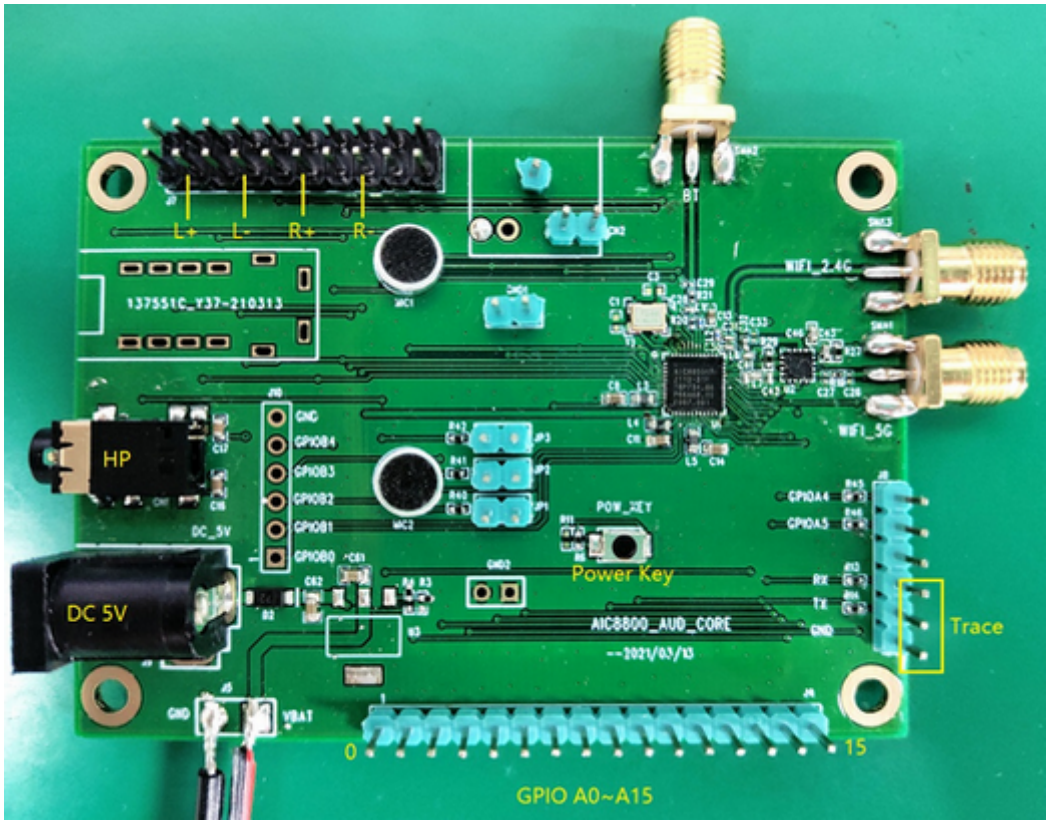
7.1 板卡外观

demo板外观如图所示：

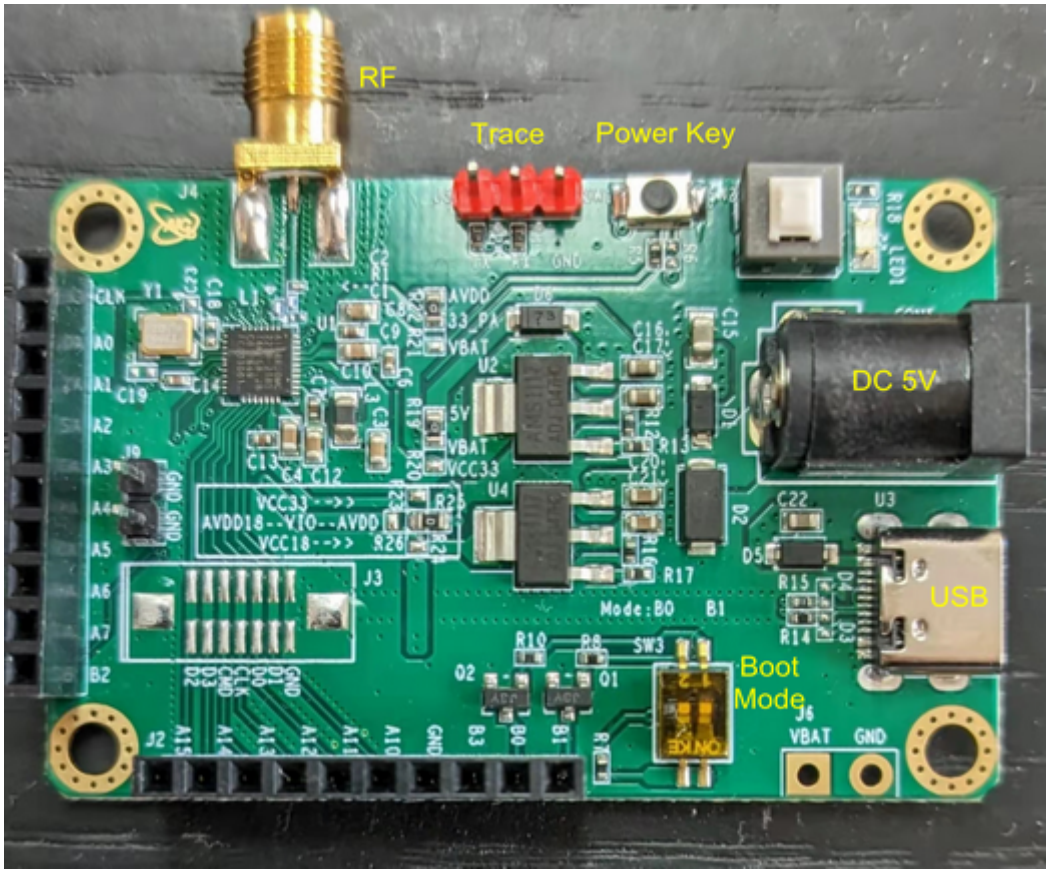
1) AIC8800M EVB



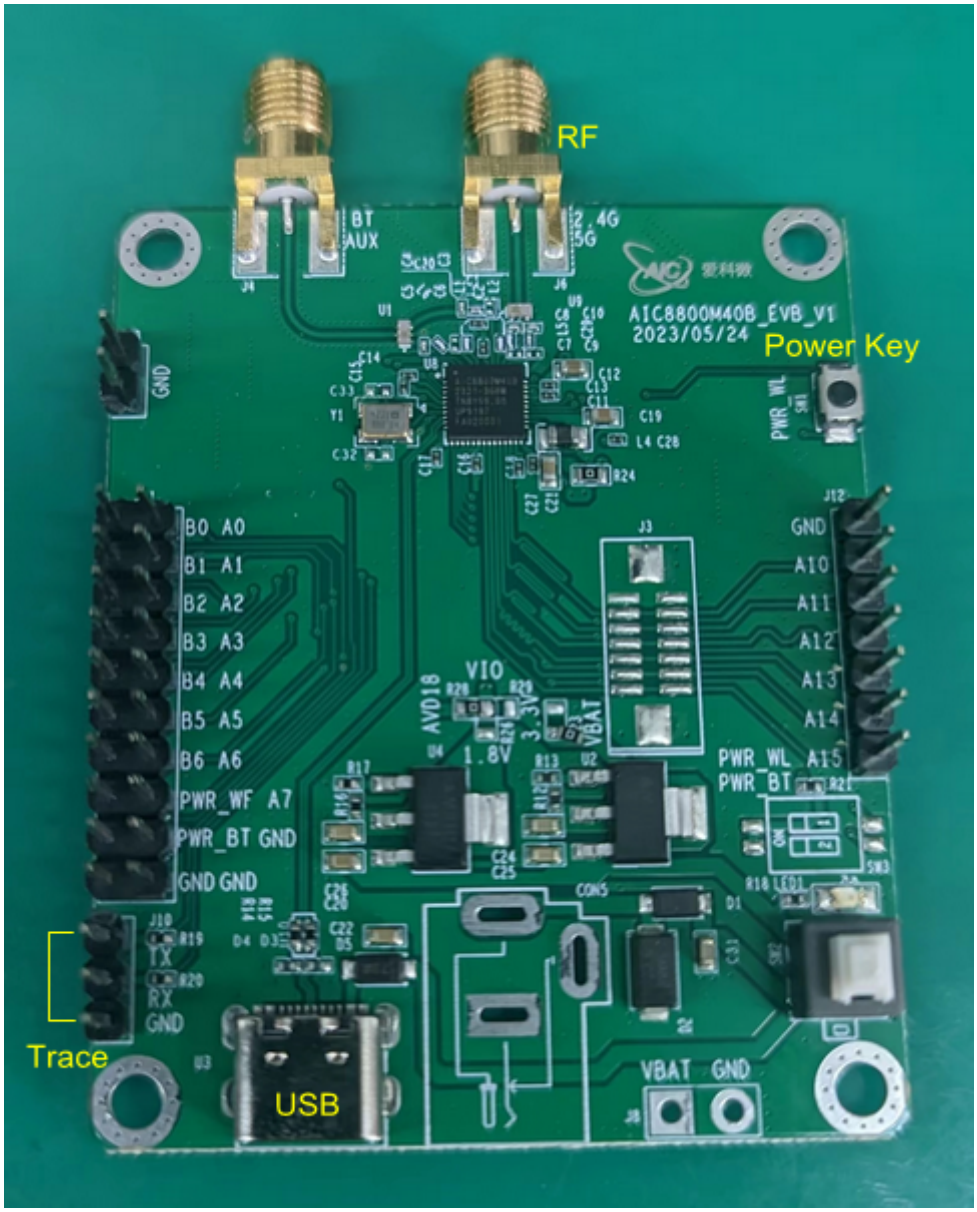
2) AIC8800A EVB



3) AIC8800MC EVB



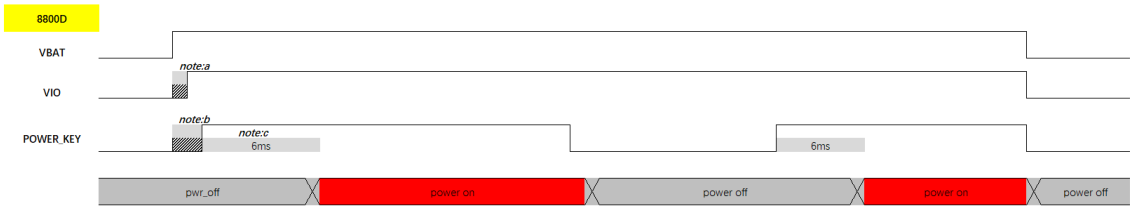
3) AIC8800M40B EVB



7.2 部分接口介绍

1) Power Key 为开/关机按键，其对应的芯片 PWRKEY 管脚有 200K 欧姆内阻，与外部上拉电阻串联后的分压值 V~PWRKEY~ >= VIO * 0.6 时有效

时序图如下：



Note a: VIO's power on time >= VBAT's
Note b: power key's high time >= VIO's
Note c: chip all power on ready time >= power key high time + 6ms

Note a: power key's high time >= VBAT's
Note b: chip all power on ready time >= power key high time + 6ms
Note c: in mcu mode, power_key must be high >=6ms
Note d: user can configure power key high threshold time when power off

注意：对于不使用按键，PWRKEY管脚常拉高的设计，SDK应定义USER_DEFINED_PWRKEY_EN为0。

2) Trace 串口可用于查看 log 和烧录 bin 文件
注意：如果使用PC的USB转串口线连接板卡Trace串口，请确保串口电平与板卡IO电平相匹配，电平高于3.3V的串口线有烧坏芯片的风险，具体IO电平选择参考第5)条。

3) Boot mode 跳线短接后能强制进入下载模式
注意：AIC8800M/AIC8800A/AIC8800MC 的 Boot mode实现方式有所不同，具体如下：

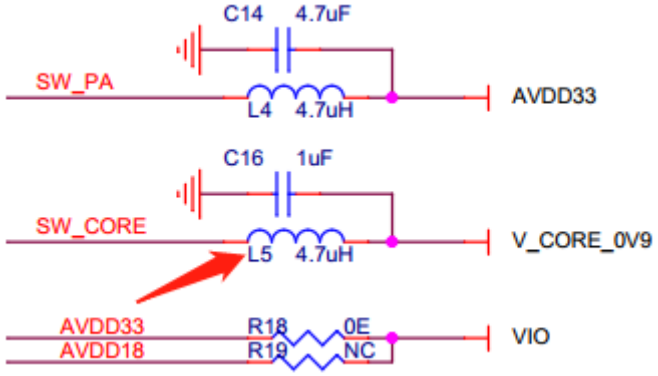
芯片型号	Boot mode实现
AIC8800M	GPIOB6/B7 同时接地
AIC8800A	HPRP/HPRN 同时接地
AIC8800MC	GPIOB0/B1 同时接地
AIC8800M40B/M80	GPIOB0/B1 同时接地

4) AIC8800M EVB 通过 USB 供电时， power selection 跳线选择 “USB”

5) AIC8800M EVB 上 IO Level 可通过R18, R19选择IO电平(VIO)

VIO \ 电阻	R18	R19
3.3V	0欧姆	NC
1.8V	NC	0欧姆

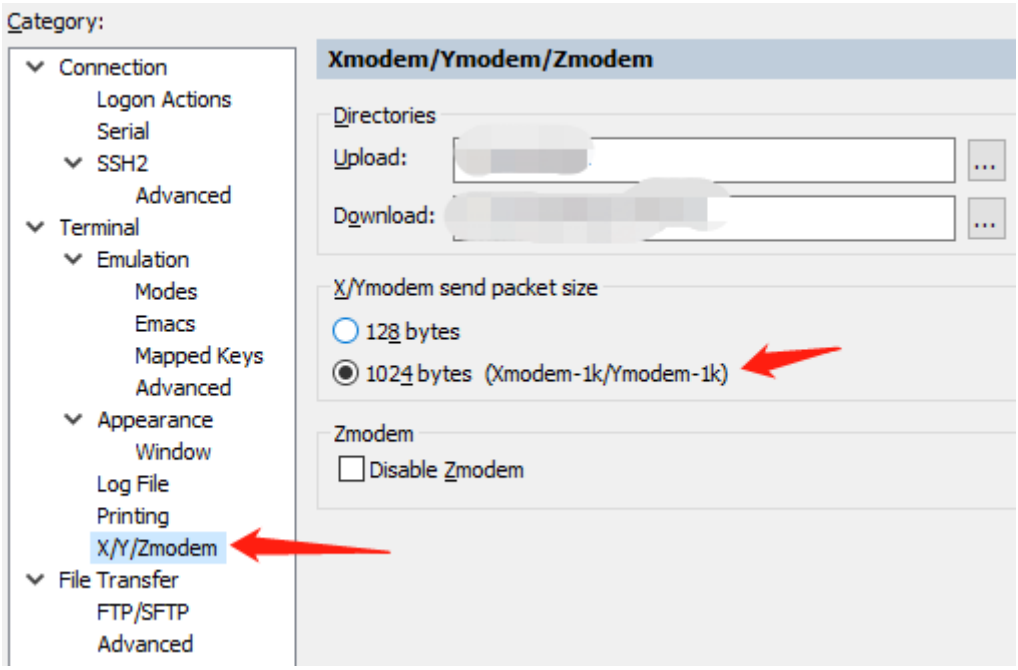
6) AIC8800M EVB 上电感器件 L5 不焊接时，SDK 编译选项应设置为：PMIC_VCORE=ldo



7.3 bin 文件烧录

release 下载模式

1) **设置串口软件** 通过串口连接Demo板Trace串口，打开SecureCRT，波特率 921600，设置 Xmodem-1k 模式。若在打开串口情况下进行设置，则需重新打开串口，使软件串口配置生效



2. **进入下载模式** Boot Mode 短接，芯片上电，按下 Power Key

```
Bootrom [Jan 30 2021, c71e5d6]
Copyright (C) 2018-2020 AICSemi Ltd.

RstCause:0000,Boot:3d,1
Mcu mode

Boot >
Boot >
```

3. **启动文件传输** 键入命令 **x ADDRESS LENGTH** （例如 x 8000000 60000），第一个参数是程序启动地址（16进制），第二个参数是擦除长度（Byte，16进制，一般是bin文件长度4KB取整），SecureCRT Xmodem 发送 bin 文件，等待传输结束。烧录时，可根据实际的 bin 文件大小确定第二个参数

```
Bootrom [Jan 30 2021, c71e5d6]
Copyright (C) 2018-2020 AICSemi Ltd.

RstCause:0000,Boot:3d,1
Mcu mode

Boot >
Boot >x 8000000 60000
Receiving XModem (921600 bps) data to 0x08000000
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
```

4. **上传文件** 如上图 SecureCRT 软件启动传输之后会持续打印 C 字符，此时通过菜单栏，Transfer->Send Xmodem，选择对应的 bin 文件即可传输

```
Boot >
Boot >x 8000000 60000
Receiving XModem (921600 bps) data to 0x08000000
CCCCCCCCCCCCCCCCCCCCCCCCCCC
Starting xmodem transfer.  Press Ctrl+C to cancel.
Transferring ble_wifi_fhostif.bin...
  100%     701 KB       21 KB/sec    00:00:32       0 Errors

xyzModem - CRC mode, 1(SOH)/701(STX)/0(CAN) packets, 3 retries
717952 (0x000AF480) bytes received

OK
```

5. **执行应用** 去除 Boot Mode 短接，芯片重新上电，按下 Power Key，程序自动执行

debug 下载模式

1. **设置串口软件** 同 **release 下载模式**

2. **进入下载模式** Boot Mode 不短接，芯片上电，在按下 Power Key 之前持续敲回车，此操作需在 count=0 计时结束之前完成

```
Bootrom [Jan 30 2021, c71e5d6]
Copyright (C) 2018-2020 AICSemi Ltd.

RstCause:0000,Boot:3d,0
Mcu mode
count=4
count=3
count=2

Boot >Boot abort

Boot >
Boot >
```

3. **启动文件传输** 同 **release 下载模式**
4. **上传文件** 同 **release 下载模式**
5. **执行应用** SecureCRT Xmodem 发送完成 bin 文件之后，可以直接键入命令 **g 8000000** 执行烧录的 bin 文件

```
Bootrom [Jan 30 2021, c71e5d6]
Copyright (C) 2018-2020 AICSemi Ltd.

RstCause:0000,Boot:3d,0
Mcu mode
count=4
count=3
count=2

Boot >Boot abort

Boot >
Boot >
Boot >
Boot >
Boot >
Boot >x 8000000 60000
Receiving XModem (921600 bps) data to 0x08000000
CCCCCCCCCCCCCCCCCCCCCCCCCCCCC
Starting xmodem transfer.  Press Ctrl+C to cancel.
Transferring ble_wifi_fhostif.bin...
  100%     701 KB      21 KB/sec    00:00:32       0 Errors

xyzModem - CRC mode, 1(SOH)/701(STX)/0(CAN) packets, 3 retries
717952 (0x000AF480) bytes received

OK
Boot >g 8000000
```

注意：如果是未经出厂设置的芯片，启动会遇到 "boot abort: -1"，此时需要执行以下命令后给芯片断电，再重复上述 2) ~ 5) 步骤

```
f 1 3 1 2 1
f 3
```

8 问题定位

8.1 操作前检查

1) 芯片供电

- AIC8800M/A/F VBAT 支持电压 2.3~5V；VIO 支持 1.8V或3.3V，配置方式见6.2节；VCORE09支持DCDC或LDO模式，对应软件配置见7.2节第6)条；
- AIC8800MC/FC VBAT 支持电压 2.3~5V；VIO 支持 1.8V或3.3V，VFLASH 支持1.8V或3.3V配置
- AIC8800M40B/M80 VBAT 支持电压 3.0~3.6V；VIO 支持 1.8V或3.3V，VFLASH 支持1.8V或3.3V配置

2) 外接 Trace 串口

Trace串口电平为VIO，外接串口电平应保持一致，高于3.3V有损坏芯片的风险

3) WiFi/BT 天线

测试WiFi/BT功能之前，请确保已经连接天线

8.2 FAQ

1) 芯片第一次使用，遇到"boot abort: -1"

见7.3节

2) GPIO在EVB板的位置

见7.1节

3) Flash 相关操作（erase/write/read）的起始地址和数据长度有何要求

Flash操作需注意： erase 地址和长度都要求4KB对齐，write 地址256B对齐长度无要求，read 地址4B对齐长度无要求

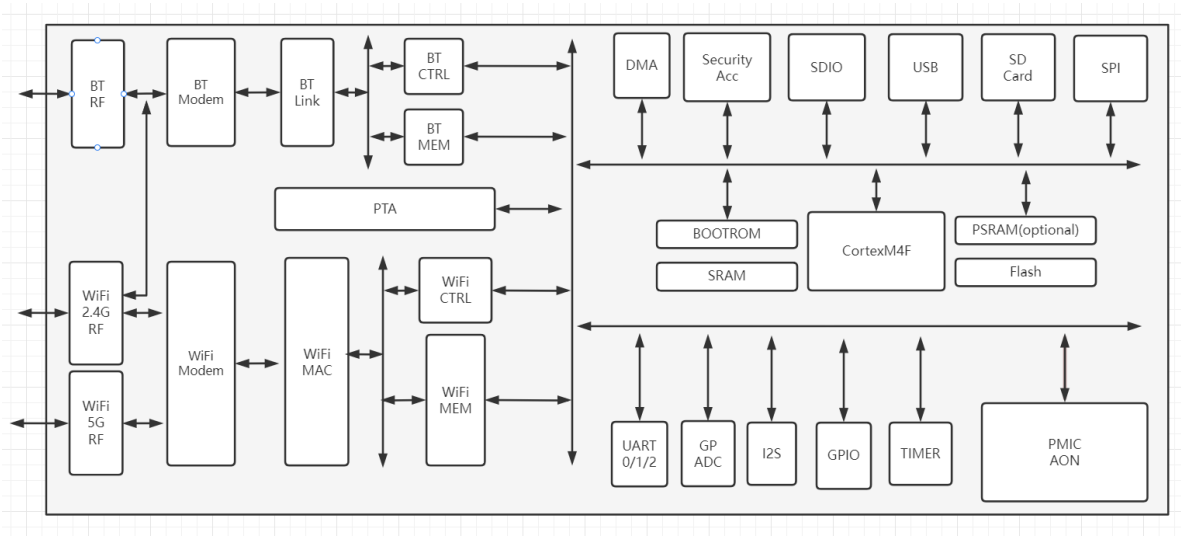
4) 系统重启接口
pmic_chip_reboot()

5) AIC8800M与AIC8800D的区别

芯片型号	内置Flash	依赖主控	应用场景	开发方式
AIC8800M	是	否	IoT 低功耗	开发基于FreeRTOS的应用程序， 在AIC8800M上运行 （虚拟网卡等应用也需开发主控端驱动程序）
AIC8800D	否	是	无线网卡 高吞吐	开发基于Linux的驱动程序， 在主控端运行

9 附件1 芯片内存外设mapping

9.1 系统架构



9.2 内存及外设mapping

AIC8800M/A/F

Module	Base	Size
ROM	0x0000_0000	64KB
WIFI ROM		
BT ROM		
RAM0	0x0010_0000	384KB
RAM1	0x0016_0000	448KB
AON RAM	0x001E_0000	64KB
FLASH	0x0400_0000	32MB
PSRAM	0x0600_0000	32MB
CACHE0	0x0800_0000	64MB
WLAN	0x4030_0000	
UART0	0x4004_1000	
UART1	0x4004_2000	
UART2	0x4004_3000	
IPC	0x4004_4000	
ASIO	0x4005_0000	
DMA	0x4007_0000	
SDMMC	0x4008_2000	
PWM	0x4010_1000	
I2C_M	0x4010_2000	
WDG0	0x4010_3000	
WDG1	0x4010_4000	
SPI	0x4010_5000	
TIMER0	0x4010_6000	
TIMER1	0x4010_7000	
SPINLOCK	0x4010_8000	
TRNG	0x4010_9000	
WDG2	0x4010_A000	
WDG3	0x4010_C000	
BT	0x4060_0000	
USB	0x4020_0000	

Module	Base	Size
SDIO	0x4024_0000	
TIMER2	0x4050_1000	
WDG4	0x4050_2000	
IOMUX	0x4050_3000	
GPIO	0x4050_4000	
PWR_CTRL	0x4050_6000	

AIC8800MC/FC

Module	Base	Size
ROM	0x0000_0000	64KB
WIFI ROM		
BT ROM		
RAM0	0x0010_0000	512KB
RAM1(CACHE)	0x0018_0000	32KB
BT_RAM	0x0018_8000	64KB
FLASH-DATA	0x0400_0000	
CACHE-DATA	0x0800_0000	
DMA	0x4002_0000	
UART0	0x4003_2000	
UART1	0x4003_3000	
UART2	0x4003_4000	
IPC	0x4003_5000	
SDMMC	0x4003_6000	
ASIO	0x4003_7000	
CPUSYSCTRL	0x4010_0000	
PWM	0x4010_1000	
I2C_M	0x4010_2000	
WDG0	0x4010_3000	
WDG1	0x4010_4000	
SPI	0x4010_5000	
TIMER0	0x4010_6000	
TIMER1	0x4010_7000	
TRNG	0x4010_8000	
ANA_REG0	0x4010_9000	
GPIOA	0x4010_B000	
MSADC	0x4010_D000	
USB	0x4020_0000	
SDIO	0x4024_0000	
MMSYSCTRL	0x4024_1000	

Module	Base	Size
AONSYSCTRL	0x4050_0000	
TIMER2	0x4050_1000	
ANA_REG1	0x4050_2000	
IOMUX	0x4050_4000	
CM_RF_INTF	0x4050_5000	
PWR_CTRL	0x4050_6000	
AON_PWM	0x4050_7000	
WDG2	0x4050_8000	

AIC8800M40B/M80

Module	Base	Size
ROM	0x0000_0000	64KB
WIFI ROM		
BT ROM		
RAM0	0x0010_0000	256KB
RAM1	0x0014_0000	384KB
RAM2	0x001A_0000	256KB
RAM3(CACHE)	0x0018_0000	32KB
BT_RAM	0x0020_0000	64KB
FLASH-DATA	0x0400_0000	
CACHE-DATA	0x0800_0000	
DMA	0x4002_0000	
UART0	0x4003_2000	
UART1	0x4003_3000	
UART2	0x4003_4000	
IPC	0x4003_5000	
SDMMC	0x4003_6000	
ASIO	0x4003_7000	
CPUSYSCTRL	0x4010_0000	
PWM	0x4010_1000	
I2C_M	0x4010_2000	
WDG0	0x4010_3000	
WDG1	0x4010_4000	
SPI	0x4010_5000	
TIMER0	0x4010_6000	
TIMER1	0x4010_7000	
TRNG	0x4010_8000	
ANA_REG0	0x4010_9000	
GPIOA	0x4010_B000	
MSADC	0x4010_D000	
USB	0x4020_0000	

Module	Base	Size
SDIO	0x4024_0000	
MMSYSCTRL	0x4024_1000	
AONSYSCTRL	0x4050_0000	
TIMER2	0x4050_1000	
ANA_REG1	0x4050_2000	
IOMUX	0x4050_4000	
GPIOB	0x4050_5000	
PWR_CTRL	0x4050_6000	
AON_PWM	0x4050_7000	
WDG2	0x4050_8000	
CM_RF_INTF	0x4050_9000	

10 附件2 芯片时钟设置

本章节中的内容位于sysctrl_api.h中。

10.1 设置时钟

通过如下接口配置时钟：

```
void sysctrl_clock_cfg(int cfg);
```

参数cfg的取值范围如下：

```
enum {
    CLK_CFG_D480S240P120F120 = 0,
    CLK_CFG_D480S240P120F80,
    CLK_CFG_D480S240P120F60,
    CLK_CFG_D480S240P60F120,
    CLK_CFG_D480S240P60F80,
    CLK_CFG_D480S240P60F60,
    CLK_CFG_D240S240P120F120,
    CLK_CFG_D240S240P120F60,
    CLK_CFG_D240S240P60F60,
    CLK_CFG_D240S120P60F120,
    CLK_CFG_D240S120P60F60,
    CLK_CFG_D240S120P30F60,
    CLK_CFG_D120S120P60F60,
    CLK_CFG_D120S120P30F60,
    CLK_CFG_D80S80P40F40,
    CLK_CFG_D52S52P26F26,
    CLK_CFG_MAX,
};
```

以CLK_CFG_D240S240P120F60为例，各部分的含义如下：

配置	含义
S240	将系统时钟（即MCU主频）配置成240M
P120	将外设时钟配置成120M
F60	将Flash时钟配置成60M

CLK_CFG_D240S240P120F60亦为系统默认配置。

10.2 获取时钟设置

通过如下接口获取时钟设置：

```
uint32_t sysctrl_clock_get(int sys_per);
```

返回值的单位是Hz。

参数sys_per的取值范围如下：

```
enum {
    SYS_FCLK = 0,
    SYS_HCLK,
    SYS_PCLK,
    PER_UART0,
    PER_UART1,
    PER_UART2,
    PER_PSRAM,
    PER_FLASH,
    PER_PWM,
};
```

10.3 设置PLL

通过如下接口设置PLL：

```
void sysctrl_pll_cfg(int cfg);
```

参数cfg的范围为：

```
enum {
    PLL_CFG_OFF = 0,
    PLL_CFG_320,
    PLL_CFG_480,
    PLL_CFG_320_480,
};
```

pll在初始化的时候已经配置好，**用户在绝大部分情况下不需要自行配置**，自行配置反而会引起不必要的问题。

PLL的配置与系统中有无BT、WIFI以及时钟配置相关，下表展示了其关系：

BT	WIFI	时钟配置	PLL配置
无	有/无	CLK_CFG_D52S52P26F26	PLL_CFG_OFF*1、PLL_CFG_320、PLL_CFG_480、PLL_CFG_320_480*2
无	有/无	CLK_CFG_D52S52P26F26以外的配置	PLL_CFG_480*1、PLL_CFG_320_480
有	无	CLK_CFG_D52S52P26F26	PLL_CFG_320*1、PLL_CFG_320_480
有	无	CLK_CFG_D52S52P26F26以外的配置	PLL_CFG_320_480
有	有	CLK_CFG_D52S52P26F26	PLL_CFG_320*1、PLL_CFG_320_480
有	有	CLK_CFG_D52S52P26F26以外的配置	PLL_CFG_320_480

*1：功耗最低的配置。

*2：兼容性最好的配置。

11 附件3 射频参数设置

11.1 WiFi射频参数

发射功率、TXOP、CCA等参数配置参考文档：WiFiApi

11.2 BT射频参数

[TBD]