

AC630X 软件问题整理

链接: <https://pan.baidu.com/s/1kjhBSPTfegAm3xRpuVv8NA>

提取码: fxq3

以下是杰理客户问题反馈的二维码图片，可以通过微信扫二维码填写反馈！

也可以通过访问链接填写反馈: <https://www.wjx.cn/vj/O1EbvrN.aspx>

我们将第一时间安排对应工程协助解决！



1.各芯片封装及 SDK 工程文件选择	20200619	xin.....	4
2.可拆卸电池(纽扣/干电池)供电(硬件有特殊接法--自行对原理图)	20200619	xin.....	4
3.单模 BLE 功耗评估(DCDC 带电感的硬件方式)	20200619	xin.....	7
4.软件可控是否进 sniff(power_down)睡眠	20200619	xin.....	7
5.串口等外设与睡眠的协调处理	20200619	xin.....	7
6. HID 开发	20200619	xin.....	8
7.631X 用户手册映射通道描述有误	20200619	xin.....	8
8.Timer 用作 PWM 输出--映射到任意 IO 上	20200619	xin.....	9
9. 631X 关闭 PB2 复位功能	20200619	xin.....	11
10. 631X 设置 PB1 为长按复位功能	20200619	xin.....	12
11. 功耗异常	20200619	xin.....	12
12.AC6318 等 VBAT 跟 VDDIO 绑定一起得供电注意	20200801	YWP.....	12
13. BLE 传输速率	20201130	xin.....	13
14. 636X 的 IO 口中断及翻转	20201130	xin.....	14
15. 动态调节功率(020 之后)	20201130	xin.....	16
16. 长按复位及高电平复位(020 之后)	20201130	xin.....	17
17. 631X 的 PB2 不能硬件接地或接到三极管基极	20201130	xin.....	17
18. 630NSDK 050 版本 使用 APP(杰理 OTA 升级)给芯片 br30(637X)升级失败问题处理方法 LAQ			20201202
19. 637X 的 timer 时钟源不能选 OSC 晶振	20201230	xin.....	17
20. OTA 升级需要关闭不必要的外设	20201230	xin.....	19
21. OTA 升级需要定义获取 BLE 广播包、profile 数据的接口	20201230	xin.....	19
22. 双模共 MAC 地址有部分手机概率配对失败	20201230	xin.....	20
23. 查看代码异常复位等异常情况原因	20210105	xin.....	20
24. 出现电压检测、ADC 检测不准	20210105	xin.....	21
25. 636N 区分复位跟唤醒	20210111	xin.....	22
26. APP OTA 双备份升级(只能用于 4Mbits 以上的芯片)	20210114	xin.....	23
27. 6351D PR0 和 PR1 当普通 IO 口使用	20210121	xin.....	24
28. 仿真、调试、烧录、升级	20210121	xin.....	25
26. SPP 开启 pin_code 功能	20210125	xin.....	26
27. 芯片供电范围及防烧芯片措施	20210125	xin.....	27
28. 内置充电功能及预留过压过流保护	20210125	xin.....	28
29. 636X 系列 5 路 PWM	20210125	xin.....	31
30. 注册软件定时器不耗费硬件定时器	20210202	xin.....	32
31. 固件加 key 用于烧录	20210202	xin.....	33
32. 测试盒用于测试 BLE 遥控器的按键	20210203	xin.....	34
33. 630 v070 版本 SDK, PC 端定时通过 AT 指令将数据发送给 ble 主机, ble 主机也定时发送数据给 ble 从机, 从机再将数据发送给 PC, 过一段时间后 ble 从机会复位的处理方法 LJW			36
34. 获取运行时堆的剩余可用 ram 的大小	20210218	xin.....	36
35. 开启硬件定时器中断注意点	20210305	xin.....	36
36. 637X 芯片唤醒部分打印信息缺失	20210309	xin.....	37
37. 632X 的 PB2 引脚修改状态会 VCM 复位	20210414	xin.....	37
38. 632X 获取复位源及唤醒源	20210603	xin.....	37

39. 升级保留 VM 及 MAC 地址数据	20210603	xin.....	39
40. 不需要按键功能，需要自行初始化唤醒口	xin	20210622.....	40
41. 低功耗流程图	xin	20210622.....	41
42. 获取系统开机时间	xin	20210622.....	41
43. SDK 框架图	xin	20210622.....	42
44. 63x 各系列的触摸按键资源	hjy	20210726.....	42
45. 632x 软关机会导致直接复位（VCM 复位）的问题	FSW	20210730.....	43
46. ac636n_chargebox_sdk_release_v1.3.0 一拖二、一拖八烧录需要修改.ini 文件	FSW	20210805.....	43
47. AC632X 芯片的 PWM 资源使用	20210823	xin.....	44
48. 内部 flash 储存数据	20210824	xin.....	44
49. 浮点打印	20210824	xin.....	46
50. 获取芯片 UID	20210824	xin.....	46
51.AC632N 系列 IO 口开机状态	20210825	YWP.....	47

1.各芯片封装及 SDK 工程文件选择 20200619 xin

632X -> bd19 -> sdk\apps\spp_and_le\board\bd19\AC632N_spp_and_le.cbp[AC632N 用户手册 V1.0.pdf]
 631X -> bd29 -> sdk\apps\spp_and_le\board\bd29\AC631X_spp_and_le.cbp [AC630N 用户手册 V1.0.pdf]
 635X -> br23 -> sdk\apps\spp_and_le\board\br23\AC635N_spp_and_le.cbp[AC695X 用户手册 V1.0.pdf]
 636X -> br25 -> sdk\apps\spp_and_le\board\br25\AC636N_spp_and_le.cbp[AC696X 用户手册 V1.0.pdf]
 637X -> br30 -> sdk\apps\spp_and_le\board\br30\AC6367_spp_and_le.cbp[AC897N 用户手册 V1.0.pdf]

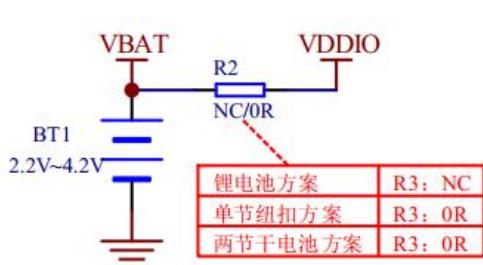
AC636N_hid.cbp 用于开发 HID 产品，手机连接是在系统的设置界面上

AC636N_spp_and_le.cbp 用于开发自定义 BLE 协议产品，手机连接是在 APP 上面连接

AC636N_mesh.cbp 用于开发 mesh 产品

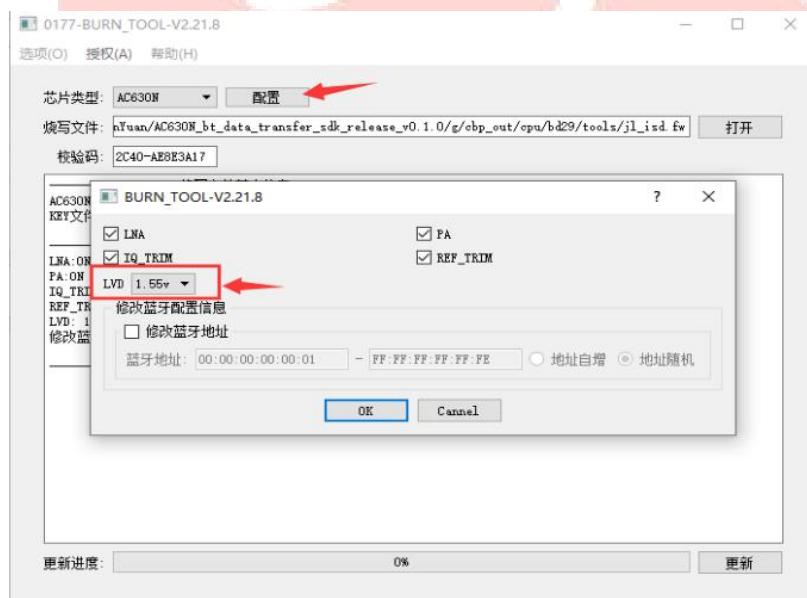
各个封装需要的用户手册，寄存器都有差别的，申请资料时，请明确用户手册！！！

2.可拆卸电池(纽扣/干电池)供电(硬件有特殊接法--自行对原理图) 20200619 xin



Power

需要用烧录器烧录**最低档位(不同封装，不一样的电压值)**，才能支持低电压供电，**只能烧录一次，无法再修改**





如若出现无法低压供电，验证方式：

```
printf("\n\n\n\n\n");
log_i("\n~~~~~");
log_i("      setup_arch %s", __DATE__, __TIME__);
log_i("~~~~~\n");

#define GET_LVD_CON() (P33_CON_GET(P3_VLVD_CON))
#define GET_LVD_EN() (P33_CON_GET(P3_VLVD_CON) & 0x01)
#define GET_LVD_LEV() (P33_CON_GET(P3_VLVD_CON)>>3 & 0x7)
printf("GET_LVD_CON:0x%x", GET_LVD_CON());
printf("GET_LVD_EN:%d", GET_LVD_EN());
printf("GET_LVD_LEV:%d", GET_LVD_LEV());
P33_VLVD_EN(1);
/* P33_VLVD_EN(0); */

/* clock_dump(); */

```

可拆卸电池，可选关闭LVD复位功能
需详细测试才能生产

```
#define GET_LVD_CON() (P33_CON_GET(P3_VLVD_CON))
#define GET_LVD_EN() (P33_CON_GET(P3_VLVD_CON) & 0x01)
#define GET_LVD_LEV() (P33_CON_GET(P3_VLVD_CON)>>3 & 0x7)

printf("GET_LVD_CON:0x%x", GET_LVD_CON());
printf("GET_LVD_EN:%d", GET_LVD_EN());
printf("GET_LVD_LEV:%d", GET_LVD_LEV());
P33_VLVD_EN(1);
/* P33_VLVD_EN(0); */
```

对应的打印：

```
[00:00:00.100]~~~~~
[00:00:00.101]~~~~~ setup_arch Dec 23 2020 10:40:54
[00:00:00.102]~~~~~[00:00:00.102]GET_LVD_CON:0x85
[00:00:00.103]GET_LVD_EN:1
[00:00:00.103]GET_LVD_LEV:0 ← 看是哪个档位
[00:00:00.104][Info]: [PMU]--Reset Source : 0x1
[00:00:00.105][Info]: [PMU]P33 RESET
[00:00:00.105]--P3 Reset Source : 0x2
```

为避免 SDK 差异和不必要的风险，建议客户主动打开 LVD 低电复位：

纽扣选型：建议客户选择脉冲电流为 20mA 的纽扣电池

电池型号	标称电压	标称容量	工作电流	连续电流	脉冲电流	最大尺寸 直径*厚度	重量
CR3032	3V	550mAh	0.2mA	3.0mA	20mA	30.0mm*3.2mm	6.8g
CR2477	3v	950mAh	0.2mA	3.0mA	20mA	24.5mm*7.7mm	9.9g

CR2450	3v	550mAh	0.2mA	3.0mA	20mA	24.5mm*5.0mm	5.8g
CR2430	3v	270mAh	0.2mA	3.0mA	20mA	24.5mm*3.0mm	4.3g
CR2412	3v	90mAh	0.1mA	1.0mA	15mA	24.5mm*1.2mm	2.2g
CR2354	3v	530mAh	0.2mA	3.0mA	20mA	23.0mm*5.4mm	6.3g
CR2335	3V	300mAh	0.2mA	3.0mA	20mA	23.0mm*3.5mm	4.2g
CR2330	3V	260mAh	0.2mA	2.0mA	20mA	23.0mm*3.0mm	3.7g
CR2325	3V	190mAh	0.2mA	2.0mA	20mA	23.0mm*2.5mm	3.2g
CR2320	3V	130mAh	0.2mA	2.0mA	20mA	23.0mm*2.0mm	2.7g
CR2032	3V	210mAh	0.2mA	2.0mA	20mA	20.0mm*3.2mm	3.0g
CR2025	3V	150mAh	0.2mA	2.0mA	20mA	20.0mm*2.5mm	2.5g
CR2016	3V	75mAh	0.1mA	1.0mA	15mA	20.0mm*1.6mm	1.7g
CR1632	3V	120mAh	0.1mA	1.0mA	15mA	16.0mm*3.2mm	1.8g
CR1620	3 V	70mAh	0.1mA	1.0mA	10mA	16.0mm*2.0mm	1.2g
CR1616	3V	50mAh	0.1mA	1.0mA	10mA	16.0mm*1.6mm	1.1g
CR1225	3V	50mAh	0.1mA	1.0mA	5mA	12.5mm*2.5mm	0.9g
CR1216	3V	25mAh	0.1mA	1.0mA	5mA	12.0mm*1.6mm	0.7
CR1025	3V	30mAh	0.1mA	1.0mA	5mA	10.0mm*2.5mm	0.6g
CR1220	3V	38mAh	0.1mA	1.0mA	5mA	12.0mm*2.0mm	0.8g

3.单模 BLE 功耗评估(DCDC 带电感的硬件方式) 20200619 xin

关机功耗 <2uA

开机瞬态电流 第一次开机 50mA 以后开机 14mA (如果已经装机,需要升级程序,注意第一次开机电流)

开机蓝牙不工作--idle 默认进睡眠 ~30uA

开机蓝牙不工作--idle 跑 CPU(24M) 400uA-2mA

开机蓝牙广播功耗平均功耗 50uA-200uA 软件可调(典型值: 1.6s 广播间隔 60uA)

开机蓝牙连接平均功耗 60uA - 500uA 软件可调(连接间隔对手机会有差异)

4.软件可控是否进 sniff(power_down)睡眠 20200619 xin

//代码任意地方添加此段代码,全局修改此变量,底层会自动获取判断

u8 can_enter_lp = 0; //置 1 可进睡眠, 置 0 不可进睡眠

static u8 custom_idle_query(void)

{

 if(can_enter_lp){

 return 1;

 }else{

 return 0;

}

}

REGISTER_LP_TARGET(custom_lp_target) = {

 .name = "custom_lp",

 .is_idle = custom_idle_query,

};

5.串口等外设与睡眠的协调处理 20200619 xin

主控进睡眠(sniff、power_down)状态, 任何外设(uart、timer、mcpwm、spi)均为失效, 无法工作, 只有唤醒 IO 有效, 能触发主控唤醒后, 其他外设才能继续工作。

举例: 串口与唤醒 IO 的协调方式

在睡眠状态下, 芯片是无法正常接收串口数据的, 需要主机 MCU 用独立的 IO, 唤醒芯片后, 主机 delay 延时一会, 才能发送数据, 芯片才能正常接收(做法与 nordic 基本一致)

```

***** PWR config *****
struct port_wakeup port0 = {
    .pullup_down_enable = ENABLE,
    .edge = FALLING_EDGE,
    .attribute = BLUETOOTH_RESUME,
    .iomap = IO_PORTB_01, // 唤醒口选择 3.
};

const struct sub_wakeup sub_wkup = {
    .attribute = BLUETOOTH_RESUME,
};

const struct charge_wakeup charge_wkup = {
    .attribute = BLUETOOTH_RESUME,
};

const struct wakeup_param wk_param = {
    .port[1] = &port0,
    .sub = &sub_wkup,
    .charge = &charge_wkup, // 自行添加此段代码
};

#if 1
u8 can_enter_lp = 0;
static u8 custom_idle_query(void)
{
    if(can_enter_lp){
        return 1; // 可以进入low_power
    }else{
        return 0;
    }
}

REGISTER_LP TARGET(custom_lp_target) = {
    .name = "custom_lp",
    .is_idle = custom_idle_query,
};
#endif

```

2. 设置PB1为唤醒口

PB1拉低后唤醒主控

判断是否为指定IO唤醒是则后续不进sniff需把can_enter_lp置0

4. 此后才能正常接收串口数据

5. 此段为demo串口通信不要维持主机需要发送命令告知从机可进sniff

```

364 void sleep_exit_callback(u32 usec)
365 {
366     putchar('>');
367     APP_IO_DEBUG_1(A, 6);
368
369     extern u8 can_enter_lp;
370     if(gpio_read(IO_PORTB_01) == 0){
371         can_enter_lp = 0;
372         void dynamic_modify_profile(void);
373         /* dynamic_modify_profile(); */
374     }else{
375         /* can_enter_lp = 1; */
376     }
377 }
378
379 void sleep_enter_callback(u8 step)
380 {
381     /* 此函数禁止添加打印 */
382     if (step == 1) {
383         putchar('<');
384         APP_IO_DEBUG_0(A, 6);
385         /* dac_power_off(); */
386     } else {
387         /* gpio_set_pull_up(IO_PORTA_03, 0); */
388     }
389 }
390
391 case AT_LOW_N:
392     printf("AT_LOW_N\n");
393     extern u8 can_enter_lp;
394     if(buf[0] == '?'){
395         can_enter_lp = 1;
396     }
397     break;
398
399 case AT_MFSD_N:
400     printf("AT_MFSD_N\n");
401     if(buf[0] == '?'){
402     }
403     break;

```

6. HID 开发 20200619 xin

无 HID 开发经验, 请参考 **HID 开发指南 V1.0.pdf** 的介绍, 再评估案子的开发方式

7.631X 用户手册映射通道描述有误 20200619 xin

```

output channel mux 15
*****
[15:0] outs_grp0 = {tmr3_pwmout, wlc_bt_freq, ch2_pwm_l, ch2_pwm_h,
                     ch1_pwm_l, ch1_pwm_h, ch0_pwm_l, ch0_pwm_h,
                     uart2_tx, p11_12m, btosc_clk, rtos1_clk,
                     tmr1_pwmout, tmr0_pwmout, uart1_tx, uart0_tx} 0
[15:0] outs_grp1 = {tmr3_pwmout, tmr2_pwmout, ch2_pwm_l, ch2_pwm_h,
                     ch1_pwm_l, ch1_pwm_h, ch0_pwm_l, ch0_pwm_h,
                     uart2_tx, p11_24m, btosc_clk, rtos1_clk,
                     wlc_bt_pro, tmr0_pwmout, uart1_tx, uart0_tx};
[15:0] outs_grp2 = {tmr3_pwmout, tmr2_pwmout, ch2_pwm_l, ch2_pwm_h,
                     ch1_pwm_l, ch1_pwm_h, ch0_pwm_l, ch0_pwm_h,
                     uart2_tx, p11_24m, btosc_clk, 1'b0, 0
                     tmr1_pwmout, wlc_bt_active, uart1_tx, uart1_rts};

```

//以下例子，是 636X 的 demo，其他芯片需要自行对照各自的用户手册，修改寄存器值

sniff 状态下 timer 会停,导致 PWM 波形不正常,需关 sniff
`#define TCFG_LOWPOWER_LOWPOWER_SEL 0//SLEEP_EN`

//OUTPUT_CH0 一般用作打印映射了

//Timer1 用作系统定时器了,不能动

`static void set_timer0_pwm(u32 fre, u8 duty)`

{

`#define OSC_HZ 24000000`

`JL_TIMER0->CON = 0x0000;`

`JL_TIMER0->CNT = 0x0000;`

`JL_TIMER0->PRD = OSC_HZ / (4 * fre);`

`JL_TIMER0->PWM = (JL_TIMER0->PRD * duty) / 100;`

`JL_TIMER0->CON = (1 << 4) | (1 << 0) | (2 << 2);/// | (1 << 8)去掉 bit8 关闭默认 IO 输出`

//只有 3 组通道

`//output_channel_0:dir=0; pd=1; pu=1; out=0; die=0;`

`//output_channel_1:dir=0; pd=1; pu=1; out=0; die=1;`

`//output_channel_2:dir=0; pd=1; pu=1; out=1; die=0;`

`#define T0_USE_CHANNEL 0 //可选配 0/1/2`

`#define PWM0_IO IO_PORTB_03`

`#if(T0_USE_CHANNEL == 0)//OUTPUT_CH0 一般用作打印映射了,使用后无法打印`

`SFR(JL_IOMAP->CON1, 8, 4, 2);`

`gpio_set_hd(PWM0_IO, 0);`

`gpio_set_pull_up(PWM0_IO, 1);`

`gpio_set_pull_down(PWM0_IO, 1);`

`gpio_direction_output(PWM0_IO, 0);`

`gpio_set_die(PWM0_IO, 0);`

`#endif`

}

`static void set_timer2_pwm(u32 fre, u8 duty)`

{

`#define OSC_HZ 24000000`

`JL_TIMER2->CON = 0x0000;`

`JL_TIMER2->CNT = 0x0000;`

`JL_TIMER2->PRD = OSC_HZ / (4 * fre);`

`JL_TIMER2->PWM = (JL_TIMER2->PRD * duty) / 100;`

`JL_TIMER2->CON = (1 << 4) | (1 << 0) | (2 << 2);/// | (1 << 8)去掉 bit8 关闭默认 IO 输出`

`#define T2_USE_CHANNEL 1 //可选配 0/1/2`

`#define PWM2_IO IO_PORTB_04`

`#if(T2_USE_CHANNEL == 1)`

```
SFR(JL_IOMAP->CON3, 20, 4, 14); //OUTPUT_CH1
gpio_set_hd(PWM2_IO, 0);
gpio_set_pull_up(PWM2_IO, 1);
gpio_set_pull_down(PWM2_IO, 1);
gpio_direction_output(PWM2_IO, 0);
gpio_set_dle(PWM2_IO, 1);

#endif
}

static void set_timer3_pwm(u32 fre, u8 duty)
{
#define OSC_HZ 24000000
    bit_clr_ie(IRQ_TIME3_IDX);
    JL_TIMER3->CON = 0x0000;
    JL_TIMER3->CNT = 0x0000;
    JL_TIMER3->PRD = OSC_HZ / (4 * fre);
    JL_TIMER3->PWM = (JL_TIMER3->PRD * duty) / 100;
    JL_TIMER3->CON = (1 << 4) | (1 << 0) | (2 << 2); // | (1 << 8) 去掉 bit8 关闭默认 IO 输出
#define T3_USE_CHANNEL 2 //可选配 0/1/2
#define PWM3_IO IO_PORTB_05
#if(T3_USE_CHANNEL == 2)
    SFR(JL_IOMAP->CON3, 24, 4, 15); //OUTPUT_CH2 //通道 2 无法映射 timer0
    gpio_set_hd(PWM3_IO, 0);
    gpio_set_pull_up(PWM3_IO, 1);
    gpio_set_pull_down(PWM3_IO, 1);
    gpio_direction_output(PWM3_IO, 1);
    gpio_set_dle(PWM3_IO, 0);

#endif
}
void pwm_init(void)
{
    set_timer0_pwm(500L, 20);
    set_timer2_pwm(1000L, 50);
    set_timer3_pwm(1500L, 80);
}
void pwm_close(void)
{
    set_timer0_pwm(0, 0);
    set_timer2_pwm(0, 0);
    set_timer3_pwm(0, 0);
}
```

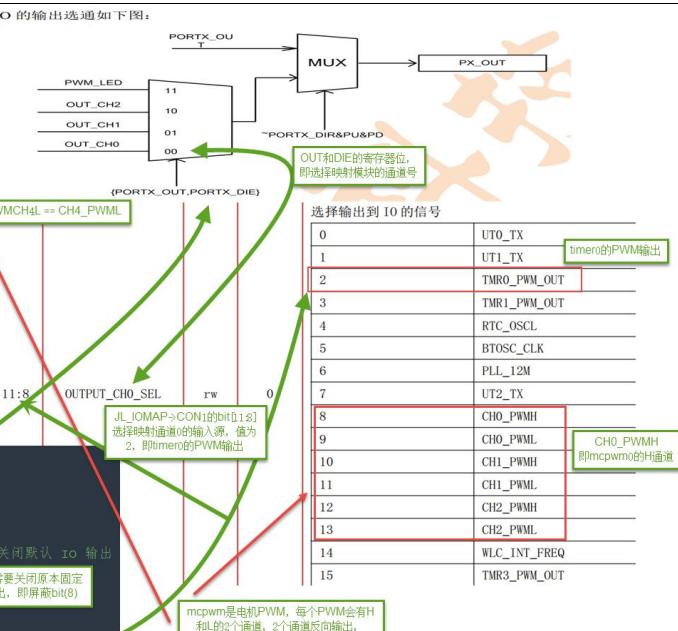
图片是 635X 的用户手册描述：

PAS	SEGS		SODDATA	BT_Active	ALNK0_DAT1A	IIC_SCL_D	ADC1	Touch5	PWM0	UART0TXA
PA6	SEG6	PSRAM_D1A	SODDATA1A	Wlan_Active	spi0_DCQ1	ALNK0_DAT2A	IC_SDA_D	ADC2	Touch7	UART0RXA
PA7	SEG7	PSRAM_D2A	SODDATA2A	BT_priority	spi0_DAT2CQ1	ALNK0_DAT3A			Touch7	TMR0
PA8	SEG8	PSRAM_D3A	SODDATA3A	BT_Freq	spi0_DCQ0	ALNK0_MCLKA				PFN2
PA9	SEG9	PA_EN	SODCMDA	SPDIF_IN_A	PAD0	ALNK0_SCLKB	UART1_CTS	ADC3	Touch8	UART2TXB
PA10	SEG10	LNA_EN	SODCLKA	SPDIF_IN_B	PAD1	ALNK0_RXKB	UART1_RTS	ADC3	Touch9	UART2RXB
PA11	SEG11			PAD_0D	ALNK0_DAT0B				UART0TD	
PA12	SEG12			PAD_WR	ALNK0_DAT1B		ADC4	PWM1	UART0RD	TMROCK

1.Tx_Conf timer x control register

Bit	Name	RW	Description
15	PND	r	PND:中断请求标志,当timer溢出或产生捕获动作时会被硬件置1,需要由软件清0。
14	PCLR	w	PCLR:软件在此位写入‘1’将清除PND中断请求标志。
13-10	reserved	r	保留
9	PWM_INV	rw	PWM_INV:PWM信号反向输出。每个timer都会有独立的固定的PWM_IO
8	PWM_EN	rw	PWM_EN:PWM信号输出使能。此位置1后,相应IO口的功能将会被PWM信号输出替代。

```
static void set_timer0_pwm(u32 fre, u8 duty)
{
#define OSC_HZ 24000000
    JL_TIMER0->CON = 0x0000;
    JL_TIMER0->CNT = 0x0000;
    JL_TIMER0->PRD = OSC_HZ / (4 * fre);
    JL_TIMER0->FWM = (JL_TIMER0->PRD * duty) / 100;
    JL_TIMER0->CON = (1 << 4) | (1 << 0) | (2 << 2); //去掉 bit8 关闭默认 io 输出
    //只有 3 须通道
    //output_channel 0:dir=0; pd=1; pu=1; out0; die=0;
    //output_channel 1:dir=0; pd=1; pu=1; out0; die=1;
    //output_channel 2:dir=0; pd=1; pu=1; out1; die=0;
#define TO_USE_CHANNEL 0 //可选配 0/1/2
#define PWM0_IO IO_PORT0_03
#if(TO_USE_CHANNEL == 0) //OUTPUT_CH0 一般用作打印映射了, 使用后无法打印
    SFR(JL_IOMAP->CON1, 0, 0);
    gpio_set_hd(PWM0_IO, 0);
    gpio_set_pull_up(PWM0_IO, 1);
    gpio_set_pull_down(PWM0_IO, 0);
    gpio_direction_output(PWM0_IO, 0);
    gpio_set_die(PWM0_IO, 0);
#endif
}
```



9. 631X 关闭 PB2 复位功能 20200619 xin

631X 的 PB2 底层默认打开了一接低就会立马复位
想用 PB2 为 IO 功能, 添加以下代码

```
void board_power_init(void)
{
    log_info("Power init : %s", FILE );
    //< close PB2 short key reset
    p33_and_1byte(P3_PR_PWR, ~BIT(3));
    //< close PB2 long key reset
    p33_and_1byte(P3_PINR_CON, 0);

    power_init(&power_param);

    /*sdpg_config(1);*/
}
```

10. 631X 设置 PB1 为长按复位功能 20200619 xin

```
5 struct reset_param rs_param = {
6     .en = 1,
7     .mode = LONG_8S_RESET, //LONG_4S_RESET, //
8     .level = 0,
9     .iomap = IO_PORTB_01, ←
10 };
11
12 void board_power_init(void)
13 {
14     log_info("Power init : %s", __FILE__);
15
16     //< close short key reset
17     p33_and_1byte(P3_PR_PWR, ~BIT(3));
18     //< close long key reset
19     /* p33_and_1byte(P3_PINR_CON, 0); */
20
21     power_init(&power_param);
22
23     /*sdpg_config(1);*/
24
25     power_set_callback(TCFG_LOWPOWER_LOWPOWER_SEL, sleep_enter_callback,
26     power_keep_dacvdd_en(0);
27
28     power_wakeup_init(&wk_param);
29
30     void power_reset_init(const struct reset_param * rs_param);
31     power_reset_init(&rs_param);
32     /* P33_CON_SET(P3_PINR_CON, 0, 1, 1); //开启 8S 复位 */
33     /* P33_CON_SET(P3_PINR_CON, 0, 1, 0); //关闭 8S 复位 */
34     /* p33_tx_1byte(P3_PINR_CON, 0x31); */
35     p33_tx_1byte(P3_PINR_CON, 0x31);
36     r_printf("p33_rx_1byte:0x%x", p33_rx_1byte(P3_PINR_CON));
37
38 #if !(TCFG_IKEY_ENABLE && TCFG_AKEY_ENABLE) ← 长按8s复位
39     //charge_check_and_set_pinr(0);
40 #endif
41 }
42 #endif
```

636X, 635X, 637X 芯片都是默认用 PB1 做按键口，跟长按复位功能口

11. 功耗异常 20200619 xin

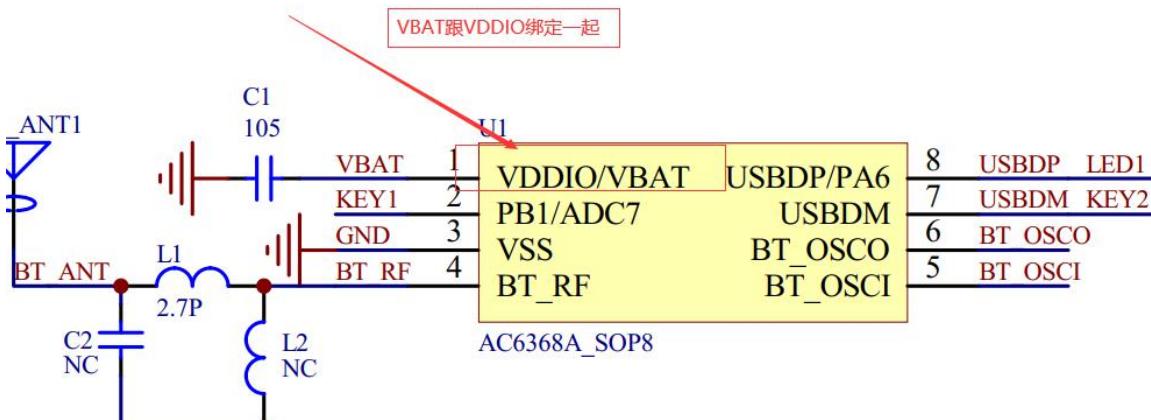
如遇关机功耗大, sniff 状态功耗大等问题, 建议升级 SDK

不要用 AC630N_bt_data_transfer_sdk_release_v0.1.0 这个版本, 公版默认功耗较高, 需要修改地方太多。
可找深圳办要测试的 SDK, 测试板子是否正常, 再同步对比

AC630N_bt_data_transfer_sdk_release_v0.6.0 这个版本的 SDK, 636X 的功耗会偏大 200uA 左右, 需要打补丁
[060-sniff 功耗大.rar]

12.AC6318 等 VBAT 跟 VDDIO 绑定一起得供电注意 20200801 YWP

AC631 的烧录需要注意电压, 特别是 AC6318A 或者 AC6368A 这种, VBAT 和 VDDIO 绑定一起的, 输入电压不能高于 3.6V, 不然会把主控烧坏。例如, 在烧录时需要把 5V 转降压为 3.3V 给主控 VBAT 供电。



13. BLE 传输速率 20201130 xin

3. 部分手机参考升级速度（有效数据速率）

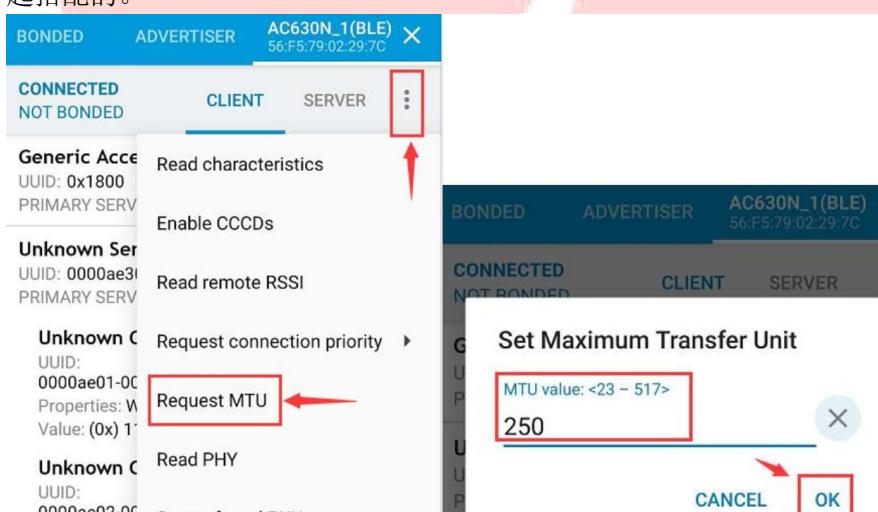
- Android 4.3 设备: < 2KB/s
- Android 4.4 设备: 4~5KB/s
- Android 5、6、7 设备: 5~8KB/s (部分手机可超过 20kB/s)
- IOS 8.3 设备: 3~4KB/S
- IOS 10、11 设备: 6~8KB/s

连接间隔参数，可固件发起申请，决定权在主机端(即手机)，没有处理好，默认速度会偏慢的
MTU 参数，要看手机 APP 的设计，MTU 越大速率越大

ACL 包长可调，要看手机兼容性，越大速率越大

```
const int config_btctler_le_features = LE_ENCRYPTION | LE_DATA_PACKET_LENGTH_EXTENSION;
const int config_btctler_le_acl_packet_length = 251;//27;
```

针对一款手机，没有说固定或默认的速率，这些都是软件可调，理论峰值，是需要系统+APP+固件，3方一起搭配的。



IO 口中断函数，及上升沿下降沿翻转例子

```
void set_port_wakeup_edge(u8 index,u8 flag) //翻转边沿触发
{
    u8 edge_reg;
    u8 wkup_en;
    u8 rising_edge = 0;
    u8 falling_edge = 0;
    if(flag){
        rising_edge = 1;
    }else{
        falling_edge = 1;
    }
    u8 i = index;
    u8 iomap = 0;
    if(i == 1) iomap = port0.iomap;
    else if(i == 2) iomap = port2.iomap;
    else if(i == 3) iomap = port3.iomap;
    else

        return;
    wkup_en = p33_rx_1byte(P3_WKUP_EN);
    r_printf("P3_WKUP_EN:%d",wkup_en);
    wkup_en &= ~BIT(i);
    p33_tx_1byte(P3_WKUP_EN,wkup_en);

    gpio_set_pull_up(iomap, 0);
    gpio_set_pull_down(iomap, 0);
    gpio_set_direction(iomap, 1);
    gpio_set_dle(iomap,1);
    edge_reg = p33_rx_1byte(P3_WKUP_EDGE);
    r_printf("P3_WKUP_EDGE:%d",edge_reg);
    if (rising_edge) {
        edge_reg &= ~BIT(i);
        gpio_set_pull_down(iomap, 1);
    } else if (falling_edge) {
        edge_reg |= BIT(i);
        gpio_set_pull_up(iomap, 1);
    }
    r_printf("P3_WKUP_EDGE:%d",edge_reg);
    p33_tx_1byte(P3_WKUP_EDGE, edge_reg);

    wkup_en = p33_rx_1byte(P3_WKUP_EN);
    r_printf("P3_WKUP_EN:%d",wkup_en);
```

```

wkup_en |= BIT(i);
p33_tx_1byte(P3_WKUP_EN,wkup_en);
}

void port_edge_wakeup_isr(u8 index,u32 gpio) //IO 中断函数，触发引脚即是唤醒 IO
{
    g_printf("wakeup port %d,gpio:PORT%c_%02d",
            index, (gpio/IO_GROUP_NUM) + 'A',gpio%IO_GROUP_NUM );
    printf("gpio_read:%d",gpio_read(port0.iomap));
    printf("gpio_read:%d",gpio_read(port2.iomap));
    printf("gpio_read:%d",gpio_read(port3.iomap));
    static u8 flag = 0;
    if(flag){
        flag = 0;
        set_port_wakeup_edge(index,0); //不需要翻转的，请注释
    }else{
        flag = 1;
        set_port_wakeup_edge(index,1);
    }
}

```

以上 2 个函数直接 copy 进 board.... .c 文件中

```

falling_edge = 1;
u8 i = index;
u8 imap = 0; //port2.iomap;
if(i == 1) imap = port0.iomap;
else if(i == 2) imap = port2.iomap;
else if(i == 3) imap = port3.iomap;
else
    return;
wkup_en = p33_rx_1byte(P3_WKUP_EN);
r_printf("P3_WKUP_EN:%d",wkup_en);
wkup_en |= -BIT(i);
p33_tx_1byte(P3_WKUP_EN,wkup_en);

gpio_set_pull_up(imap, 0);
gpio_set_pull_down(imap, 0);
gpio_set_direction(imap, 1);
gpio_set_dle(imap, 1);
edge_reg = p33_rx_1byte(P3_WKUP_EDGE);
r_printf("P3_WKUP_EDGE:%d",edge_reg);
if(rising_edge){
    edge_reg &= -BIT(i);
    gpio_set_pull_down(imap, 1);
} else if(falling_edge){
    edge_reg |= BIT(i);
    gpio_set_pull_up(imap, 1);
}
r_printf("P3_WKUP_EDGE:%d",edge_reg);
p33_tx_1byte(P3_WKUP_EDGE, edge_reg);

wkup_en = p33_rx_1byte(P3_WKUP_EN);
r_printf("P3_WKUP_EN:%d",wkup_en);
wkup_en |= BIT(i);
p33_tx_1byte(P3_WKUP_EN,wkup_en);
}

void port_edge_wakeup_isr(u8 index,u32 gpio)
{
    g_printf("wakeup port %d,gpio:PORT%c_%02d",
            index, (gpio/IO_GROUP_NUM) + 'A',gpio%IO_GROUP_NUM );
    printf("gpio_read:%d",gpio_read(port0.iomap));
    printf("gpio_read:%d",gpio_read(port2.iomap));
    printf("gpio_read:%d",gpio_read(port3.iomap));
    static u8 flag = 0;
    if(flag){
        flag = 0;
        set_port_wakeup_edge(index,0);
    }
}

```

```

303 struct port_wakeup port0 = {
304     .pullup_down_enable = ENABLE,
305     .edge               = FALLING_EDGE,
306     .attribute          = BLUETOOTH_RESUME,
307     .imap               = IO_PORTB_02,
308     .filter_enable      = ENABLE,
309 };
310 struct port_wakeup port2 = {
311     .pullup_down_enable = ENABLE,
312     .edge               = FALLING_EDGE,
313     .attribute          = BLUETOOTH_RESUME,
314     .imap               = IO_PORTB_01,
315     .filter_enable      = ENABLE,
316 };
317 struct port_wakeup port3 = {
318     .pullup_down_enable = ENABLE,
319     .edge               = FALLING_EDGE,
320     .attribute          = BLUETOOTH_RESUME,
321     .imap               = IO_PORTB_00,
322     .filter_enable      = ENABLE,
323 };
324 struct port_wakeup port1 = {
325     .pullup_down_enable = ENABLE,
326     .edge               = FALLING_EDGE,
327     .attribute          = BLUETOOTH_RESUME,
328     .imap               = IO_PORTB_05,
329     .filter_enable      = DISABLE,
330 };
331 const struct sub_wakeup sub_wkup = {
332     .attribute          = BLUETOOTH_RESUME,
333 };
334 const struct charge_wakeup charge_wkup = {
335     .attribute          = BLUETOOTH_RESUME,
336     .imap               = IO_PORTB_04,
337 };
338 const struct wakeup_param wk_param = {
339     .filter             = PORT_FLT_2ms,
340     .port[1]            = &port0,
341     .port[2]            = &port1,
342 #if CONFIG_APP_AT_COM
343     .port[2]            = &port1,
344 #endif
345     .port[2]            = &port2,
346     .port[3]            = &port3,
347     .sub               = &sub_wkup
};

IO中断必须事先设置成唤醒口

```

```

1, struct port_wakeup port3 = {
2,     .pullup_down_enable = ENABLE,
3,     .edge              = FALLING_EDGE,
4,     .attribute          = BLUETOOTH_RESUME,
5,     .iomap              = IO_PORTB_00,
6,     .filter_enable      = DISABLE, ←
7};

void board_power_init(void)
{
    log_info("Power init : %s", __FILE__);

    //< close short key reset
    /* p33_and_1byte(P3_PR_PWR, ~BIT(3)); */
    //< close long key reset
    /* p33_and_1byte(P3_PINR_CON, 0); */

    power_init(&power_param);

    //< close long key reset
    VCM_DET_EN(0);

    power_set_callback(TCFG_LOWPOWER_LOWPOWER_SEL, sleep_enter_callback, sleep_exit_callback, board_set_sof);
    power_keep_dacvdd_en(0);
    power_wakeup_init(&wk_param);

    extern void port_edge_wkup_set_callback(void (*wakeup_callback)(u8 index, u32 gpio));
    port_edge_wkup_set_callback(port_edge_wakeup_isr); ←

/* #if !(TCFG_IKEY_ENABLE && !TCFG_AKEY_ENABLE) */
/* charge_check_and_set_pinr(0); */
/* #endif */
}
#endif

```

030之后，696X的工程需要加这

如果中断频率过快，需要改成把过滤设置成 DISABLE，若出现看门狗复位，主动在中断函数清狗

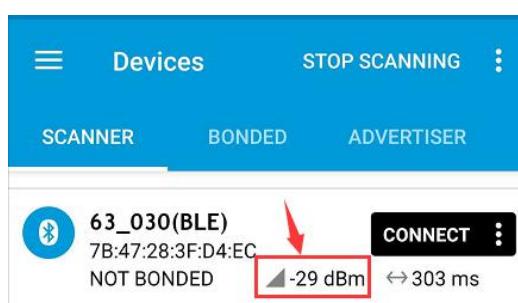
其他封装系列，可以参考这个试试，不行再联系深圳办

15. 动态调节功率 (020 之后) 20201130 xin

```

static void pwr_adjust(void *priv)
//范围0-9,nrf软件可以看功率值,正常范围在-30dB ~ -60dB
{
    static u8 pwr = 9;
    if(pwr) pwr--;
    struct ble_server_operation_t *test_opt;
    ble_get_server_operation_table(&test_opt);
    test_opt->adv_enable(NULL, 0);
    extern void bt_max_pwr_set(u8 pwr, u8 pg_pwr, u8 iq_pwr, u8 ble_pwr);
    bt_max_pwr_set(pwr, 5, 8, pwr); ←
    test_opt->adv_enable(NULL, 1);
    g_printf("rf config:%d", pwr);
}

```



16. 长按复位及高电平复位 (020 之后) 20201130 xin

```
//< close short key reset
power_mclr(0);
//< close long key reset
power_pin_reset(0);
/*sdpg_config(1);*/

power_set_callback(TCFG_LOWPOWER_LOWPOWER_SEL, sleep_enter_callback, sleep_exit_callback,
power_keep_dacvdd_en(0);

power_wakeup_init(&wk_param);

/* p33_and_1byte(P3_PR_PWR, ~BIT(3)); //关闭PB2短按复位 */
/* p33_or_1byte(P3_PR_PWR, 0x04); */

/* p33_tx_1byte(P3_PORT_SEL10, 0x11); //PB1长按复位 */
/* p33_tx_1byte(P3_PINR_CON, 0x31); //PB1长按复位 */

p33_tx_1byte(P3_PORT_SEL10, 0x12); //PB2长按复位
p33_tx_1byte(P3_PINR_CON, 0x31); //PB2长按复位

/* p33_or_1byte(P3_PINR_CON, BIT(2)); //高电平有效 */
```

只能开启一个IO做复位功能

没有设置默认是低复位

17. 631X 的 PB2 不能硬件接地或接到三极管基极 20201130 xin

631X 芯片默认 PB2 接地，会一直在复位会下载不了程序，软件修改不了，请硬件设计要避免
636X、635X、637X 是 PB1，也要避免使用这个脚去接地。

18. 630NSDK 050 版本 使用 APP (杰理 OTA 升级) 给芯片 br30 (637X) 升级失败问题处理方法 20201202 LAQ

- 根据下面图片处理,文件路径是 AC630N_bt_data_transfer_sdk_release_v0.5.0\ sdk\cpu\br30\ tools\ download.c

```

148 REM %OBJDUMP% -D -address-mask=0xffffffff -print-dbg $1.elf > $1.lst
149 %OBJCOPY% -O binary -j .text %ELFFILE% text.bin
150 %OBJCOPY% -O binary -j .data %ELFFILE% data.bin
151 %OBJCOPY% -O binary -j .data_code %ELFFILE% data_code.bin
152 %OBJCOPY% -O binary -j .overlay_aec %ELFFILE% aec.bin
153 %OBJCOPY% -O binary -j .overlay_aac %ELFFILE% aac.bin
154
155
156 %OBJDUMP% -section-headers -address-mask=0xffffffff %ELFFILE%
157 %OBJDUMP% -t %ELFFILE% > symbol_tbl.txt
158
159 copy /b text.bin+data.bin+data_code.bin+aec.bin+aac.bin app.bin
160
161 #ifdef CONFIG_BR30_C_VERSION
162 copy br30c_p11_code.bin p11_code.bin
163 copy br30c_ota.bin ota.bin
164 #else
165 copy br30_p11_code.bin p11_code.bin
166 //copy br30_ota.bin ota.bin
167 #endif /* #ifdef CONFIG_BR30_C_VERSION */
168
169 #if CONFIG_EARPHONE_CASE_ENABLE

```

2. 如图处理，文件路径是 AC630N_bt_data_transfer_sdk_release_v0.5.0\ sdk\cpu\br30\ tools\ download.bat

```

16 set ELFFILE=sdk.elf
17
18 REM %OBJDUMP% -D -address-mask=0xffffffff -print-dbg $1.elf > $1.lst
19 %OBJCOPY% -O binary -j .text %ELFFILE% text.bin
20 %OBJCOPY% -O binary -j .data %ELFFILE% data.bin
21 %OBJCOPY% -O binary -j .data_code %ELFFILE% data_code.bin
22 %OBJCOPY% -O binary -j .overlay_aec %ELFFILE% aec.bin
23 %OBJCOPY% -O binary -j .overlay_aac %ELFFILE% aac.bin
24
25 %OBJDUMP% -section-headers -address-mask=0xffffffff %ELFFILE%
26 %OBJDUMP% -t %ELFFILE% > symbol_tbl.txt
27
28 copy /b text.bin+data.bin+data_code.bin+aec.bin+aac.bin app.bin
29
30
31
32
33
34 copy br30_p11_code.bin p11_code.bin
35 copy br30_ota.bin ota.bin
36 copy app.bin bluetooth\standard\app.bin
37 copy br30loader.bin bluetooth\standard\br30loader.bin
38
39 bluetooth\standard\download.bat AD697N

```

3. 需要更换 ota.bin 文件,如果 AC630N_bt_data_transfer_sdk_release_v0.5.0\ sdk\cpu\br30\ tools\ 路径下面已经有 ota.bin 文件直接替换，如果没有 ota.bin 文件直接将文件粘贴进去即可

ota.bin 文件获取方法：链接：<https://pan.baidu.com/s/1JZDIpUnGb5ErNY6585SJ0w>
提取码：JLKJ

19. 637X 的 timer 时钟源不能选 OSC 晶振 20201230 xin

3-2	SSEL[1:0]	rw	SSEL1-0: timer 驱动源选择 注意: 选中的时钟需要使用上面 PSET 分频到(1sb_clk/2)以下! 00: 使用系统时钟作为 timer 的驱动源; 只能用这个 01: 使用 I0 口信号 CIN 作为 timer 的驱动源; 10: 使用 OSC 时钟作为 timer 的驱动源; 不能用 11: 使用 RC 时钟作为 timer 的驱动源。
-----	-----------	----	---

驱动文件: https://github.com/JieLi-Tech/fw-AC63_BT_SDK/issues/28

github.com/JieLi-Tech/fw-AC63_BT_SDK/issues/28

SE GNU TI course ST other JL homepage entertainment Mac

Open 关于637N的定时中断与PWM #28

maxbang5 opened this issue 28 days ago · 8 comments

lawrencejiabin commented 26 days ago

```
#include "includes.h"
#include "app_config.h"
#include "asm/cpu.h"
#include "asm/irq.h"
#include "asm/clock.h"
#include "system/timer.h"
#include "system/init.h"
#include "gpio.h"
```

20. OTA 升级需要关闭不必要的外设 20201230 xin

```
void rcsp_ch_update_init(void (*resume_hdl)(void *priv), int (*sleep_hdl)(void *priv))
{
    deg_puts("-----rcsp_ch_update_init\n");
    rcsp_update_resume_hdl_register(resume_hdl, sleep_hdl);
    //register_receive_fw_update_block_handle(rcsp_update_handle);
}
```

如果开了定时器中断，串口，PWM 等外设时，启动 APP OTA 升级前，需要把这些外设关闭，否则会影响到 OTA 升级时序，导致连接中断。

bit_clr_ie(IRQ_TIME3_IDX); //关闭中断使能, 参考此函数, 参数为中断号

21. OTA 升级需要定义获取 BLE 广播包、profile 数据的接口 20201230 xin

如果需要新建 BLE 的 profile 文件，不用 SDK 现有的 le_trans_data.c，需要在新建的文件，添加以下 3 个接口，用于返回数据给 OTA 的 uboot，才能确保升级的跳转的正常，否则会断连。

```

u8 *ble_get_scan_rsp_ptr(u16 *len)
{
    if (len) {
        *len = scan_rsp_data_len;
    }
    return scan_rsp_data; ← 返回自定义的数组
}

u8 *ble_get_adv_data_ptr(u16 *len)
{
    if (len) {
        *len = adv_data_len;
    }
    return adv_data;
}

u8 *ble_get_gatt_profile_data(u16 *len)
{
    *len = sizeof(profile_data);
    return (u8 *)profile_data; ← 返回profile数组
}

```

22. 双模共 MAC 地址有部分手机概率配对失败 20201230 xin

```

static void bt_function_select_init()
{
    /* __change_hci_class_type(0x240424); */
    __change_hci_class_type(0x10091c);

    __set_user_ctrl_conn_num(TCFG_BD_NUM);
    __set_support_msbc_flag(1);

    __change_hci_class_type(0x240424);
    __change_hci_class_type(0x10091c);
}

```

看需求任选一种

23. 查看代码异常复位等异常情况原因 20210105 xin

程序默认没有显示太多异常复位的原因，需要自己开启

```

#ifndef CONFIG_RELEASE_ENABLE
#define LIB_DEBUG 1 ←
#else
#define LIB_DEBUG 1
#endif

/// 异常中断，assert打印开启
#ifndef CONFIG_RELEASE_ENABLE
const int config_assert = 1; ←
#else
const int config_assert = 1;
#endif

```

```
const char log_tag_const_v_PMU AT(.LOG_TAG_CONST) = CONFIG_DEBUG_LIB(FALSE);
const char log_tag_const_i_PMU AT(.LOG_TAG_CONST) = CONFIG_DEBUG_LIB(TRUE); ↑
const char log_tag_const_d_PMU AT(.LOG_TAG_CONST) = CONFIG_DEBUG_LIB(FALSE);
const char log_tag_const_w_PMU AT(.LOG_TAG_CONST) = CONFIG_DEBUG_LIB(TRUE); ↑
const char log_tag_const_e_PMU AT(.LOG_TAG_CONST) = CONFIG_DEBUG_LIB(TRUE); ↑
const char log_tag_const_c_PMU AT(.LOG_TAG_CONST) = CONFIG_DEBUG_LIB(TRUE);
```

```
#define __LOG_ENABLE

#ifndef CONFIG_RELEASE_ENABLE
#define __LOG_LEVEL 0//0xff
#endif ↑
```

调试时确保这4个开启，就能看到详细的死机原因，生产时，记得关闭!!!!!!

```
[00:00:00.100][Debug]: [CLOCK]CACHE_CON : 60F23100
[00:00:00.100][Info]: [MISC]--Reset Source : 0x1
[00:00:00.100][Info]: [MISC]P33 RESET
[00:00:00.100][Info]: [PMU]--P3 Reset Source : 0x4
[00:00:00.100][Info]: [PMU]WDT ↑
[00:00:00.100]==version check=====
[00:00:00.100]=======
```

```
XX
0x5A5A5A5A 0x5A5A5A5A 0xFFFFFFFF
usp : 0x00201600
ssp : 0x00004A70
sp : 0x00004960
Stack : 0x00004A70
rets : 0x01E0CBDE
reti : 0x01E08118
retx : 0x00000000
rete : 0x00000000
psr : 0x00000000
icfg : 0x07010284
          |-----|-----|-----|-----|-----|-----|-----|-----|
          0x01E08050 ->0x01E080B4 ->0x01E080D0 ->0x01E080EE
```

出现此种现象，不能再改动代码，
需要提供对应的下载目录里面的sdk.elf

\Test\AC630N_bt_data_transfer_sdk_release_v0.6.0\c-br25\ sdk\cpu\br25\tools			
名称	修改日期	类型	大小
mp30.bin	2020/12/23 9:41	FTE Binary Exported	0 KB
ota.bin	2020/12/15 19:49	FTE Binary Exported	171 KB
ota_lrc.bin	2020/12/15 19:49	FTE Binary Exported	160 KB
remove_tailing_zeros.exe	2021/1/5 14:11	应用程序	239 KB
rom.lst	2020/12/15 19:49	LST 文件	714 KB
run_jtag.sh	2020/12/15 19:49	Shell Script	1 KB
sdk.elf ←	2020/12/16 15:04	ELF 文件	2,045 KB
sdk.elf.0.0.preopt.bc	2020/12/16 15:04	BC 文件	7,885 KB

24. 出现电压检测、ADC 检测不准 20210105 xin

```
void app_main()
{
    struct int16 it;

    log_info("app_main\n");
    extern u16 get_vbat_trim();
    extern u16 get_vbg_trim();
    r_printf("vbat_trim:%d...vbg_trim:%d",get_vbat_trim(),get_vbg_trim());
    r_printf("cpu_reset_by_soft:%d",cpu_reset_by_soft());
    r_printf("is_ldo5v_wakeup:%d",is_ldo5v_wakeup());
```

```
log_info("app_main\n");
extern u16 get_vbat_trim();
extern u16 get_vbg_trim();
printf("vbat_trim:%d...vbg_trim:%d",get_vbat_trim(),get_vbg_trim());
添加以上代码，看打印值，如果为 15 跟 63，那就要用烧录器先烧录一遍。
```

25. 636N 区分复位跟唤醒 20210111 xin

获取复位源信息，接口只能调用一次，保存在静态变量中，以后都判断变量的。

```
power_reset_src = power_reset_source_dump();
r_printf("power reset src:0x%x", power_reset_src);
```

VDDIO POR	->	power_reset_src:0x1	上电复位
VDDIO LVD	->	power_reset_src:0x2	上电复位或低电复位
WDT	->	power_reset_src:0x4	看门狗复位
VCM	->	power_reset_src:0x8	硬复位
PPINR	->	power_reset_src:0x10	按键长按复位

上电复位需要用时判断 VDDIO POR 和 VDDIO LVD。

开启按键长按复位功能，是在下载目录可以配置功能的。

```
#01BD-10000000; //uboot中口没有
UTRX=PB05;串口升级[PB00 PB05 PA05]
RESET=PB01_08_0; //port口_长按时间_有效电平(长按时间)
UPDATE_JUMP=0;
- 2.7k isd_config.ini Conf[Unix] ⓘ
```

获取 IO 口唤醒源信息，也只能调用一次，需要自己定义静态变量来保存。

```
reset_source_dump();

power_reset_src = power_reset_source_dump();
r_printf("power reset_src:0x%x", power_reset_src);
r_printf("R3 WKUP PND:0x%x", P33_CON_GET(R3_WKUP_PND));

request_irq(1, 2, exception_irq_handler, 0); 只能放这里,自己定义变量
保存
debug_init();
4.8k setup.c<sdk> C/1 AC@ ⓘ
```

最多可以配置 7 个 IO 口的唤醒，任意 IO 均可。

```

struct port_wakeup port0 = {
    .pullup_down_enable = ENABLE,
    .edge                = FALLING_EDGE,
    .attribute           = BLUETOOTH_RESUME,
    .iomap               = TCFG_IKEY_POWER_ONE_PORT,
    .filter_enable       = ENABLE,
};

struct port_wakeup port1 = {
    .pullup_down_enable = ENABLE,
    .edge                = FALLING_EDGE,
    .attribute           = BLUETOOTH_RESUME,
    .iomap               = TCFG_IKEY_PREV_ONE_PORT,
    .filter_enable       = ENABLE,
};

struct port_wakeup port2 = {
    .pullup_down_enable = ENABLE,
    .edge                = FALLING_EDGE,
    .attribute           = BLUETOOTH_RESUME,
    .iomap               = TCFG_IKEY_NEXT_ONE_PORT,
    .filter_enable       = ENABLE,
};

const struct sub_wakeup sub_wkup = {
    .attribute = BLUETOOTH_RESUME,
};

const struct charge_wakeup charge_wkup = {
    .attribute = BLUETOOTH_RESUME,
};

const struct wakeup_param wk_param = {
    .filter      = PORT_FLT_2ms,
    .port[1]     = &port0,   ← 对应bit(1)
    .port[2]     = &port1,   bit(2)
    .port[3]     = &port2,   bit(3)
    .sub        = &sub_wkup,
    .charge     = &charge_wkup,
};

```

例子，定义3个IO唤醒

```

[]>[]>[]>[]>[]>[]>[]>[]>[]>[]>[]>[]>[]
[00:03:30.311]bv:472, bl:9 , check_vbat:0
[00:03:30.323]bv:472, bl:9 , check_vbat:0
<>[]>[]>[]>[]>[]>[]>[]>[]>[]>[]>[]>[]>[]
[]>[]>[]>[]>[]>[]>[]>[]>[]>[]>[]>[]>[]
>[]>[]>[]>[]>[]>[]>[]>[]>[]>[]>[]>[]
[00:04:00.321]bv:472, bl:9 , check_vbat:0
[00:04:00.333]bv:472, bl:9 , check_vbat:0
<>[]>[]>[]>[]>[]>[]>[]>[]>[]>[]>[]>[]
[]>[]>[]>[]>[]>[]>[]>[]>[]>[]>[]>[]
>[]>[]>[]>[]>[]>[]>[]>[]>[]>[]>[]
[00:04:30.321]bv:472, bl:9 , check_vbat:0
[00:04:30.333]bv:472, bl:9 , check_vbat:0
<>[]>[]>[]>[]>[]>[]>[]>[]>[]>[]>[]
[]>[]>[]>[]>[]>[]>[]>[]>[]>[]>[]
[00:04:49.642]app_key_evnet: 0,2
[00:04:49.642][Info]: [SPP_AND_LE]set_soft_poweroff
[00:04:49.643]mode_en:0
[00:04:49.644]adv_en:0
[00:04:49.644]ble_work_st:20->2
[00:04:49.947][Info]: [PMU]--Enter Power off(Soft)

```

判断是否内置充电唤醒。

```

0
1 void app_main()
2 {
3     struct intent it;
4
5     log_info("app_main\n");
6     r_printf("is_ldo5v_wakeup:%d", is_ldo5v_wakeup());
7

```

这个接口需要放到这边判断

```

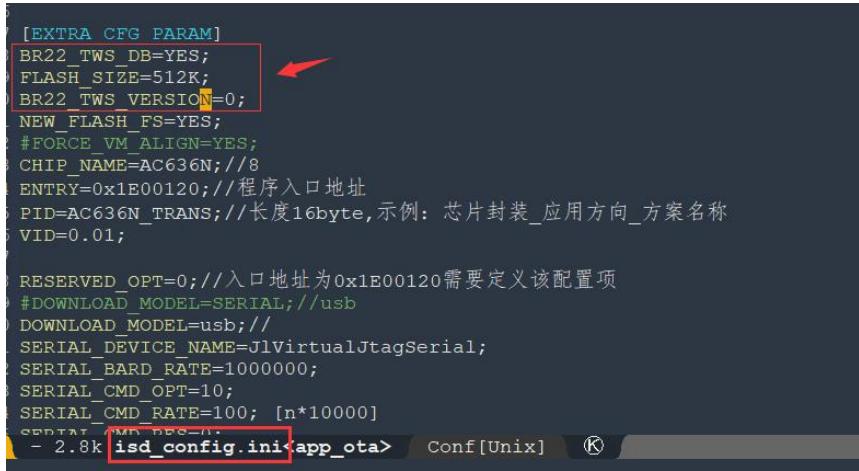
[00:00:00.269][Info]: [APP]app_main
[00:00:00.270]is_ldo5v_wakeup:8

```

26. APP OTA 双备份升级(只能用于 4Mbits 以上的芯片) 20210114 xin

c-br25-OTA-double > sdk > cpu > br25 > tools > bluetooth > app_ota

名称	修改
AC696X_config_tool	2021,
app.bin	2021,
br25loader.bin	2020,
cfg_tool.bin	2020,
config.dat	2021,
download.bat	2021,
isd_config.ini	2021,
jl.isd.bin	2021,



```
[EXTRA CFG PARAM]
BR22_TWS_DB=YES;
FLASH_SIZE=512K;
BR22_TWS_VERSION=0; // Red arrow points here
NEW_FLASH_FS=YES;
#FORCE_VMS_ALIGN=YES;
CHIP_NAME=AC636N;//8
ENTRY=0x1E00120;//程序入口地址
PID=AC636N_TRANS;//长度16byte,示例：芯片封装_应用方向_方案名称
VID=0.01;

RESERVED_OPT=0;//入口地址为0x1E00120需要定义该配置项
#DOWNLOAD_MODEL=SERIAL;//usb
DOWNLOAD_MODEL=usb;;
SERIAL_DEVICE_NAME=J1VirtualJtagSerial;
SERIAL_BARD_RATE=1000000;
SERIAL_CMD_OPT=10;
SERIAL_CMD_RATE=100; [n*10000]
SERIAL_MAX_BAUD=2_8k
isd_config.ini app_ota > Conf[Unix] ⓘ
```



```
//是否采用双备份升级方案:0-单备份;1-双备份
const int support_dual_bank_update_en = 1;
```

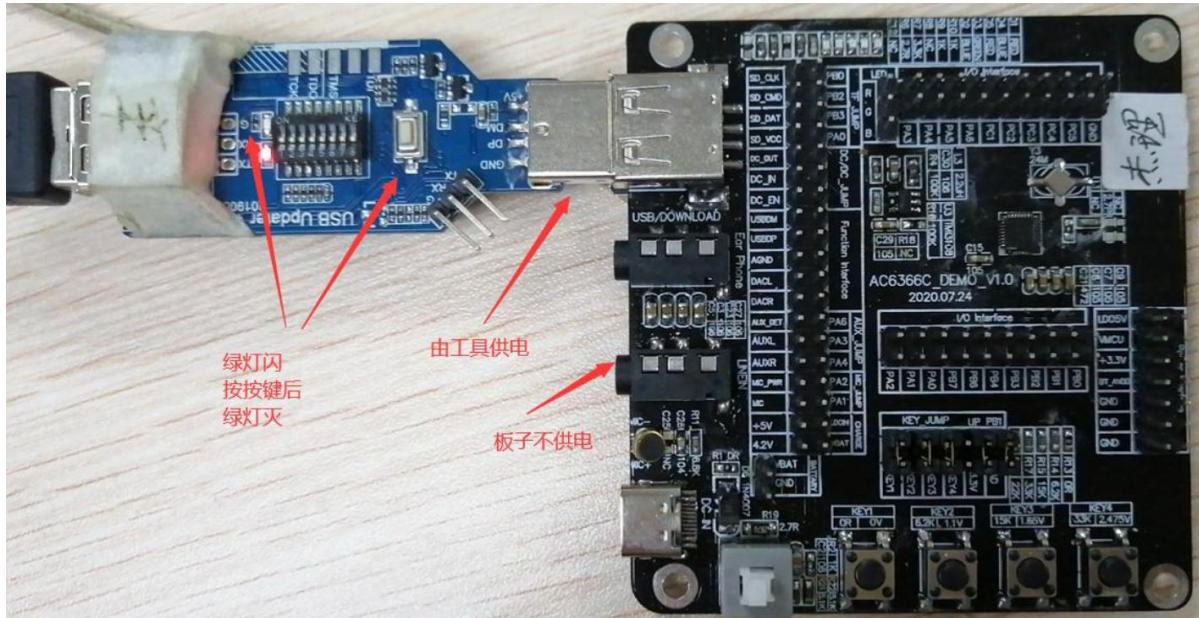
27. 6351D PR0 和 PR1 当普通 IO 口使用 20210121 xin

```
int rtc_port_pr_in(u8 port);
int rtc_port_pr_read(u8 port);
int rtc_port_pr_out(u8 port, bool on);
int rtc_port_pr_hd(u8 port, bool on);
int rtc_port_pr_pu(u8 port, bool on);
int rtc_port_pr_pd(u8 port, bool on);
int rtc_port_pr_die(u8 port, bool on);
#define IO_PORTR_00 0
#define IO_PORTR_01 1
#define OUT_LOW 0
#define OUT_HIGH 1

p33_tx_1byte(R3_OSL_CON, 0);
rtc_port_pr_die(0, 1);
rtc_port_pr_die(1, 1);
rtc_port_pr_out(IO_PORTR_00, OUT_LOW);
rtc_port_pr_out(IO_PORTR_01, OUT_LOW);
rtc_port_pr_out(IO_PORTR_00, OUT_HIGH);
rtc_port_pr_out(IO_PORTR_01, OUT_HIGH);
```

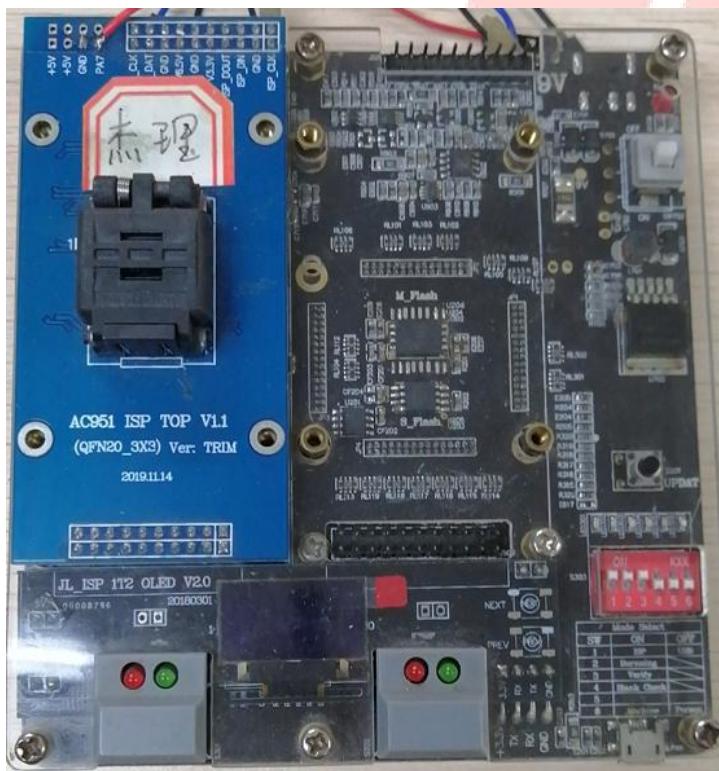
不支持仿真，没有断点执行方式。

只能用串口的打印方式，串口随便一个 IO 都可以，软件默认做了映射，串口波特率必须 1000000。
开发时烧录是需要硬件工具，板子断电的时候，插入工具，板子由工具供电



生产烧录，由 2 种工具：

1.1 拖 2 烧录器，先烧芯片，再贴板

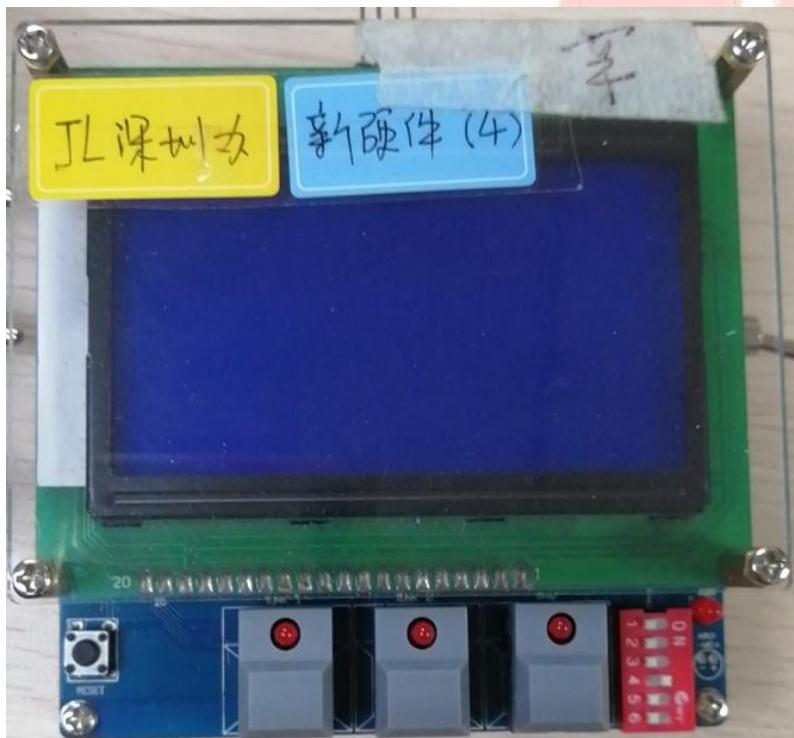


2.1 拖 8 工具烧录，先贴片，预留 USB 测试点，需用测试架搭配接线烧录

fw 文件需要使用"固件文件烧写授权工具_1.2.9.exe"授权后才能使用 1 拖 8 工具



产线升级，使用测试盒工具，可以测试频偏和代码升级



26. SPP 开启 pin_code 功能 20210125 xin

```
const char *bt_get_pin_code()
{
    return "6688";
    return "0000";
}
```

```

static void bt_function_select_init()
{
    __set_user_ctrl_conn_num(TCFG_BD_NUM);
    __set_support_msbc_flag(1);
#ifndef BT_SUPPORT_DISPLAY_BAT
    __bt_set_update_battery_time(60);
#else
    __bt_set_update_battery_time(0);
#endif
    __set_page_timeout_value(8000); /*回连搜索时间长度设置,可使用该函数注册使用, ms单位,u16*/
    __set_super_timeout_value(8000); /*回连时超时参数设置。ms单位。做主机有效*/
#if (TCFG_BD_NUM == 2)
    __set_auto_conn_device_num(2);
#endif
#ifndef BT_SUPPORT_MUSIC_VOL_SYNC
    vol_sys_tab_init();
#endif

//io_capabilities ; /*0: Display only 1: Display YesNo 2: KeyboardOnly 3: NoInputNoOutput*/
//authentication_requirements: 0:not protect 1 :protect
    __set_simple_pair_param(3, 0, 2);
    extern void __set_simple_pair_flag(u8 flag);
    __set_simple_pair_flag(0);/*提供接口外部设置配对方式,1使能简易配对。0使用pin code*/
}

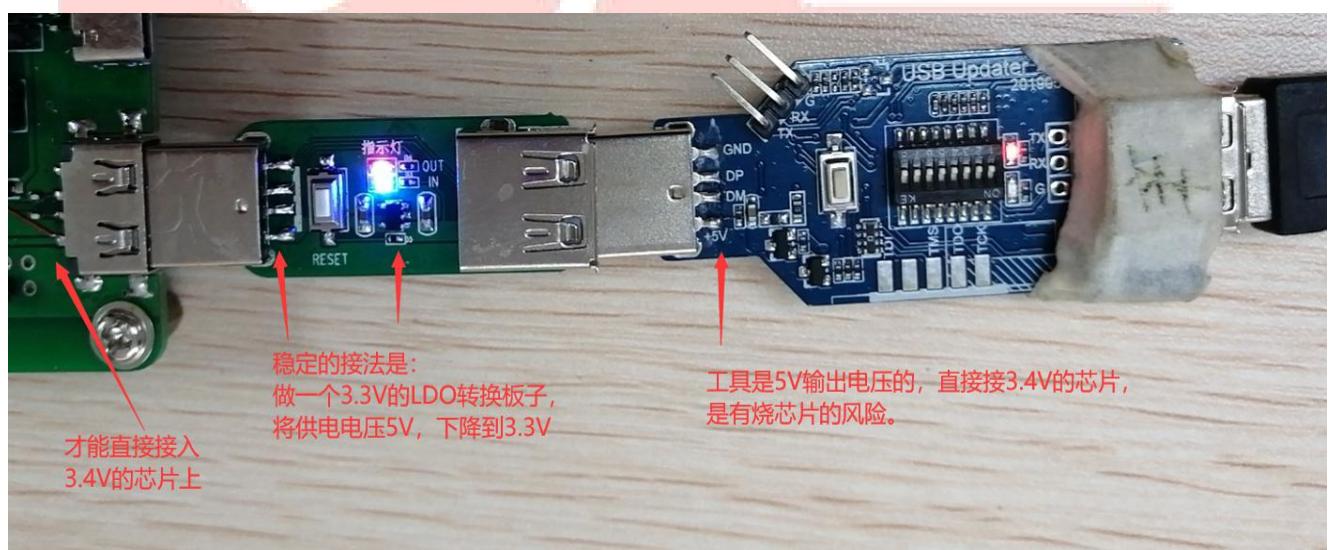
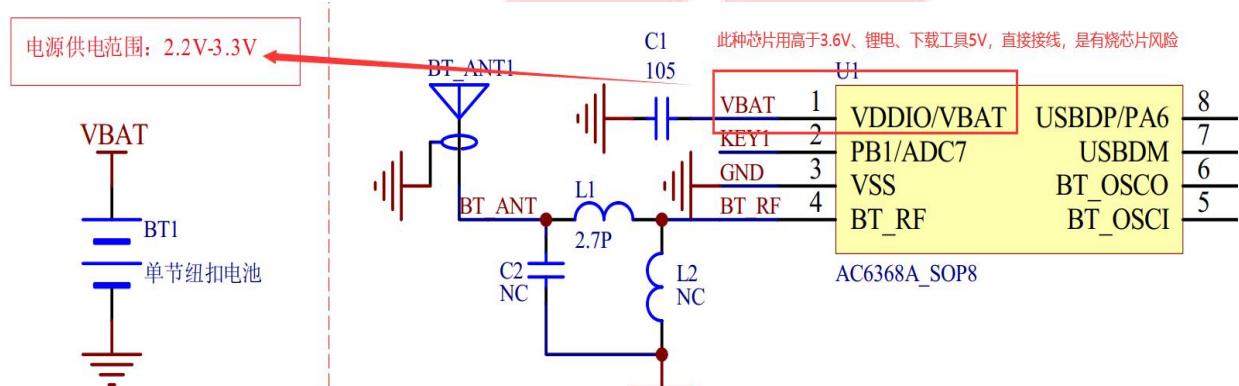
```

extern void __set_simple_pair_flag(u8 flag);
 __set_simple_pair_flag(0);/*提供接口外部设置配对方式,1使能简易配对。0 使用 pin code*/

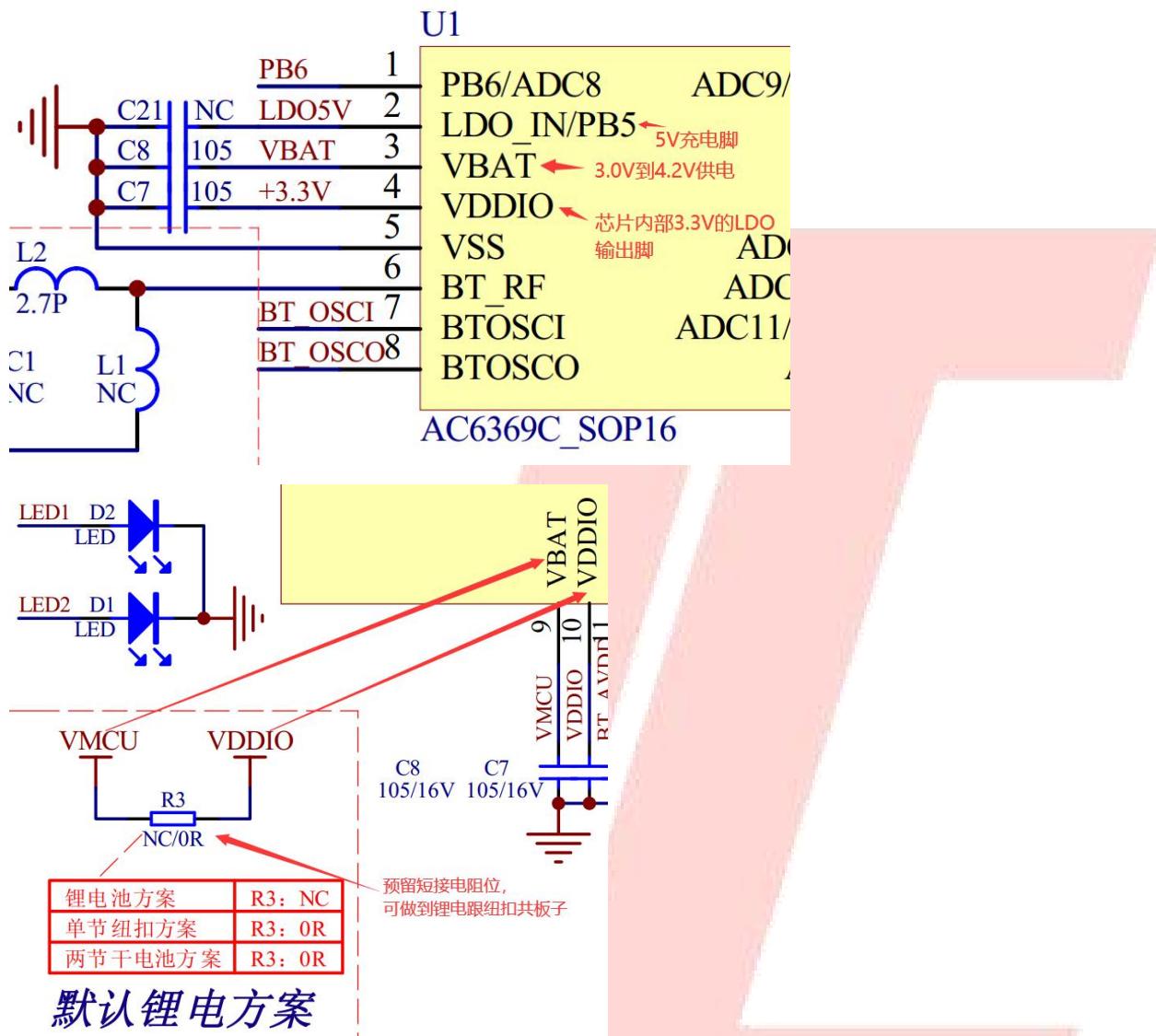
27. 芯片供电范围及防烧芯片措施 20210125 xin

芯片分 2 种供电方式:

第一种: 3.4V 以下供电, 适用 2 节干电池, 纽扣电池的方案。芯片有 AC6368A、AC6369F。



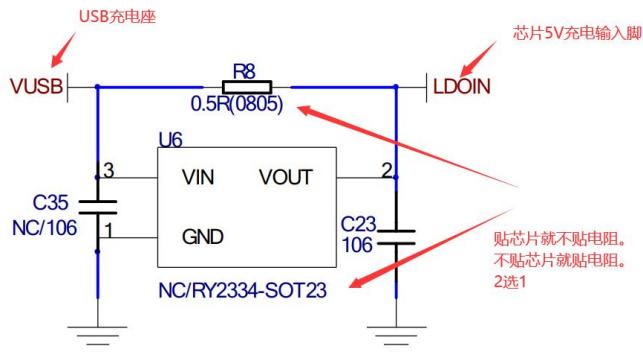
第二种：3.0V - 5V 之间供电，适用锂电池方案。除第一种之外，大部分都是此种接法。
但是硬件电路预留短接电阻，也可变成第一种供电方式。



28. 内置充电功能及预留过压过流保护 20210125 xin

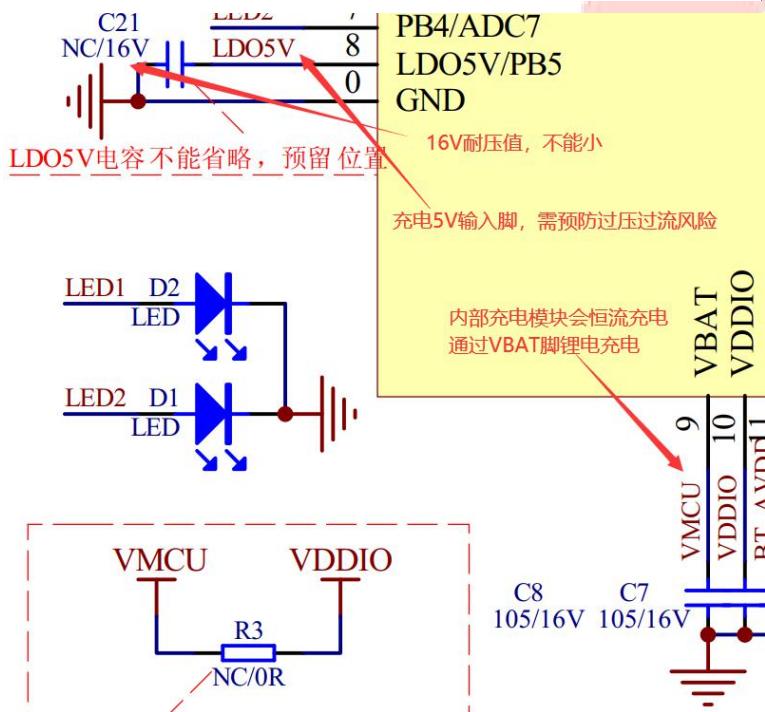
以 AC6366C 为例：芯片 LDOIN 引脚没有过流过压保护，使用的充电器有大于 5.5V 输出的情况(如快充充电器等)，均需预留过压过流保护 IC

Table 2-2						
Symbol	Parameter	Min	Typ	Max	Unit	
VBAT	Voltage Input	2.2	3.7	4.5	V	
LDOIN	Charger Voltage	4.5	5	5.5	V	



过压过流保护

USB



```
***** CHARGE config *****/
#ifndef TCFG_CHARGE_ENABLE
#define TCFG_CHARGE_ENABLE
CHARGE_PLATFORM_DATA_BEGIN(charge_data)
    .charge_en          = TCFG_CHARGE_ENABLE,           // 内置充电使能
    .charge_poweron_en  = TCFG_CHARGE_POWERON_ENABLE,   // 是否支持充电开机
    .charge_full_v      = TCFG_CHARGE_FULL_V,          // 充电截止电压
    .charge_full_ma     = TCFG_CHARGE_FULL_MA,         // 充电截止电流
    .charge_ma          = TCFG_CHARGE_MA,              // 充电电流
/* ldo5v 拔出过滤值，过滤时间 = (filter*2 + 20)ms, ldo5v<0.6V且时间大于过滤时间才认为拔出
对于充满直接从5V掉到0V的充电仓，该值必须设置成0，对于充满由5V先掉到0V之后再升压到xV的
充电仓，需要根据实际情况设置该值大小 */
    .ldo5v_off_filter = 0,
/* ldo5v的10k下拉电阻使能，若充电舱需要更大的负载才能检测到插入时，请将该变量置1，默认值设置为1
对于充电舱需要按键升压，且维持电压是从充电舱经过上拉电阻到充电口的舱，请将该值改为0*/
    .ldo5v_pulldown_en = 0,
CHARGE_PLATFORM_DATA_END()
#endif // TCFG_CHARGE_ENABLE

static void board_devices_init(void)
{
#if TCFG_PWMLED_ENABLE
    pwm_led_init(&pwm_led_data);
#endif
}
```

```

void board_init()
{
    board_power_init();
    adc_vbg_init();
    adc_init();
    cfg_file_parse(0);

    devices_init();

    board_devices_init();

    log_info("board_init");

    //必须在切换到dcdc前设置好
    if(get_charge_online_flag()){
        log_info("charge...select LDO15");
        power_set_mode(PWR_LDO15);
    }else{
        power_set_mode(TCFG_LOWPOWER_POWER_SEL);
    }
#if TCFG_CHARGE_ENABLE
    extern int charge_init(const struct dev_node *node, void *arg);
    charge_init(NULL, (void *)&charge_data);
#endif

#ifndef TCFG_CHARGE_ENABLE
    /*close FAST CHARGE */
    CHARGE_EN(0);
    CHGBG_EN(0);
#endif
}

```

***** CHARGE config*****

```

#if TCFG_CHARGE_ENABLE
CHARGE_PLATFORM_DATA_BEGIN(charge_data)
.charge_en          = TCFG_CHARGE_ENABLE,           //内置充电使能
.charge_poweron_en = TCFG_CHARGE_POWERON_ENABLE,   //是否支持充电开机
.charge_full_V      = TCFG_CHARGE_FULL_V,          //充电截止电压
.charge_full_mA     = TCFG_CHARGE_FULL_MA,         //充电截止电流
.charge_mA          = TCFG_CHARGE_MA,              //充电电流
/*ldo5v 拔出过滤值, 过滤时间 = (filter*2 + 20)ms,ldoin<0.6V 且时间大于过滤时间才认为拔出
对于充满直接从 5V 掉到 0V 的充电仓, 该值必须设置成 0, 对于充满由 5V 先掉到 0V 之后再升压到 xV 的
充电仓, 需要根据实际情况设置该值大小*/
.ldo5v_off_filter = 0,
/*ldo5v 的 10k 下拉电阻使能,若充电舱需要更大的负载才能检测到插入时, 请将该变量置 1,默认值设置为 1
对于充电舱需要按键升压,且维持电压是从充电舱经过上拉电阻到充电口的舱, 请将该值改为 0*/
.ldo5v_pulldown_en = 0,
CHARGE_PLATFORM_DATA_END()
#endif//TCFG_CHARGE_ENABLE

```

```

#if TCFG_CHARGE_ENABLE
extern int charge_init(const struct dev_node *node, void *arg);
charge_init(NULL, (void *)&charge_data);
#endif

```

```

//***** 充电参数配置 *****
//是否支持芯片内置充电          ENABLE_THIS_MODULE
#define TCFG_CHARGE_ENABLE      DISABLE   ← 开启内置充电
//是否支持开机充电
#define TCFG_CHARGE_POWERON_ENABLE DISABLE
//是否支持拔出充电自动开机功能
#define TCFG_CHARGE_OFF_POWERON_NE DISABLE
/*
充电截止电压可选配置：
    CHARGE_FULL_V_3869  CHARGE_FULL_V_3907  CHARGE_FULL_V_3946  CHARGE_FULL_V_3985
    CHARGE_FULL_V_4026  CHARGE_FULL_V_4068  CHARGE_FULL_V_4122  CHARGE_FULL_V_4157
    CHARGE_FULL_V_4202  CHARGE_FULL_V_4249  CHARGE_FULL_V_4295  CHARGE_FULL_V_4350
    CHARGE_FULL_V_4398  CHARGE_FULL_V_4452  CHARGE_FULL_V_4509  CHARGE_FULL_V_4567
*/
#define TCFG_CHARGE_FULL_V      CHARGE_FULL_V_4202 ← 恒流转恒压的输出电压
/*
充电截止电流可选配置：
    CHARGE_FULL_mA_2       CHARGE_FULL_mA_5       CHARGE_FULL_mA_7       CHARGE_FULL_mA_10
    CHARGE_FULL_mA_15       CHARGE_FULL_mA_20      CHARGE_FULL_mA_25      CHARGE_FULL_mA_30
*/
#define TCFG_CHARGE_FULL_MA    CHARGE_FULL_mA_10 ← 恒压充电状态，电流下降到多少，认为
/*
充满，电池容量大，需要调大
    CHARGE_mA_20           CHARGE_mA_30         CHARGE_mA_40         CHARGE_mA_50
    CHARGE_mA_60           CHARGE_mA_70         CHARGE_mA_80         CHARGE_mA_90
    CHARGE_mA_100          CHARGE_mA_110        CHARGE_mA_120        CHARGE_mA_140
    CHARGE_mA_160          CHARGE_mA_180        CHARGE_mA_200        CHARGE_mA_220
*/
#define TCFG_CHARGE_MA        CHARGE_mA_60 ← 恒流充电的充电电流，一般按照1C的
//***** USB 配置 *****
//***** TCFG_DC_CHARGE *****

```

29. 636X 系列 5 路 PWM 20210125 xin

636X 芯片，没有 mcpwm 模块，只有 6 个 timer 定时器，除掉 timer1 被用做系统定时器，剩余的 5 个定时器，最多可以输出 5 路 PWM。由于映射通道只有 3 路，其中一路用作打印口，剩余 2 路可以映射。

所以，其中 3 路只能固定 IO 输出，2 路可以任意 IO 输出。

以 AC6369F 为例：

```

static void set_timer2_pwm(u32 fre, u8 duty)
{
#define OSC_HZ 24000000
    JL_TIMER2->CON = 0x0000;
    JL_TIMER2->CNT = 0x0000;
    JL_TIMER2->PRD = OSC_HZ / (4 * fre);
    JL_TIMER2->PWM = (JL_TIMER2->PRD * duty) / 100;
    JL_TIMER2->CON = (1 << 4) | (1 << 0) | (2 << 2); // | (1 << 8)去掉 bit8 关闭默认 IO 输出
#define T2_USE_CHANNEL_1 //可选配 0/1/2
#define PWM2_IO          IO_PORTC_04
#if(T2_USE_CHANNEL == 1)
    SFR(JL_IOMAP->CON3, 20, 4, 14); //OUTPUT_CH1
    gpio_set_hd(PWM2_IO, 0);
    gpio_set_pull_up(PWM2_IO, 1);
    gpio_set_pull_down(PWM2_IO, 1);
    gpio_direction_output(PWM2_IO, 0);
    gpio_set_die(PWM2_IO, 1);
#endif
}
static void set_timer4_pwm(u32 fre, u8 duty)
{
#define OSC_HZ 24000000
/* bit_clr_ie(IRQ_TIME4_IDX); */
    JL_TIMER4->CON = 0x0000;
    JL_TIMER4->CNT = 0x0000;
    JL_TIMER4->PRD = OSC_HZ / (4 * fre);
    JL_TIMER4->PWM = (JL_TIMER4->PRD * duty) / 100;
    JL_TIMER4->CON = (1 << 4) | (1 << 0) | (2 << 2); // | (1 << 8)去掉 bit8 关闭默认 IO 输出
#define T4_USE_CHANNEL_2 //可选配 0/1/2
#define PWM4_IO          IO_PORTA_00
#if(T4_USE_CHANNEL == 2)
    SFR(JL_IOMAP->CON3, 24, 4, 12); //OUTPUT_CH2
    gpio_set_hd(PWM4_IO, 0);
    gpio_set_pull_up(PWM4_IO, 1);
    gpio_set_pull_down(PWM4_IO, 1);
    gpio_direction_output(PWM4_IO, 1);
    gpio_set_die(PWM4_IO, 0);
#endif
}

```

2路映射的PWM的写法

```

void five_pwm_set(void)
{
    timer_pwm_init(JL_TIMER0, 10000, 2000, IO_PORTA_05, 0); ← 3路固定
    timer_pwm_init(JL_TIMER3, 10000, 5000, IO_PORTB_05, 0);
    timer_pwm_init(JL_TIMER5, 10000, 8000, IO_PORTB_07, 0);
    set_timer2_pwm(20000, 10);
    set_timer4_pwm(20000, 50); ← 2路映射
}

```

30. 注册软件定时器不耗费硬件定时器 20210202 xin

系统默认使用了 timer1 为系统 2ms 时钟定时用，所有的软件定时器都基于这个 timer1 拓展出来。

拓展出 2 种定时器使用方式：

1. 在中断函数回调执行(切记不可调用延时或耗时等操作)

sys_hi_timer_add() 定时循环函数，会导致不进低功耗，直到主动删除

sys_hi_timeout_add() 定时超时执行一次函数

sys_s_hi_timer_add() 定时循环函数，不影响系统进低功耗，周期会变，**建议使用此种定时器**

sys_s_hi_timeout_add() 定时超时执行一次函数

注意点：此种定时器，注册的函数是在**中断函数里面回调的**，不可添加延时，或耗时的操作。

单位是 ms，但是是以 2ms 步进的。3ms 等同于 4ms 的。

2. 在系统线程中执行，几乎可执行所有的操作

sys_timer_add() 定时循环函数，**会影响下次进低功耗的时间点，周期太短会影响功耗。**

sys_timeout_add() 定时超时执行一次函数

注意点：此种定时器，**注册的函数是在线程执行的**，优先级依赖于注册的线程的优先级。

单位是 ms，但是是以 10ms 步进的。5ms 等同于 10ms 的。系统是以 10ms 为系统滴答的。

//循环定时器写法

```

//软件定时器使用规范

static u16 timer_loop = 0;

static void timer_loop_func(void *priv)
{
    static u8 cnt = 0;
    cnt++;
    log_info("%s[%d]", __func__, cnt);
    if(cnt > 3) {
        if(timer_loop) {
            sys_timer_del(timer_loop); //删除前要判断
            timer_loop = 0; //删除后记得清0
        }
    }
}

void timer_loop_init(void)
{
    log_info("%s[%d]", __func__, timer_loop);
    if(timer_loop) {
        sys_timer_del(timer_loop); //注册前，要判断是否注册过
        timer_loop = 0;
    }
    timer_loop = sys_timer_add(NULL, timer_loop_func, 5 * 1000);
}

```

//延时定时器写法

```

static u16 timer_one = 0;

static void timer_one_func(void *priv)
{
    static u8 cnt = 0;
    cnt++;
    log_info("%s[%d]", __func__, cnt);
    //sys_timeout_add注册的只会执行一次,底层会自动删除
    //回调函数内不可调用sys_timeout_del
    timer_one = 0;//ID号清0,必须要看!! ! ←
}

void timer_one_init(void)
{
    log_info("%s[%d]", __func__, timer_one);
    if(timer_one){
        sys_timeout_del(timer_one); //注册前,要判断是否注册过
        timer_one = 0;
    }
    timer_one = sys_timeout_add(NULL, timer_one_func, 5 * 1000);
}

void timer_one_del(void)
{
    log_info("%s[%d]", __func__, timer_one);
    if(timer_one){
        sys_timeout_del(timer_one);
        timer_one = 0;
    }
}

```

31. 固件加 key 用于烧录 20210202 xin

芯片空片不需要加 key, 就能下载更新程序, 若被烧录过, 则调试的程序也要一起加 key。

AC630N_bt_data_transfer_sdk_release_v0.7.0 > c-br25 > sdk > cpu > br25 > tools > bluetooth > standard >			
名称	修改日期	类型	大小
AC696X_config_tool	2021/1/13 16:27	文件夹	
128_深圳办AC690x-813B.key	2021/2/10 14:26	KEY 文件	1 KB
app.bin	2021/1/31 14:26	FTE Binary Expo...	234 KB
br25loader.bin	2021/1/8 15:45	FTE Binary Expo...	29 KB
cfg_tool.bin	2021/1/32 10:11	FTE Binary Expo...	1 KB
download.bat	2021/2/2 10:15	Windows 批处理...	2 KB
isd_config.ini	2021/1/8 15:45	配置设置	4 KB
jl_isd.bin	2021/1/31 14:26	FTE Binary Expo...	272 KB
jl_isd/fw	2021/1/31 14:26	FW 文件	70 KB
ota.bin	2021/1/8 15:45	FTE Binary Expo...	171 KB
script.ver	2021/1/22 10:11	VER 文件	1 KB
tone.cfg	2021/1/8 15:45	CFG 文件	34 KB
uboot.boot	2021/1/8 15:45	BOOT 文件	4 KB
update.ufw	2021/1/31 14:26	UFW 文件	720 KB

```
programme.txt tmp.txt download.bat
```

```
@echo off
cd %~dp0
copy ..\..\uboot.boot .
copy ..\..\ota.bin .

..\..\isd_download.exe -tonorflash -dev br25 -boot 0x12000 -div8 _wait 300 -uboot uboot.boot
-app app.bin cfg_tool.bin -res tone.cfg %1 -uboot_compress -key 128_深圳办AC690x-813B.key
:: -format all
:::reboot 2500
```

-key固定，后面加key文件文件名

32. 测试盒用于测试 BLE 遥控器的按键 20210203 xin

```
static int bt_hci_event_handler(struct bt_event *bt)
{
    //对应原来的蓝牙连接上断开处理函数 ,bt->value=reason
    r_printf("-----bt_hci_event_handler reason %x %x", bt->event, bt->value);

    if (bt->event == HCI_EVENT_VENDOR_REMOTE_TEST) {
        if (0 == bt->value) {
            set_remote_test_flag(0);
            log_info("clear_test_box_flag");
            return 0;
        } else {

#ifndef TCFG_USER_BLE_ENABLE
            //1:edr con;2:ble con;
            if (1 == bt->value) {
                extern void bt_ble_adv_enable(u8 enable);
                bt_ble_adv_enable(0);
            } else if (2 == bt->value) {
                set_remote_test_flag(1);
            }
#endif
        }
    }
}

static void app_key_event_handler(struct sys_event *event)
{
    /* u16 cpi = 0; */
    u8 event_type = 0;
    u8 key_value = 0;

    if (event->arg == (void *)DEVICE_EVENT_FROM_KEY) {
        event_type = event->u.key.event;
        key_value = event->u.key.value;
        printf("app_key_evnet: %d,%d\n", event_type, key_value);
        void ble_server_send_test_key_num(u8 key_num);
        ble_server_send_test_key_num(key_value);
```

/* app_key_deal_test(event_type,key_value); */

```
void ble_server_send_test_key_num(u8 key_num)
{
    printf("%s[%d %d %d]", __func__, con_handle, key_num, get_remote_test_flag());
    if (con_handle) {
        if (get_remote_test_flag()) {
            ble_op_test_key_num(con_handle, key_num);
        } else {
            log_info("-not conn testbox\n");
        }
    }
}
```

测试盒可以连接 BLE 设备，可以显示频偏跟功率值。



遥控器按下按键，测试盒就可以显示按键值出来。



33. 630 v070 版本 SDK, PC 端定时通过 AT 指令将数据发送给 ble 主机, ble 主机也定时发送数据给 ble 从机, 从机再将数据发送给 PC, 过一段时间后 ble 从机会复位的处理方法 LJW

```

act630n_demo.c
custom.cfg.c
at_uart.c
at_cmds.c
uart.dev.c
uart.dev.h
app_at_com.c
le_at_com.c

// at_uart.c
313     _this->config.dev_name = "at-uart";
314
315     _this->udev = 0;
316     _this->data_length = 0;
317     _this->dbuf = devBuffer_static;
318
319 }
320 OS_MUTEX mutex_a; // send uart data mutex
321 static int ct_dev_open(void)
322 {
323     os_mutex_create(&mutex_a);
324     ct_uart_init();
325     return 0;
326 }
327
328 static int ct_dev_close(void)
329 {
330     if(_this->udev){
331         log_info("uart dev close\n");
332         os_mutex_delete(mutex_a);
333         uart_dev_close(_this->udev);
334     }
335     return 0;
336 }
337

// at_cmds.c
378 extern void at_send_event_cmd_complete(u8 opcode, u8 status, const u8 *packet, int size);
379
380 void predn_set_fix_pwr(u8 fix);
381 void ble_set_fix_pwr(u8 fix);
382
383 static u8 event_buffer[64 + 4];
384
385 extern OS_MUTEX mutex_a;
386
387 static void at_send_event_cmd_complete(u8 opcode, u8 status, const u8 *packet, int size);
388 {
389     if (size > 64) {
390         log_error("EVT payload overflow");
391         return;
392     }
393     os_mutex_pend(&mutex_a, 0);
394     event_buffer[0] = opcode;
395     event_buffer[1] = status;
396     memcpy(event_buffer + 2, packet, size);
397     r_printf("at_send_event_cmd complete\n");
398     at_send_event(AT_EVT_CMD_COMPLETE, event_buffer, size + 2);
399     os_mutex_post(&mutex_a);
400 }

// le_at_com.c
1329 {
1330     at_send_event_callback = cbk;
1331 }
1332
1333 static void ble_at_send_event(u8 event_type, const u8 *packet, int size)
1334 {
1335     if (at_send_event_callback) {
1336         at_send_event_callback(event_type, packet, size);
1337     }
1338 }
1339 extern OS_MUTEX mutex_a;
1340
1341 static void ble_at_receive_data(u16 handle, const u8 *packet, int size)
1342 {
1343     os_mutex_pend(&mutex_a, 0);
1344
1345     memcpy(at_tmp_buffer, &handle, 2);
1346     memcpy(&at_tmp_buffer[2], packet, size);
1347     ble_at_send_event(AT_EVT_BLE_DATA_RECEIVED, at_tmp_buffer, size + 2);
1348     os_mutex_post(&mutex_a);
1349 }
1350
1351 int ble_at_set_address(u8 *addr)
1352 {

```

34. 获取运行时堆的剩余可用 ram 的大小 20210218 xin

```

472 [00:02:41.144][Info]: [BOARD] [90640 90528 101760]
473 [00:02:42.143][Info]: [BOARD] X_timer_scan
474 [00:02:42.144][Info]: [BOARD] [90640 90528 101760] 注意区分堆、bss的区别
475 [00:02:43.143][Info]: [BOARD] X_timer_scan
476 [00:02:43.144][Info]: [BOARD] [90640 90528 101760]
477
478
479 // 仅将文本发送到当前选项卡
480
481 void X_timer_scan(void *priv)
482 {
483     log_info("%s", __func__);
484     log_info("[%d %d %d]", xPortGetFreeHeapSize(),
485             xPortGetMinimumEverFreeHeapSize(), xPortGetPhyiscalMemorySize());
486 }
487

```

此 3 个函数, 不同的 SDK 版本的含义不太一样, 以实际打印测试为准。

35. 开启硬件定时器中断注意点 20210305 xin

```

#define TCFG_LOWPOWER_LOWPOWER_SEL 0//SLEEP_EN //SNIFF 状态下芯片是否进入 powerdown
#define CONFIG_BT_NORMAL_HZ (96 * 1000000L)

void hard_timerDeal(void) AT VOLATILE RAM CODE
{
    //硬件定时器频率小于100uS, 需要将代码指定到ram
    //整个函数调用栈, 都要在ram里
}

////下面函数调用的使用函数都必须放在ram
interrupt
AT VOLATILE RAM CODE
static void timer2_isr()
{
    TIMER_CON |= BIT(14);
    if (timer_led_scan) {
        timer_led_scan(NULL);
    }
    hard_timerDeal();
}

```

36. 637X 芯片唤醒部分打印信息缺失 20210309 xin

建议使用 DP 口打印，打印信息会很完整。

如果使用其他 IO 口打印，关机唤醒后，开机开头会有一部分打印缺失。属于正常现象，是 SDK 做了一些限制操作，暂不支持修改，需要看这部分打印，换用 DP 口打印。

```
void board_power_init(void)
{
    log_info("Power init : %s", __FILE__);
    gpio_direction_output(IO_PORTC_03, 1); //touch chip
    power_init(&power_param);
    power_set_callback(TCFG_LOWPOWER_LOWPOWER_SEL, sleep);
    wl_audio_clk_on();
    power_keep_dacvdd_en(0);
    r_printf("%s", __func__);
    power_wakekeep_init(&awk_param);
    r_printf("%s", __func__);
}

#if (!TCFG_IKEY_ENABLE && !TCFG_AKEY_ENABLE)
    charge_check_and_set_pinr(1);
#endif
```

[00:00:07.155]app_key_evnet: 2.0
[00:00:07.303]key_value: 0x0, event: 2
[00:00:07.304]app_key_evnet: 2.0
[00:00:07.306] [Info]: [PMU]—4
[00:00:07.307] [Info]: [PMU]PWR_LDO15
[00:00:07.307] [Info]: [PMU]Enter Power off (Soft)
[00:00:07.309] [Info]: [PMU]P3_VLD_KEEP 0x4
[00:00:07.310] [Info]: [PMU]P1_M2P_INT_IE 0xff
[00:00:07.154]board_power_init 637唤醒后的打印，只有初始化唤醒
[00:00:07.155]wvdd_lev: 6 口之后，才能打印出来
[00:00:07.157]pvdd_lev: 13, pvd_lev_l: 10
[00:00:07.174]vbat_adc_value = 358
[00:00:07.181]vbg_adc_value = 275
[00:00:07.181]add sample ch 5000d
[00:00:07.182] [Info]: [APP_AUDIO]sys_cvol_max:16,call_cvol_max:15
[00:00:07.184] [Info]: [USER_CFG]bt name config:AC697N

```
#define TCFG_UART0_ENABLE          ENABLE_THIS_MOUDLE
#define TCFG_UART0_RX_PORT          NO_CONFIG_PORT
#define TCFG_UART0_TX_PORT          IO_PORT_DP
#define TCFG_UART0_BAUDRATE         1000000
```

37. 632X 的 PB2 引脚修改状态会 VCM 复位 20210414 xin

```
VCM_DET_EN(0);
gpio_set_direction(IO_PORTB_02, 0);
```

修改 PB2 引脚前，先调用 VCM_DET_EN(0);

注意：PB2 硬件上不能默认接地(如接灯、接三极管基极)，否则会开不了机。

38. 632X 获取复位源及唤醒源 20210603 xin

在 setup.c 添加此段代码：

```
u8 p11_reset_src = 0;
u8 p33_reset_src = 0;
u8 p33_wakeup_src0 = 0;
u8 p33_wakeup_src1 = 0;
extern u8 p33_reset_source_dump();
extern u8 p11_reset_source_dump();
void power_reset_source_info(void) //必须要用 DP 打印,唤醒后才能打印出来
{
    p33_wakeup_src0 = P33_CON_GET(P3_WKUP_PND0);
    p33_wakeup_src1 = P33_CON_GET(P3_WKUP_PND1);
    p33_reset_src = p33_reset_source_dump();
    p11_reset_src = p11_reset_source_dump();
    r_printf("%s -> START", __func__);
    printf("p11_reset_src:0x%02x", p11_reset_src);
    printf("p33_reset_src:0x%02x", p33_reset_src);
    printf("P3_WKUP_PND0:0x%02x", p33_wakeup_src0);
```

```
printf("P3_WKUP_PND1:0x%02x", p33_wakeup_src1);
if(p11_reset_src & BIT(1)){
    printf("P11 -> P33_SOFT_RESET");
}
if(p11_reset_src & BIT(2)){
    printf("P11 -> P33_SYS_RESET");
}
if(p11_reset_src & BIT(3)){
    printf("P11 -> WDT_RESET");
}
if(p11_reset_src & BIT(4)){
    printf("P11 -> P11_SOFT_RESET");
}
if(p33_reset_src & BIT(0)){
    printf("P33 -> VDDIO_POR");
}
if(p33_reset_src & BIT(1)){
    printf("P33 -> VDDIO_LVD");
}
if(p33_reset_src & BIT(2)){
    printf("P33 -> VCM");
}
if(p33_reset_src & BIT(3)){
    printf("P33 -> PPINR_8s_reset");
}
if(p33_reset_src & BIT(4)){
    printf("P33 -> P11_SYS_RESET");
}
if(p33_reset_src & BIT(5)){
    printf("P33 -> CPU_RESET");
}
if(p33_reset_src & BIT(7)){
    printf("P33 -> WAKEUP");
    extern const struct wakeup_param wk_param;
    for(u8 i; i < 8; i++){
        if(p33_wakeup_src0 & BIT(i)){
            u8 gpio = wk_param.port[i]->iomap;
            printf("IO_WAKEUP:[%d 0x%02x]:IO_PORT%c_%02d", i, gpio,
                   (gpio/IO_GROUP_NUM) + 'A', gpio%IO_GROUP_NUM);
        }
    }
    r_printf("%s -> END", __func__);
}
```

```

log_i("~~~~~\n");
log_i("      setup_arch %s %s", __DATE__, __TIME__);
log_i("~~~~~\n");

power_reset_source_info(); //只能放这个位置

/* clock_dump(); */

efuse_dump();

- 5.7k setup.c  C/l  AcP@K  edit  un

```

power_reset_source_info -> START
p11_reset_src:0x02
p33_reset_src:0x80
P3_WKUP_PNDO:0x08
P3_WKUP_PND1:0x00
P11 -> P33_SOFT_RESET 唤醒口
P33 -> WAKEUP
IO_WAKEUP:[3 17]:IO_PORTB_01
power_reset_source_info -> END

power_reset_source_info -> START
p11_reset_src:0x04
p33_reset_src:0x20
P3_WKUP_PNDO:0x0a
P3_WKUP_PND1:0x00
P11 -> P33_SYS_RESET 复位源
P33 -> CPU_RESET
power_reset_source_info -> END

39. 升级保留 VM 及 MAC 地址数据 20210603 xin

```

//是否支持升级之后保留vm数据
const int support_vm_data_keep = 1;

```

变量置 1

修改对应下载目录的 isd_config.ini 文件

```

74 #操作符说明 OPT:
75 # 0: 下载代码时擦除指定区域
76 # 1: 下载代码时不操作指定区域
77 # 2: 下载代码时给指定区域加上保护
78 ######
79 [RESERVED_CONFIG]
80 BTIF_ADR=AUTO;
81 BTIF_LEN=0x1000;
82 BTIF_OPT=1;          需要固定VM起始地址到flash的末端
83
84 #flash_size - VM_size - BTIF_size - keep_size = memory_start_addr(4K unit)
85 #256K - 8K - 4K - 60K = 184K (支持APP-OTA升级)
86 #256K - 8K - 4K - 40K = 204K (支持测试盒升级)
87 VM_ADR=204K;        起始地址按照上面公式计算
88 VM_LEN=8K;           VM区最小为8K, 不可再小
89 VM_OPT=1;
90
91 [BURNER_CONFIG]     不同芯片型号内置flash大小不一样,
92 SIZE=32;             不同应用开发需要的保留区大小也不一样, 编译信息会有提示
93

```

不同芯片型号内置flash大小不一样,
不同应用开发需要的保留区大小也不一样, 编译信息会有提示

- 2.4k isd_config.ini<standard> Conf[Unix] general unix | 93: 0

```
SPI nor flash online.
Online flash id: eb6012
Online flash size: 256K ← 芯片flash大小
Erase Falsh Size is 256
ota.bin: E:/630X/1Test/AC630N_bt_data_transfer_sdk_release_v0.9.2/c-bd19-001/
ISDdownload
----- OTA UPDATE INFO -----
| VM大小 = 0xbff00
| PASS: 测试盒BLE升级 (大小=0xa6f4) 需要最小空间为 0xb000
| PASS: 测试盒经典蓝牙升级 (大小=0x8868) 需要最小空间为 0x9000
| !!!!!!! FAIL: BLE RCSP升级 (大小=0xf12c) 需要最小空间为 0x10000
~~~~~
| 此VM空间支持升级方式有:
| * 测试盒BLE升级 ← 此时支持的升级方式
| * 测试盒经典蓝牙升级
```

修改 isd_config.ini 文件后，注意看编译后的提示信息。

40. 不需要按键功能，需要自行初始化唤醒口 xin 20210622

```
static void board_devices_init(void)
{
#if TCFG_PWMLED_ENABLE
    pwm_led_init(&pwm_led_data);
#endif

#if (TCFG_IODEV_ENABLE || TCFG_ADKEY_ENABLE || TCFG_TOUCH_KEY_ENABLE)
    key_driver_init(); 没有开启按键功能,需要自行初始化唤醒口
#else
    extern const struct wakeup_param wk_param;
    gpio_set_direction(wk_param.port[1]->iomap, 1);
    gpio_set_dle(wk_param.port[1]->iomap, 1);
    gpio_set_dleh(wk_param.port[1]->iomap, 1);
    gpio_set_hd(wk_param.port[1]->iomap, 0);
    gpio_set_hd0(wk_param.port[1]->iomap, 0);
    if(wk_param.port[1]->pullup_down_enable){
        if(wk_param.port[1]->edge == FALLING_EDGE){
            gpio_set_pull_up(wk_param.port[1]->iomap, 1);
            gpio_set_pull_down(wk_param.port[1]->iomap, 0);
        }else{
            gpio_set_pull_up(wk_param.port[1]->iomap, 0);
            gpio_set_pull_down(wk_param.port[1]->iomap, 1);
        }
    }
#endif

#if (!TCFG_CHARGE_ENABLE)
    CHARGE_EN(0);
#endif

- 20k board_ac632n_demo.c C/1 AC@R
struct port_wakeup port0 = {
    .pullup_down_enable = ENABLE,
    .edge               = FALLING_EDGE,
    .both_edge          = 0,
    .iomap              = TCFG_IODEV_POWER_ONE_PORT, 唤醒口 //配置I/O
    .filter             = PORT_FLT_2ms,
};

- 20k board_ac632n_demo.c C/1
const struct wakeup_param wk_param = {

#if 1//TCFG_ADKEY_ENABLE || TCFG_IODEV_ENABLE
    .port[1]      = &port0,
#endif
/* .sub         = &sub_wkup, */
/* .charge     = &charge_wkup, */
}
```

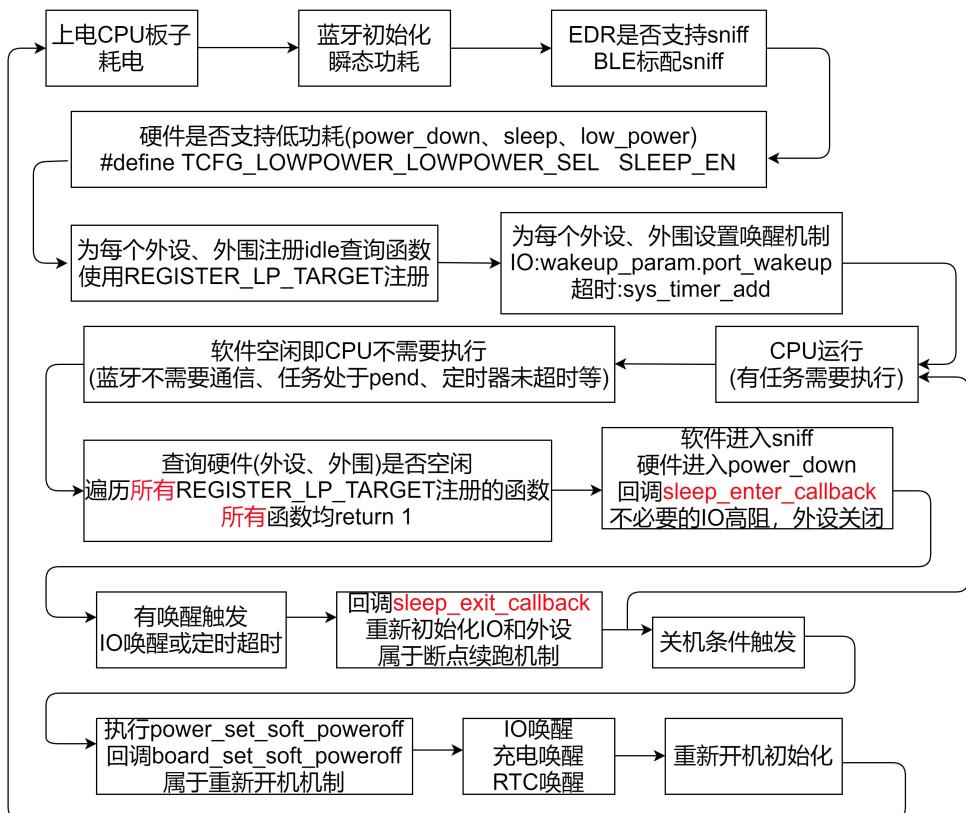
```
/*进软关机之前默认将IO口都设置成高阻状态，需要保留原来状态的请修改该函数*/
static void close_gpio(void)
{
    u16 port_group[] = {...};
    if(P3_ANA_CON2 & BIT(3))
    {...}
    if(P3_PINR_CON & BIT(0))
    {...}
    extern const struct wakeup_param wk_param;
    port_protect(port_group, wk_param.port[1]->iomap);
}

* 20k board_ac632n_demo.c C/1 AC@XK
void key_active_set(u8 port)
{
#define TCFG_IOKEY_ENABLE || TCFG_ADKEY_ENABLE || TCFG_TOUCH_KEY_ENABLE
    #if TCFG_IOKEY_ENABLE
        is_key_active = 35; //35*10Ms
    #endif
}
```

进入power_down需要屏蔽唤醒口

添加宏包含,不设置此变量

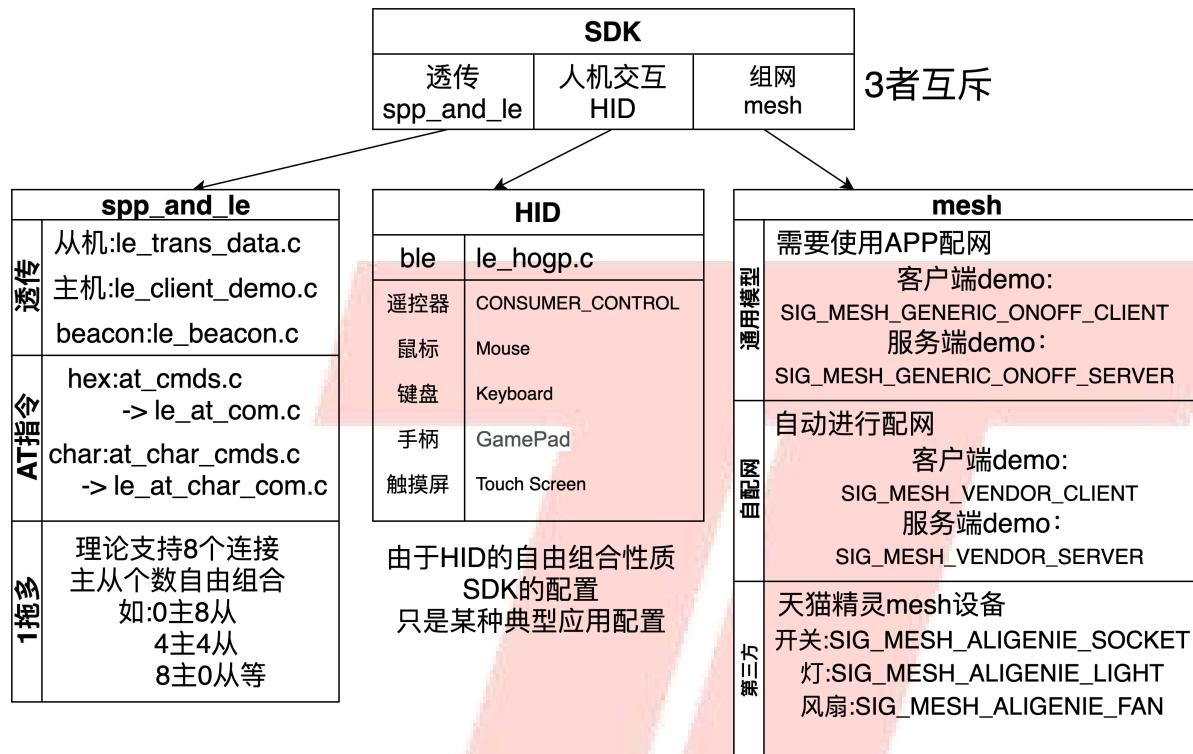
41. 低功耗流程图 xin 20210622



42. 获取系统开机时间 xin 20210622

```
extern u32 timer_get_ms(void); //10ms 步进
extern unsigned long jiffies_msec(); //1ms 步进
```

43. SDK 框架图 xin 20210622



44. 63x 各系列的触摸按键资源 hgy 20210726

芯片型号	ctmu	p1cnt	1p key
632x	X	不限路数和GPIO	X
635x	16	不限路数和GPIO	X
636x	X	不限路数和GPIO	X
637x	8	不限路数和GPIO	2(PB_01、PB_02)
638x	8	不限路数和GPIO	2(PB_01、PB_02)

注:ctmu 为不带唤醒功能的硬件计数触摸, 不可映射(具体管脚需看代码及用户手册)

p1cnt为不带唤醒功能的软件计数触摸, 可以任意GPIO(GPIO为普通IO, PA_00-PA_15、PB_00-PB_15、PC_00-PC_15、PD_00-PD_07), 任意路数

1p key为带唤醒功能的硬件计数触摸, 不可映射

设计优先使用ctmu, 后使用p1cnt

X: 代表没有该资源

各系列ctmu硬件对应管脚

635x	IO_PORTA_00, IO_PORTA_01, IO_PORTA_02, IO_PORTA_03, IO_PORTA_04, IO_PORTA_05, IO_PORTA_06, IO_PORTA_07, IO_PORTA_09, IO_PORTA_10, IO_PORTC_00, IO_PORTC_01, IO_PORTC_02, IO_PORTC_03, IO_PORTC_04, IO_PORTC_05
637x	IO_PORTA_03, IO_PORTA_04, IO_PORTA_05, IO_PORTA_06, IO_PORTA_07, IO_PORTA_08, IO_PORTB_05, IO_PORTB_06,
638x	IO_PORTA_03, IO_PORTB_07, IO_PORTA_05, IO_PORTA_06, IO_PORTC_02, IO_PORTC_03, IO_PORTB_04, IO_PORTB_06,

软关机发生 VCM (PB2) 复位时

```
[00:00:00.100][2-P33]--Reset Source : 0x4
[00:00:00.101][VCM]----->
[00:00:00.101][1-P11]--Reset Source : 0x4
[00:00:00.102]P33 SYS RESET
[00:00:00.103]P33 POWER RETURN
[00:00:00.103]p11_reset_src:0x04
```

方法一：按照第 37 点修改

方法二：在 close_gpio 函数添加下面的代码

```
if(P3_ANA_CON2 & BIT(3))
{
    port_protect(port_group, IO_PORTB_02); //protect VCM_IO
}
```

```
void usb1_iomode(u32 enable);
/*进软关机之前默认将IO口都设置成高阻状态，需要保留原来状态
的请修改该函数*/
static void close_gpio(void)
{
    u16 port_group[] = {
        [PORTA_GROUP] = 0xff,
        [PORTB_GROUP] = 0x3ff,// 
        [PORTC_GROUP] = 0x3ff,// 
    };

    if(P3_ANA_CON2 & BIT(3))
    {
        port_protect(port_group, IO_PORTB_02); //protect VCM_IO
    }

    if(D2_DINR_CON & BIT(0))
```

46. ac636n_chargebox_sdk_release_v1.3.0 一拖二、一拖八烧录需要修改.ini 文件 FSW
20210805

修改 ac636n_chargebox_sdk_release_v1.3.0\SDK\cpu\br25\tools\sandbox\standard\isd_config_lrc.ini 为下图

```
ac636n_chargebox_sdk_release_v1.3.0 > SDK > cpu > br25 > tools > soundbox > standard > isd_config_lrc.ini
1 #####
2 #
3 # 配置数据按照 长度+配置名字+数据的方式存储
4 #
5 #####
6
7 [EXTRA_CFG_PARAM]
8 NEW_FLASH_FS=YES;
9 #FORCE VM ALIGN=YES;
10 CHIP_NAME=AC636N;//8
11 ENTRY=0x1E00120;//程序入口地址
12 PID=AC696x_TWS;//长度16byte,示例:芯片封装_应用方向_方案名称
13 VID=0.01;
```

47. AC632X 芯片的 PWM 资源使用 20210823 xin

3 个 timer_pwm(timer0/2/3)

4 个 mcpwm(ch0/1/2/3)

全映射，任意 IO

//Timer_PWM 使用方式

```
timer_pwm_init(JL_TIMER0, IO_PORTA_00, 10000, 1000);
```

//timer 1 不能使用,是系统定时器

```
timer_pwm_init(JL_TIMER2, IO_PORTA_01, 10000, 2000);
```

```
timer_pwm_init(JL_TIMER3, IO_PORTA_02, 10000, 5000);
```

///电机 mcpwm 使用方式

```
pwm_p_data.pwm_aligned_mode = pwm_edge_aligned; //边沿对齐
pwm_p_data.pwm_ch_num = pwm_ch0; //pwm_ch1、pwm_ch2、pwm_ch3 //通道号 4 个选一个
pwm_p_data.frequency = 1000; //频率
pwm_p_data.duty = 5000; //占空比
pwm_p_data.h_pin = IO_PORTA_06; //任意引脚
pwm_p_data.l_pin = -1; //任意引脚,不需要就填-1
pwm_p_data.complementary_en = 0; //h_pin 和 l_pin 波形 0:同步,1:互补
mcpwm_init(&pwm_p_data);
///需要多个则重复上段代码初始化, 选不同通道号
```

48. 内部 flash 储存数据 20210824 xin

数据存储在内部 flash，有 2 种方式。

第一种内部 VM 存储，双备份存储，擦除均衡机制，V100 版本及以前版本，限制总大小为 4K，如果单个数据大于 512B，或者所有数据大于 4KB，则建议用第二种。

第二种直接操作 flash 区域，参考《flash 操作地址》补丁。

//VM 方式使用例子

```

79 [RESERVED_CONFIG]
80 BTIF_ADR=AUTO;
81 BTIF_LEN=0x1000;
82 BTIF_OPT=1;
83
84 VM_ADR=0;
85 VM_LEN=24K; 可以修改大小
86 VM_OPT=1;
87

```

```

//-----  

// 配置项ID分配说明  

// 1.配置项ID号根据存储区域进行分配;  

// 2.存储区域有3个： 1)VM区域； 2)sys_cfg.bin； 3)BTIF区域  

// 3.配置项ID号分配如下：  

// 0)[0]: 配置项ID号0为配置项工具保留ID号;  

// 1)[ 1 ~ 49]: 共49项，预留给用户自定义，只存于VM区域；  

// 2)[ 50 ~ 99]: 共50项，sdk相关配置项，只存于VM区域；  

// 3)[100 ~ 127]: 共28项，sdk相关配置项，可以存于VM区域，sys_c  

// 4)[512 ~ 700]: 共188项，sdk相关配置项，只存于sys_cfg.bin;  

//-----  

//-----  

// 用户自定义配置项[1 ~ 49]  

//-----  

#define CFG_USER_DEFINE_BEGIN 1  

#define CFG_VM_TEST 2 ←  

#define CFG_VM_VAR 3 ←  

#define CFG_USER_DEFINE_END 49  

//-----  


```

```

//VM使用规范
// syscfg_id.h 定义一个ID号宏

static u8 data[512] __attribute__((aligned(4)));
static u32 var;
void vm_op_demo(void)

{
    log_info("%s", __func__);
    int ret = 0;
    memset(data, 0x00, sizeof(data)); //先清0数组buf
    var = 0;

    //数组形式
    ret = syscfg_read(CFG_VM_TEST, data, sizeof(data));
    log_info("%s[read:%d]", __func__, ret);
    if(ret != sizeof(data)){
        log_info("%s[Error line:%d]", __func__, LINE__);
        //负值,或大小不符,都可以表示异常,进行异常处理
    }
    put_buf(data, 16);
    data[1]++;
    //赋值要储存的数据
    ret = syscfg_write(CFG_VM_TEST, data, sizeof(data));
    log_info("%s[write:%d]", __func__, ret);
    if(ret != sizeof(data)){
        log_info("%s[Error line:%d]", __func__, LINE__);
        //负值,或大小不符,都可以表示异常,进行异常处理
    }

    //变量形式
    ret = syscfg_read(CFG_VM_VAR, &var, sizeof(u32)); //变量要取值,传变量的数据类型
    log_info("%s[read:%d]", __func__, ret);
    if(ret != sizeof(u32)){
        log_info("%s[Error line:%d]", __func__, LINE__);
        //负值,或大小不符,都可以表示异常,进行异常处理
    }
    log_info("%s[var:%d]", __func__, var);
    var++;
    //赋值要储存的数据
    ret = syscfg_write(CFG_VM_VAR, &var, sizeof(u32));
    log_info("%s[write:%d]", __func__, ret);
    if(ret != sizeof(u32)){
        log_info("%s[Error line:%d]", __func__, LINE__);
        //负值,或大小不符,都可以表示异常,进行异常处理
    }
}

```

49. 浮点打印 20210824 xin

硬件支持浮点 float 的 IC 有 AC635X、AC636X、AC637X、AC638X

硬件不支持浮点、软件实现浮点的 IC: AC631X、AC632X

SDK 需要添加 put_float.c, 文件可以放到 apps\common\debug\目录下面, 记得添加到工程文件

```
void put_float(double fv)
{
    if (__builtin_isnan(fv)) {
        puts("nan");
    } else if (__builtin_isinf(fv)) {
        puts("inf");
    } else {
        /* flt((void *)0, fv, 10, 3, 'f', ' ' | SIGN); */
        /* flt((void *)0, fv, 10, 7, 'f', ' ' | SIGN); */
        flt((void *)0, fv, 10, 6, 'f', ' ' | SIGN); //小数点只能到6位
    }
    puts("\n");
}
```

建议只用 float, 立即数之后需要加 f, 至于 double 所有 IC 都是软件实现, 运算速度会比较慢。

```
void float_test(void)
{
    //小数点只能到6位
    r_printf("%s\n", __func__);
#define PI_1 3.141592654f //只能保证小数点后6位精准
#define PI_2 3.141592714f
#define PI_3 3.141592394f
    float pi1 = PI_1;
    float pi2 = PI_2;
    float pi3 = PI_3;
    double db = PI_1;
    float ft = 0.5f; //立即数后必须加f
    float ft1 = 4.09f;
    float ft2 = ft * ft1;
    put_float(pi1);
    put_float(pi2);
    put_float(pi3);
    put_float(db);
    put_float(ft);
    put_float(ft1);
    put_float(ft2);
}
```

put_float.c 可以到网盘下载。

获取地址: https://pan.baidu.com/s/1XbI8sD0_-7EXJZkn0b7Bmg 提取码: hjfw

50. 获取芯片 UID 20210824 xin

//芯片没有 UID, 用内置 flash 的 UUID 使用

```
static u8 chip_uid[16] = {0};
u8 *get_chip_uid(void)//芯片没有 UID,用内置 flash 的 ID 使用
{
    log_info("%s", __func__);
    extern __attribute__((weak)) u8 *get_norflash_uuid(void);
    memcpy(chip_uid, get_norflash_uuid(), sizeof(chip_uid));
    /* put_buf(chip_uid, sizeof(chip_uid)); */
    return chip_uid;
}
```

下图中的“ROM”中就是芯片 ROM 启动中会调用到，然后释放的 IO 口。

下图中有些有括号 () 有上拉、下拉的是默认的 IO 电阻上下拉状态。

LAT 释放时间			
	PA0	耐高压	ROM后
1	PA1		ROM后
2	PA2		ROM后
3	PA3		ROM后
4	PA4	耐高压	ROM后
5	PA5		ROM后
6	PA6		ROM后
7	PA7		ROM中
8	PA8		ROM后
9	PB0	耐高压	ROM后
10	PB1 (上拉)	默认长按Reset	ROM后
11	PB2		ROM后
12	PB3		ROM后
13	PB4		ROM后
14	PB5		ROM后
15	PB6		ROM后
16	PB7		ROM后
17	PB8	耐高压	ROM后
18	PC0	耐高压	ROM后
19	PC1		ROM后
20	PC2		ROM后
21	PC3		ROM后
22	PC4		ROM后
23	PC5		ROM后
24	PDO (上拉)		ROM中
25	PD1		ROM中
26	PD2		ROM中
27	PD3 (上拉)		ROM中
28	PD4(输出0)	Flash_A供电脚	ROM中
29	PD5(输出0)	Flash_B供电脚	ROM中
30	PD6		ROM中
31	PD7		ROM中
32	USBDP(下拉)		ROM中
33	UDBDM(下拉)		ROM中
34	LD05V	耐高压	ROM中
35	P00	耐高压	ROM后

注：ADC13~15接模拟模块，用于测试