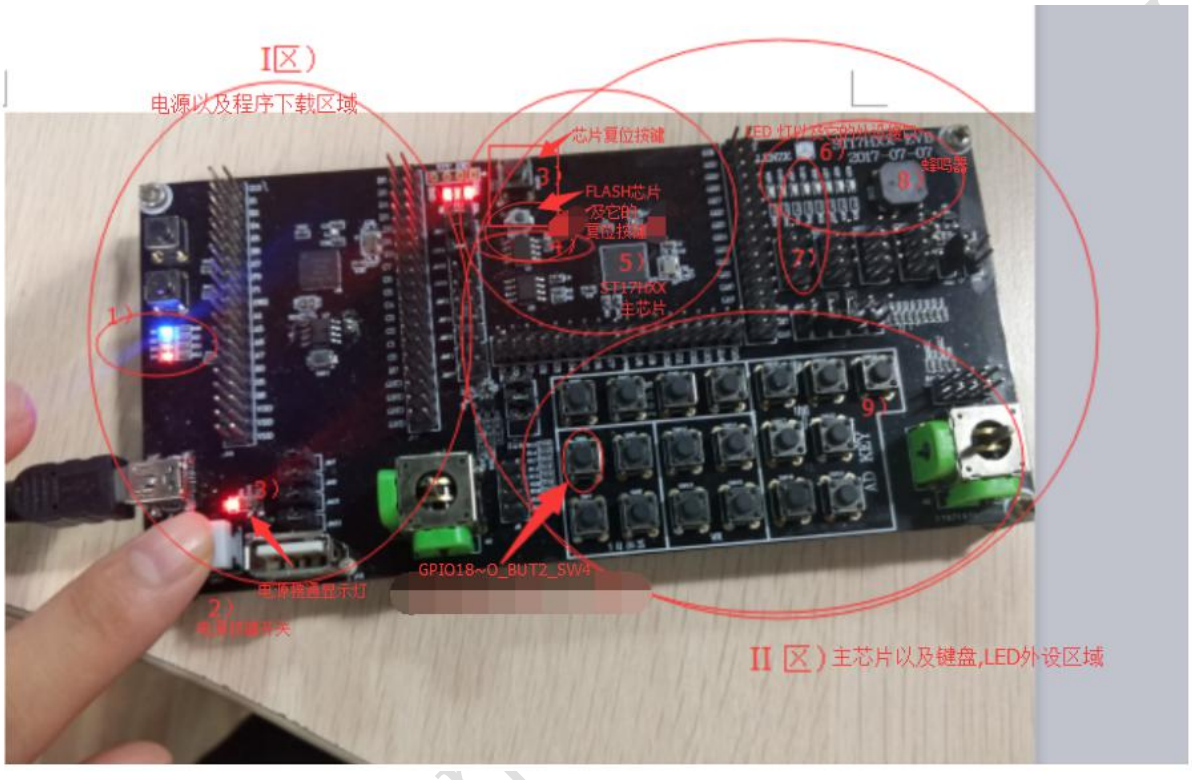
ST17HXX-EVB 开发者指南

一）认识开发板的分布图



上图所示的 I)区，是供电和完成程序下载功能的区域。

标识 1）处是显示此功能区正常状态的 led 灯区（其中刚通过 USB 线连上电脑后的 LED 灯会亮 2s 左右，随后在电脑端的驱动程序启动完成时全灭。如果常亮则说明下载区芯片工作异常，处理方法见后面第 5 部分--问题解决一栏）。

而 II)区则是用来完成开发功能的区域，包括图中带绿色安装件的左右 2 个摇杆，

标识 9）全部 19 个的按键，

标识 6）的 LED 上下拉可控灯区，

标识 5）是主控制芯片 ，

标识 3）芯片硬件复位按键，以及其下方的用来控制板上的的 flash 复位的小按键，

标识 4）板上的的 flash 芯片，

标识 7）这样的 GPIO 引脚外接插针，在满足基本开发目标的基础上，便于人员用示波器或逻辑分析仪观测引脚动态。

二）介绍 ST17HXX-EVB 内部电路原理图，及使用方法

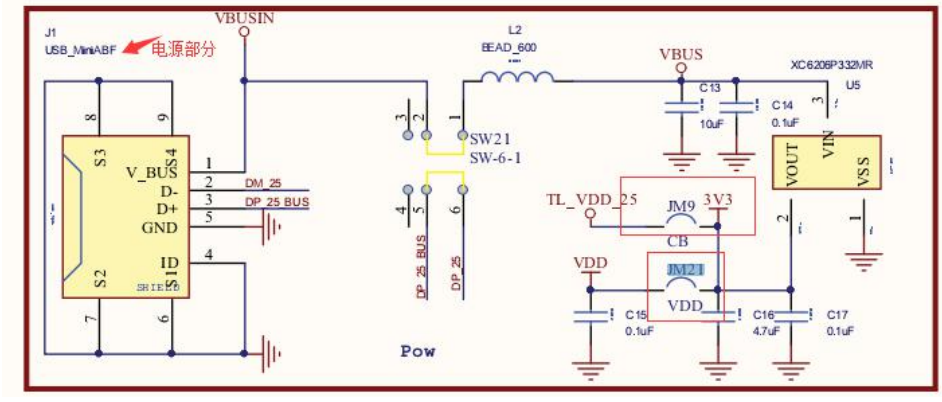


图 2 EVB 的供电电源区域

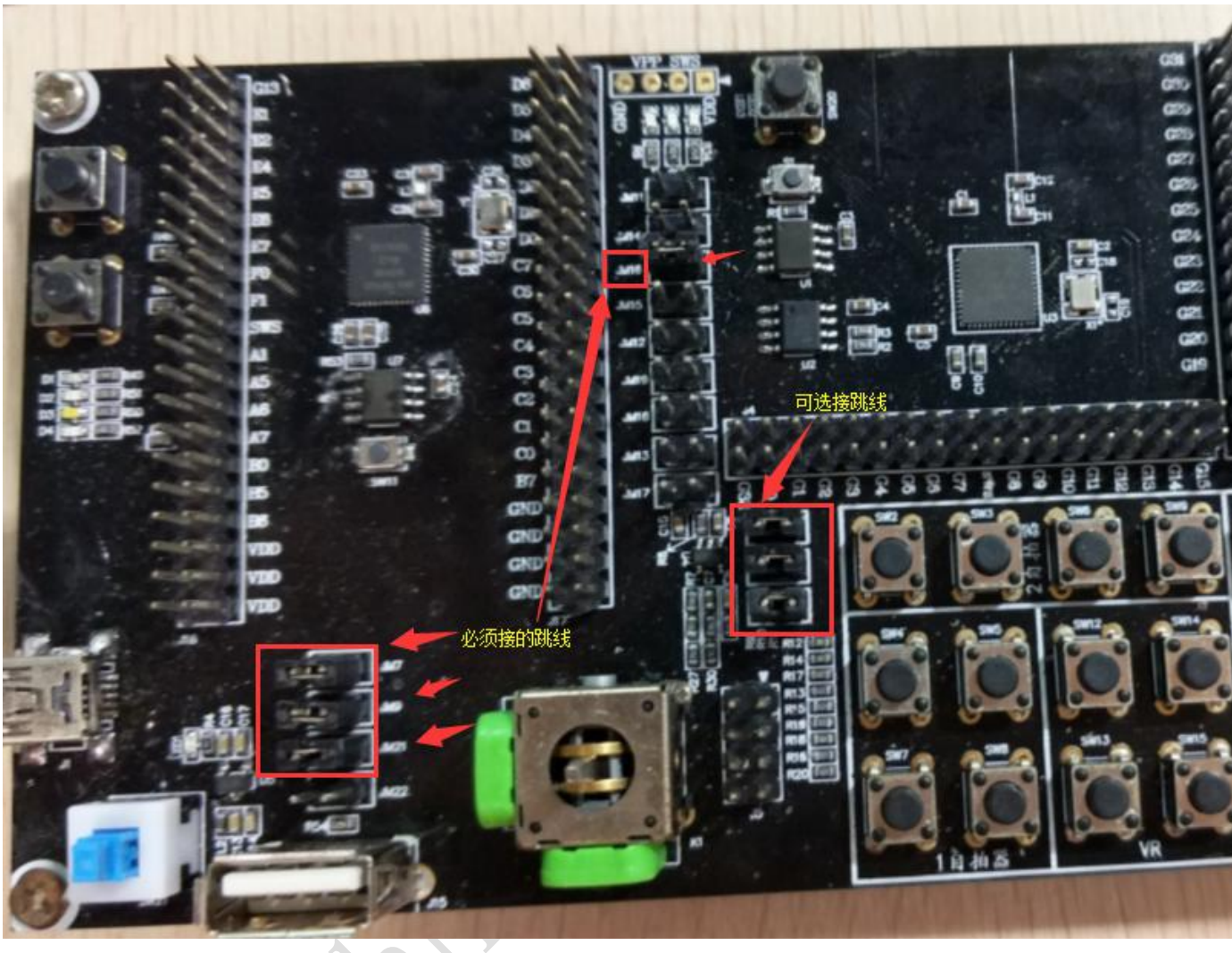
图 3

图 3-1

从上面可以看到我们要让电路有正常的供电，需要将 JM7,JM9,JM21, 几个跳线接通。

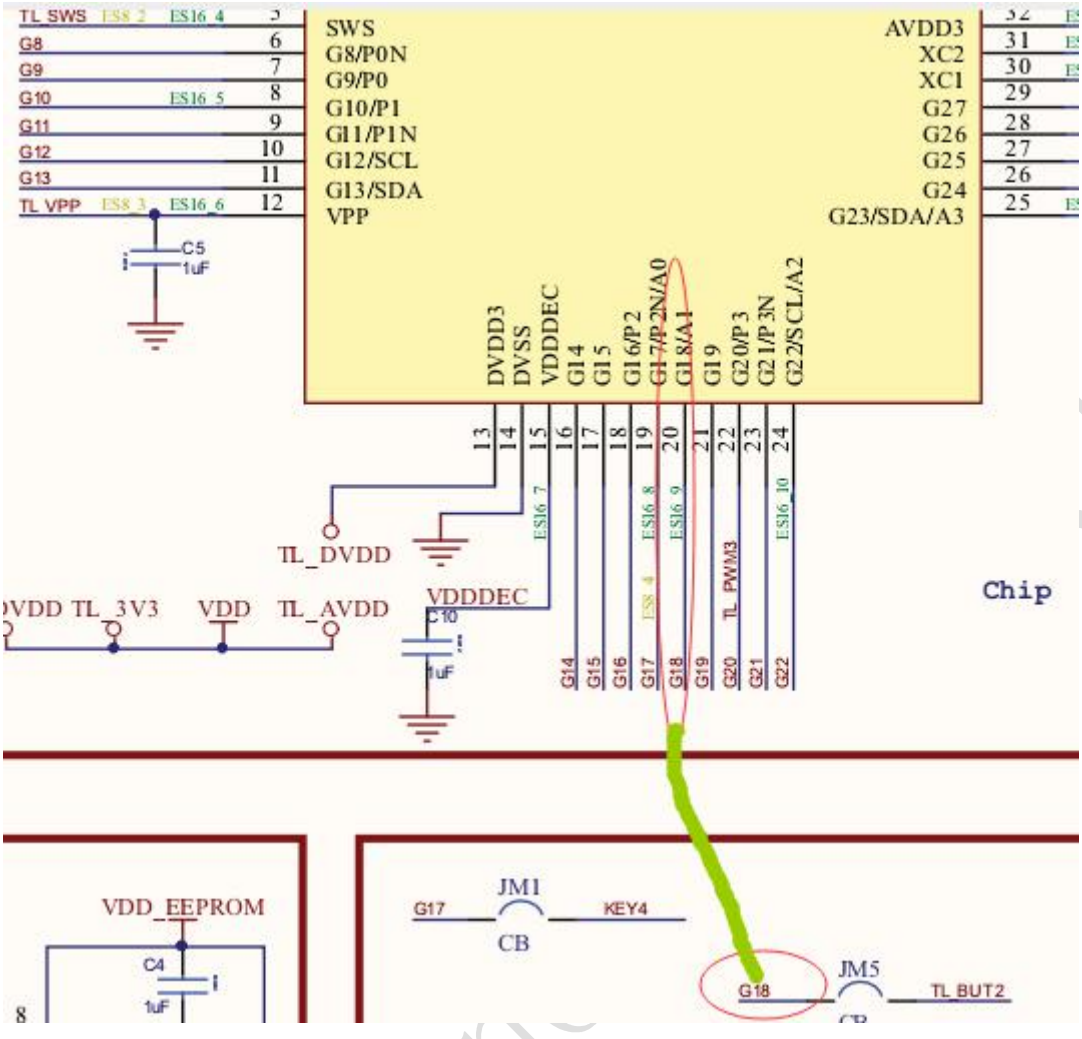
除此之外，如果想用 GPIO7 来控制 TL_LED2 的亮灭状态，我们还需要接通 JM14 ;想要使用外部 EEPROM 我们就要接通 JM16 给 EEPROM 供电.(如上图示).

跳线帽的作用是将两个分离的节点互相连接起来，拿到开发板后，按照下图所标示的方式将跳线帽接到电路中，即可开始程序的下载。

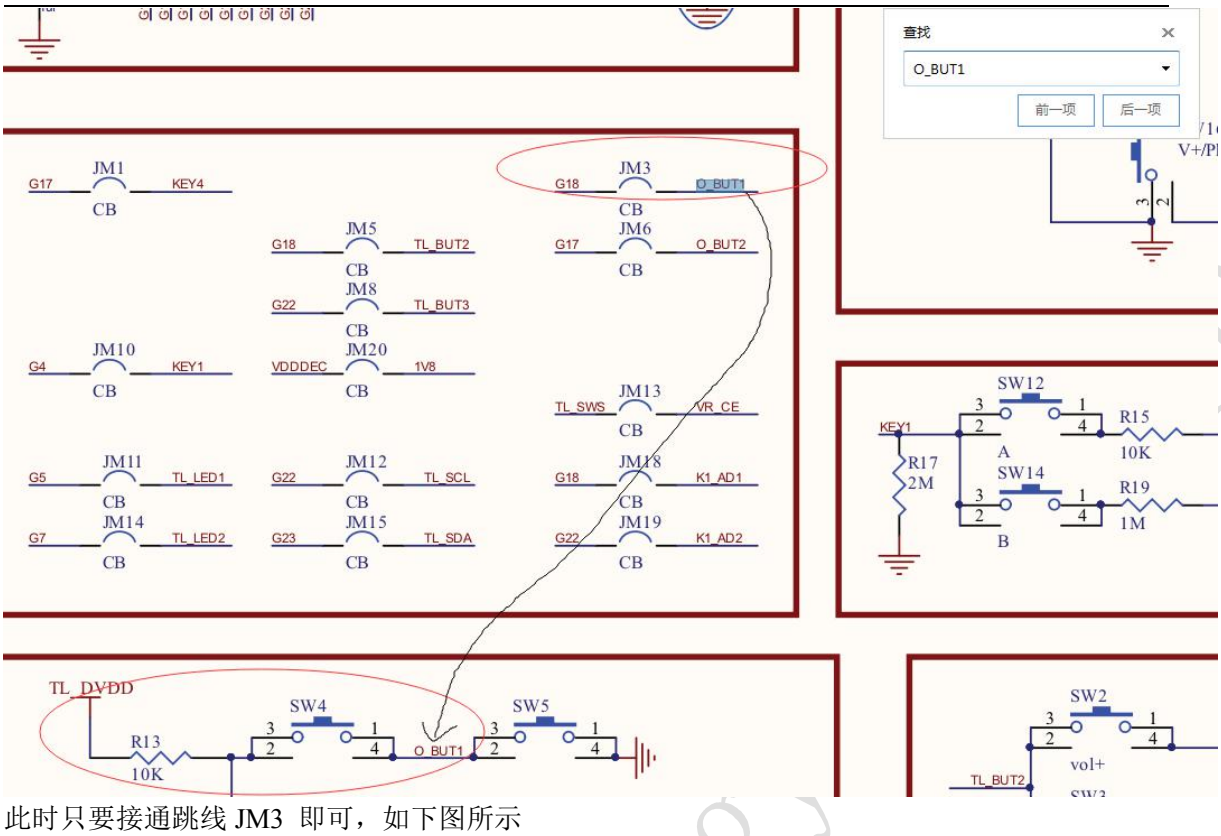


对于开发中的实际需求，可以以此类推，根据电路原理图来有选择地连接跳线，从而接通按键和具体 I0 口。

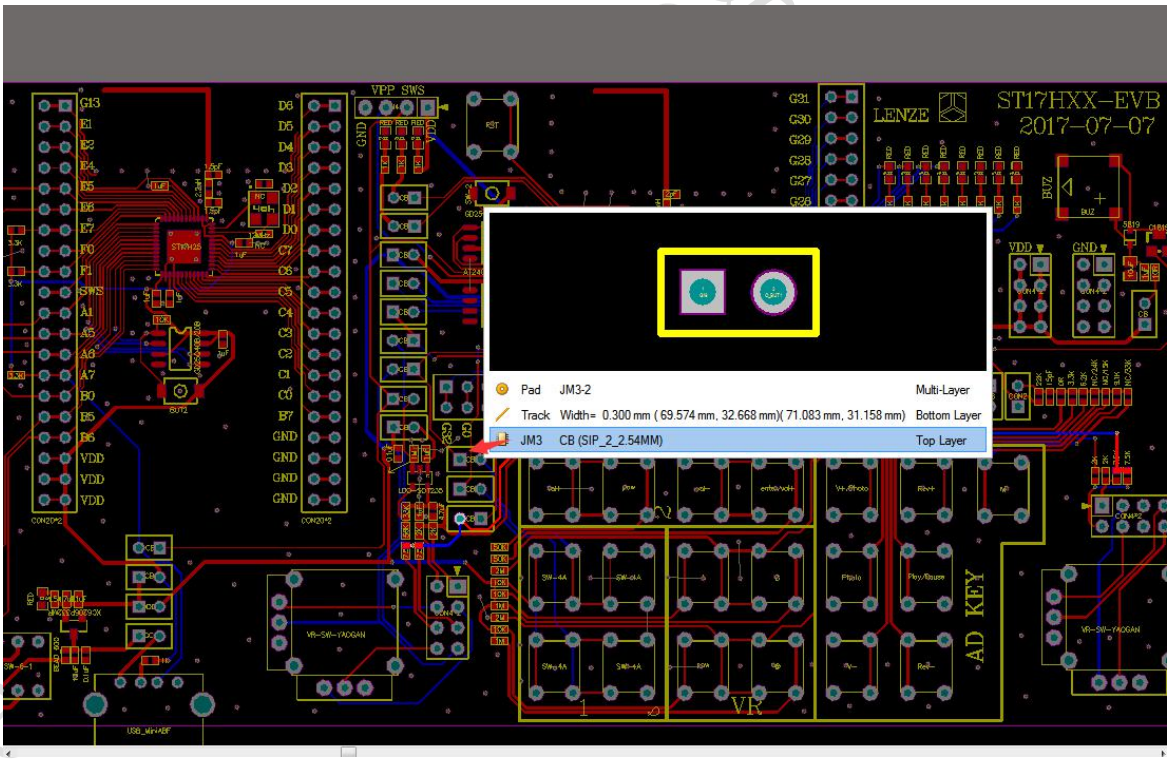
如果想用 GPIO18 来读取按键 TL_BUT2 的状态，我们要接通 JM5.(如下图所示)



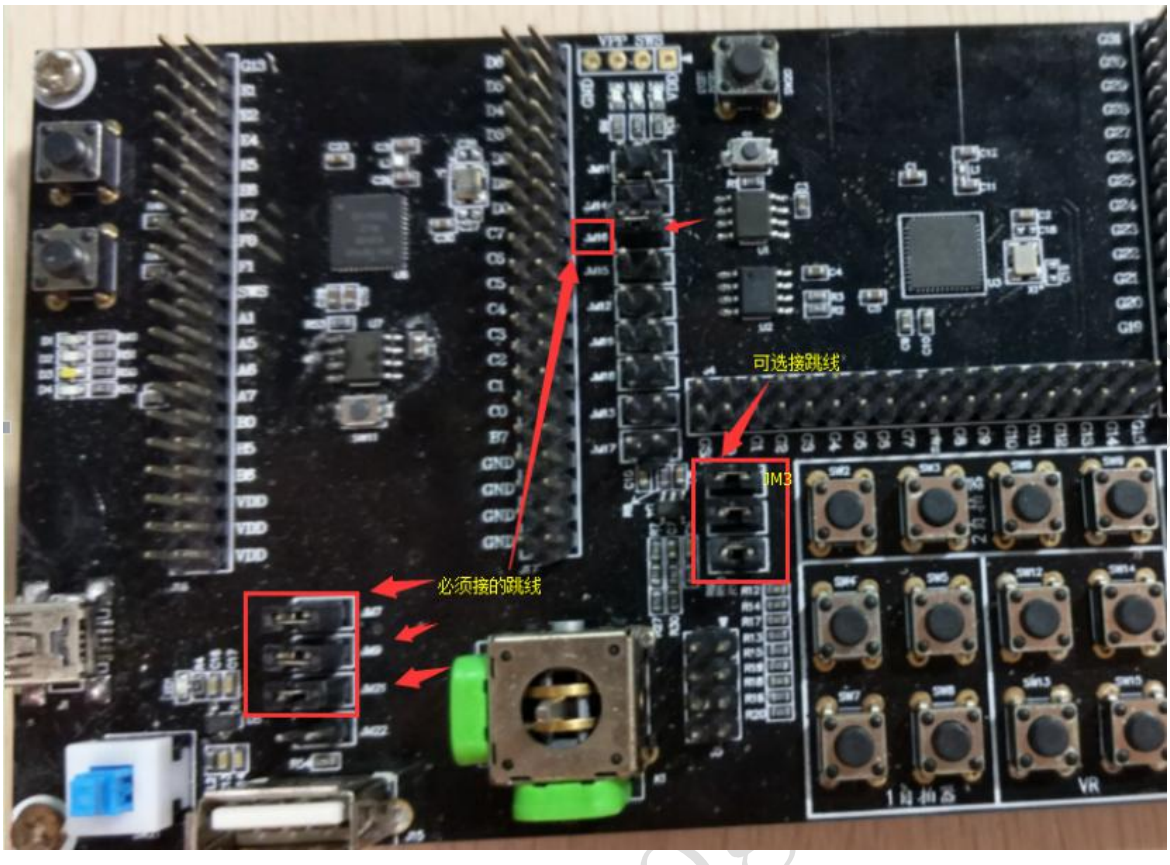
如果要用 GPIO18 来读取 O_BUT1 的状态，我们要接通 JM3.(如 下图所示)



此时只要接通跳线 JM3 即可，如下图所示



放大后从 PCB 板上对照到下图的“可选接跳线”框中的第一行，即为 JM3.



总之我们需要根据具体的需求，查找电路图并对照 PCB 原理图，通过跳线将相关的电路 GPIO 口与对应的外设所对应的网格标号引脚相连接，从而实现我们需要的控制效果。

三）程序编辑

3-1)、普通 IO 口设定的实现方法

本例中我们用 GPIO18 来读取 O_BUT1 的状态，首先接通 JM3.(如上图)，然后在 user_init() 程序中设定 GPIO18 为普通 GPIO 的输入口(在程序中画圈部分前 3 行),最后配置 GPIO18 在芯片内部的下拉电阻阻值为 100k 欧姆（在程序中画圈部分（第 4 行)。配置原理参照图 10 数据手册中对应部分。

```
gpio_set_func(GPIO_GP7, AS_GPIO);
gpio_set_output_en(GPIO_GP7,1);
gpio_set_input_en(GPIO_GP7,0);
```

17HXX 的 Datasheet 查阅， 默认初始化 GP7 口没有下拉电阻，

afe3V_reg09<5:4>	<1:0>	00	01 -- 1MOhm pull-up resistor 10 – 10kOhm pull-up resistor 11 – 100kOhm pull-down resistor
afe3V_reg09<7:6>	pullupdown_ctrl <1:0>	00	Wake up mux input GP24 pull up/down controls 00 -- No pull up/down resistor 01 -- 1MOhm pull-up resistor 10 – 10kOhm pull-up resistor 11 – 100kOhm pull-down resistor
afe3V_reg40<7:0>	pulldown_ctrl <7:0>	00000000	GP6 ~GP0, GP32 pull down enable 0--No pull down resistor 1--enable 100kOhm pull down resistor
afe3V_reg41<7:0>	pulldown_ctrl <15:8>	00000000	GP14 ~GP7 pull down enable 0--No pull down resistor 1--enable 100kOhm pull down resistor
afe3V_reg42<1:0>	pulldown_ctrl <17:16>	00	GP16~GP15 pull down enable 0--No pull down resistor

DS-ST17HXX-E11 78 Ver2.0.0

8.1.1.2 Drive strength

The registers in the "**DS**" column are used to configure corresponding pin's driving strength: "1" indicates maximum drive level, while "0" indicates minimal drive level. The "DS" configuration will take effect when the pin is used as output. It's set as the strongest driving level by default. In actual applications, driving strength can be decreased to lower level if necessary.

结合上下两张图，共同描述了 GPIO 口的驱动电流可以根据实际需要来配置。

Table 8- 7 IO drive strength for ST17H26/29ES16

No.	Pin Name	Drive Strength	
		"DS"=0	"DS"=1
1	GP4/scl/pwm2 #	0.7mA	4mA
2	GP5/sda/pwm2_inv/ pwm3 #	0.7mA	4mA
3	GP7/pwm0 #	0.7mA	4mA
4	SWS	4mA	8mA
5	GP10/pwm1 #	0.7mA	4mA
8	GP17/pwm2_inv/ANA0 *	0.7mA	4mA
9	GP18/ANA1 *	0.7mA	4mA
10	GP22/scl/ANA2 *	0.7mA	4mA
11	GP23/sda/ANA3 *	0.7mA	4mA

具体的配置寄存器在下图标识处，其中 580[7]代表 寄存器 580 的第 7 位。

Table 8- 3 GPIO lookup table 1 for the ST17H26/29ES16

Pin Name	Default Function	Priority0	Priority1	Priority2	Act as GPIO	Act as GPIO			Input Enable	DS (Drive Strength)
						OEN	Input	Output		
GP4/ scl/ pwm2	GPIO input	5d4[0] =1 scl	pwm2		586[4]	582[4]	580[4]	583[4]	581[4]	585[4]
GP5/ sda/ pwm2_inv/ pwm3	GPIO input	5d4[0] =1 sda	5d4[1]=1 pwm3	pwm2_inv	586[5]	582[5]	580[5]	583[5]	581[5]	585[5]
GP7/pwm0	GPIO input	pwm0			586[7]	582[7]	580[7]	583[7]	581[7]	585[7]

3.2 蓝牙省电协议相关的程序

```
static inline void public_loop()
{
    tick_app_wakeup = buzzer_led_ui();

    blt_brx_sleep (tick_app_wakeup);
    if(blt_state!=BLT_LINK_STATE_ADV) {
       blt_brx ();
    }
    else {
        if(is_buzzer_working){
            return;
        }
        static u8 adv_channel_sel = 0;
        adv_channel_sel++;
        blt_send_adv (BLT_ENABLE_ADV_ALL);
//      if(adv_channel_sel%3 == 0){
//          blt_send_adv (BLT_ENABLE_ADV_37|BLT_ENABLE_ADV_38);
//      }
//      else if(adv_channel_sel%3 == 1){
//          blt_send_adv (BLT_ENABLE_ADV_37|BLT_ENABLE_ADV_39);
//      }
//      else if(adv_channel_sel%3 == 2){
//          blt_send_adv (BLT_ENABLE_ADV_38|BLT_ENABLE_ADV_39);
//      }
    }
}

//u32 AAA_battery;
```

这段程序基于下图的理论，简单地将蓝牙工作状态分为睡眠模式，连接模式和广播模式。

3 BLE 模块

3.1 BLE 状态机

目前的 BLE SDK 有两个最基本的状态：广播状态（advertising state）和连接状态（connection state），当加入了低功耗（power management，简称 PM）管理后，增加一个 deepsleep 状态。

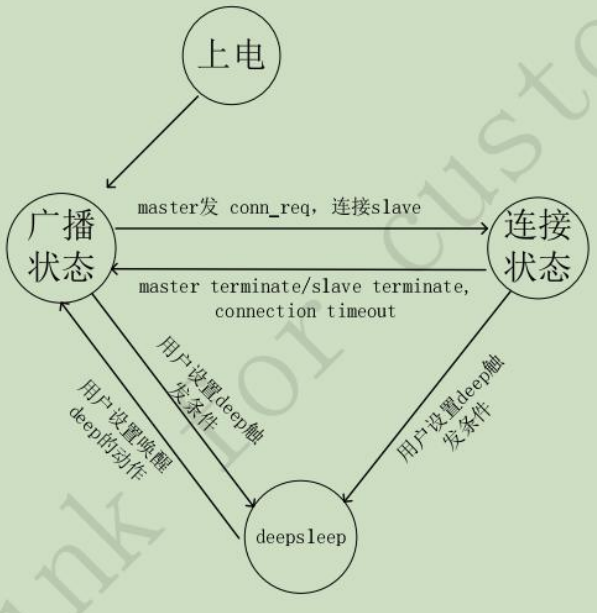


图 3-1 BLE 状态机

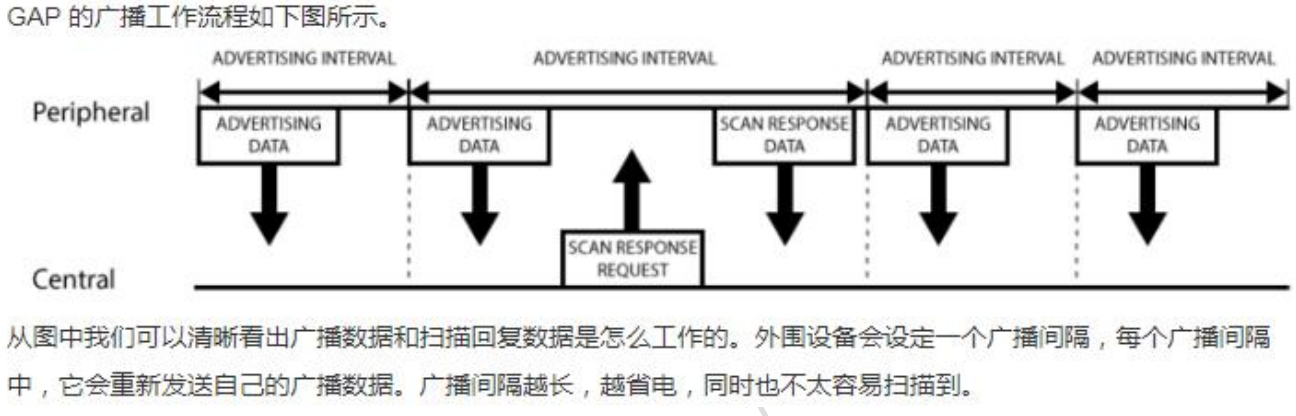
另外，我们还提供了可供灵活编程的蓝牙广播间隔参数 blt_adv_interval ：我们可以根据实际省电的需要在上层程序对 blt_adv_interval 参数进行调整，SDK 内部封装好的函数会调用到 blt_adv_interval 从而达到控制广播间隔的目的。

```
else {
    if(clock_time_exceed(selfie_adv_mode_start_tick,linkLoss_value*100*1000)) {
    if(clock_time_exceed(selfie_adv_mode_start_tick,6000*1000)){
        if(proshutter_disconnect_state) {
            u8 disconnect = 1;
            task_connection_terminated(&disconnect);
        }
    }
    if(clock_time_exceed(selfie_adv_mode_start_tick,6*1000*1000)) {
        blt_adv_interval = ((rand()%20) + 600)*CLOCK_SYS_CLOCK_1MS;
    }
    else if(clock_time_exceed(selfie_adv_mode_start_tick,4*1000*1000)) {
        blt_adv_interval = ((rand()%20) + 50)*CLOCK_SYS_CLOCK_1MS;
    }
    else if(clock_time_exceed(selfie_adv_mode_start_tick,2*1000*1000)) {
        blt_adv_interval = ((rand()%20) + 500)*CLOCK_SYS_CLOCK_1MS;
    }
    else {
        blt_adv_interval = ((rand()%5) + 15)*CLOCK_SYS_CLOCK_1MS;
    }
}
```

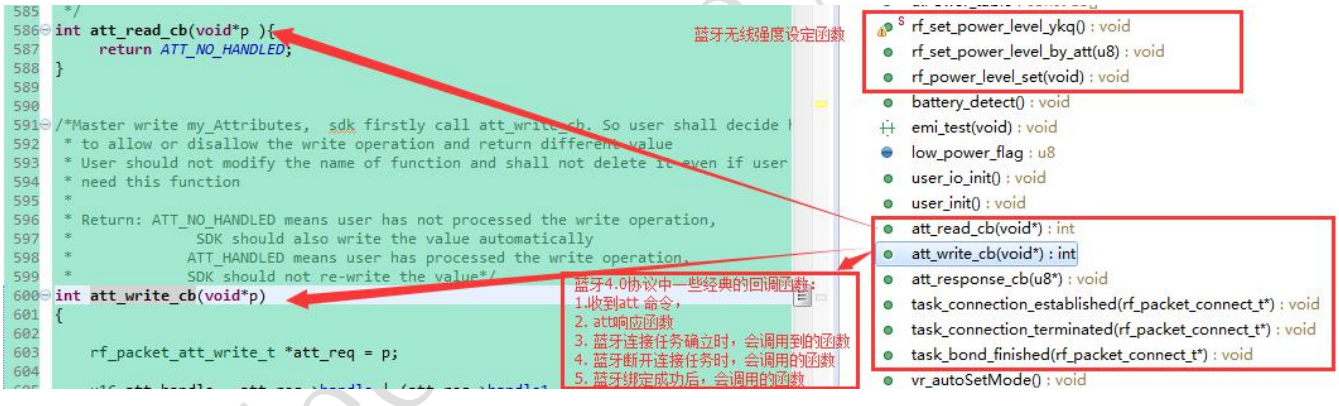
此外，我们还可以通过调用下图标注的函数：blt_update_connPara_request(u16 min_interval, u16 max_interval, u16 latency, u16 timeout),来设置我们需要的连接间隔参数。

```
        if((device_status_ykq==CONNECTED_DEVICE_STATUS_YKQ) && clock_time_exceed(tick_connected_timer_ykq,6*1000*1000))// 1
        {
            if(blt_fifo_num()<3) {
                device_status_ykq = AFTER_CONNECTED_DEVICE_STATUS_YKQ;
//              blt_update_connPara_request(400,480,2,600);
//              blt_update_connPara_request(240,288,4,600);
                blt_update_connPara_request(200,220,4,600); 蓝牙连接间隔参数设置
            }
        }
```

连接参数的作用方式如下，从图中，我们可以清晰地看出广播数据和扫描回复数据是怎么工作的。广播间隔越长，越省电，同时也不大容易被扫描到。



下图圈注的部分是几个蓝牙建立连接时重要的函数：



针对每一个回调函数的介绍，读者可以查看《AN_BLE-16051801-C1_Telink 8266 BLE SDK Developer Handbook .pdf》.

3.3 PWM 设定的实现方法

PWM 频率范围：16Mhz ~ 1hz

PWM 电流驱动能力：0.7mA~4mA

为了测试本 PWM 模块能达到的最大频率波形，我们修改程序为如下：

```
    write_reg8(0x781,0x00);//32M/(15+1) = 2M
    /* set buzzer pwm */
//  write_reg16(0x79a,727);//set pwm3 max cycle 2.7K = 2M/741
//  write_reg16(0x798,363);//set pwm3 duty_cycle = 50% = 370/1000
    write_reg16(0x79a,0x02);//set pwm3 max cycle 2.7K = 2M/741
    write_reg16(0x798,(0x02>>1));//set pwm3 duty_cycle = 50% = 370/1000

    gpio_set_output_en(GPIO_GP10,1);
    gpio_set_input_en(GPIO_GP10,0);
    gpio_set_func(GPIO_GP10,AS_PWM);
```

在数据手册中，找到支持 PWM 功能的 IO 口，我们选择 PWM1- GP10

6	GP8/pwm0_inv #	Digital I/O	GPIO8/PWM0 inverting output
7	GP9/pwm0 #	Digital I/O	GPIO9/PWM0 output
8	GP10/pwm1 #	Digital I/O	GPIO10/PWM1 output
9	GP11/pwm1_inv #	Digital I/O	GPIO11/PWM1 inverting output

11.2 Enable PWM

Register PWM_EN (address 0x780)[3:0] serves to enable PWM3~PWM0 respectively via writing "1" for the corresponding bits.

11.3 Set PWM clock

PWM clock derives from system clock. Register PWM_CLK (address 0x781) serves to set the frequency dividing factor for PWM clock. Formula below applies:

$$F_{PWM} = F_{System\ clock} / (PWM\_CLK+1)$$

根据上面芯片数据手册上的描述，在下图 IO 初始化程序中使能 PWM 口(例如 write_reg(0x780,0x02))，初始化 PWM 模块的基准频率分频系数(例如 write_reg(0x781,0x0f) ，代表将 PWM 模块的时钟频率从 32M 分频 15 ，变成 2M)以及初始化 PWM 口的频率(例如 write_reg(0x79a,727)，代表将 PWM 模块的时钟分频 727，变成 2.75kHz)和高电平持续时间(例如 write_reg(0x79a,756>>1)， 高电平时间是周期的一半时可以如此设定)。

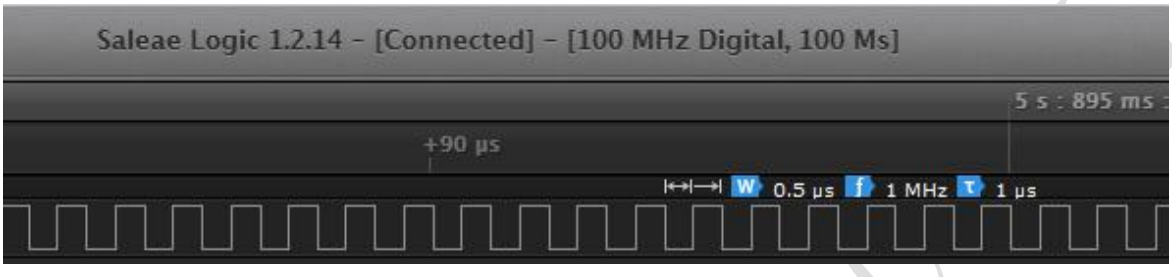
```
538     write_reg8(0x781,0x0f);//32M/(15+1) = 2M
539     /* set buzzer pwm */
540 //  write_reg16(0x79a,727);//set pwm3 max cycle 2.7K = 2M/741
541 //  write_reg16(0x798,363);//set pwm3 duty_cycle = 50% = 370/1000
542     write_reg16(0x79a,buzzer_freq);//set pwm3 max cycle 2.7K = 2M/741
543     write_reg16(0x798,(buzzer_freq>>1));//set pwm3 duty_cycle = 50% = 370/1000
544
545     gpio_set_output_en(GPIO_GP10,1);
546     gpio_set_input_en(GPIO_GP10,0);
547     gpio_set_func(GPIO_GP10,AS_PWM);
```

示波器查看到的波形如下。

2）为了得到 1MHZ 的波形，我们修改程序为如下

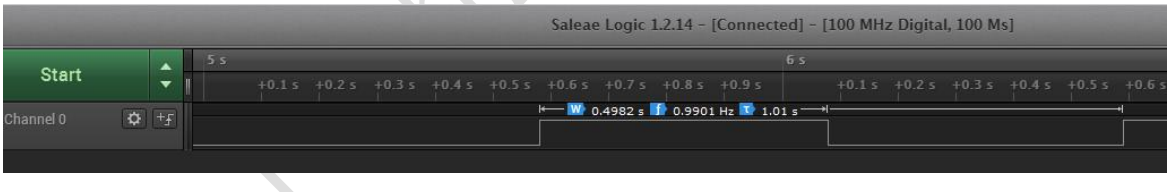
```
537
538     write_reg8(0x781,0x0f);//32M/(15+1) = 2M
539     /* set buzzer pwm */
540 //  write_reg16(0x79a,727);//set pwm3 max cycle 2.7K = 2M/741
541 //  write_reg16(0x798,363);//set pwm3 duty_cycle = 50% = 370/1000
542     write_reg16(0x79a,0x02);//set pwm3 max cycle 2.7K = 2M/741
543     write_reg16(0x798,(0x02>>1));//set pwm3 duty_cycle = 50% = 370/1000
544
545     gpio_set_output_en(GPIO_GP10,1);
546     gpio_set_input_en(GPIO_GP10,0);
547     gpio_set_func(GPIO_GP10,AS_PWM);
548 }
```

示波器查看到的波形如下。



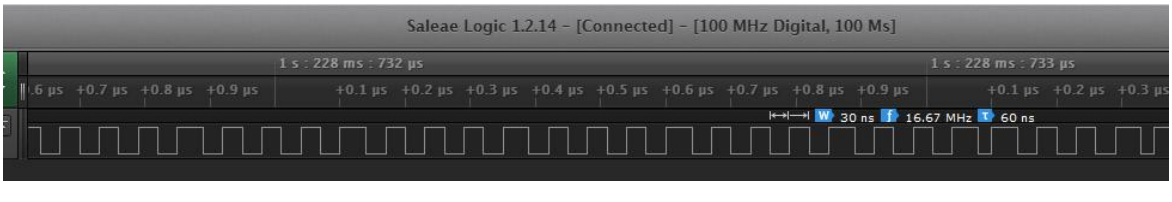
3）测试 PWM 的最小频率

```
538     write_reg8(0x781,0xff);//32M/(15+1) = 2M
539     /* set buzzer pwm */
540 //  write_reg16(0x79a,727);//set pwm3 max cycle 2.7K = 2M/741
541 //  write_reg16(0x798,363);//set pwm3 duty_cycle = 50% = 370/1000
542     write_reg16(0x79a,0xe848 );//set pwm3 max cycle 2.7K = 2M/741
543     write_reg16(0x79b,0x01 );//set pwm3 max cycle 2.7K = 2M/741
544     write_reg16(0x798,0xf424);//set pwm3 duty_cycle = 50% = 370/1000
545
546     gpio_set_output_en(GPIO_GP10,1);
547     gpio_set_input_en(GPIO_GP10,0);
548     gpio_set_func(GPIO_GP10,AS_PWM);
549 }
```



4） 测试 PWM 的最大频率

```
537
538     write_reg8(0x781,0x00);//32M/(15+1) = 2M
539     /* set buzzer pwm */
540 //  write_reg16(0x79a,727);//set pwm3 max cycle 2.7K = 2M/741
541 //  write_reg16(0x798,363);//set pwm3 duty_cycle = 50% = 370/1000
542     write_reg16(0x79a,0x02 );//set pwm3 max cycle 2.7K = 2M/741
543     //write_reg16(0x79b,0x01 );//set pwm3 max cycle 2.7K = 2M/741
544     write_reg16(0x798,0x01);//set pwm3 duty_cycle = 50% = 370/1000
545
546     gpio_set_output_en(GPIO_GP10,1);
547     gpio_set_input_en(GPIO_GP10,0);
548     gpio_set_func(GPIO_GP10,AS_PWM);
549 }
```



我们发现波形频率有些不稳定，所以虽然可以将频率设计得超过 1Mhz 但并不稳定。

3.4 ADC 模块的使用

ADC 引脚输入电压范围：0~1.8V

ADC 采样频率：小于 5MHZ

$$F_{ADCclock} = F_{FHX} * adc\_step[10:0]/adc\_mod[11:0]$$

ADC 采样转换的数值范围：

[4:3] ADC resolution select
00:7bit 01:8bit 10:9bit 11:10bit
[5] Select sign of ADC output data bit<9>
0: positive 1: negative

10 SAR ADC

The ST17HXX integrates one ADC module, which can be used to sample battery voltage and external analog input.

10.1 Register table

Table 10- 1 Register table for SAR ADC

Address	Mnemonic	Type	Description	Reset Value
Digital Registers				
0x2b	ADCREF	RW	[0]select reference 0:Vbg 1:VDDH [7:1] rsvd	0x03
			Analog inputs select bit [2:0] sel ana input 000:close all 001:GP17	

基于上图中数据手册上对 ADC 的描述，我们写出下面的初始化和采样程序。

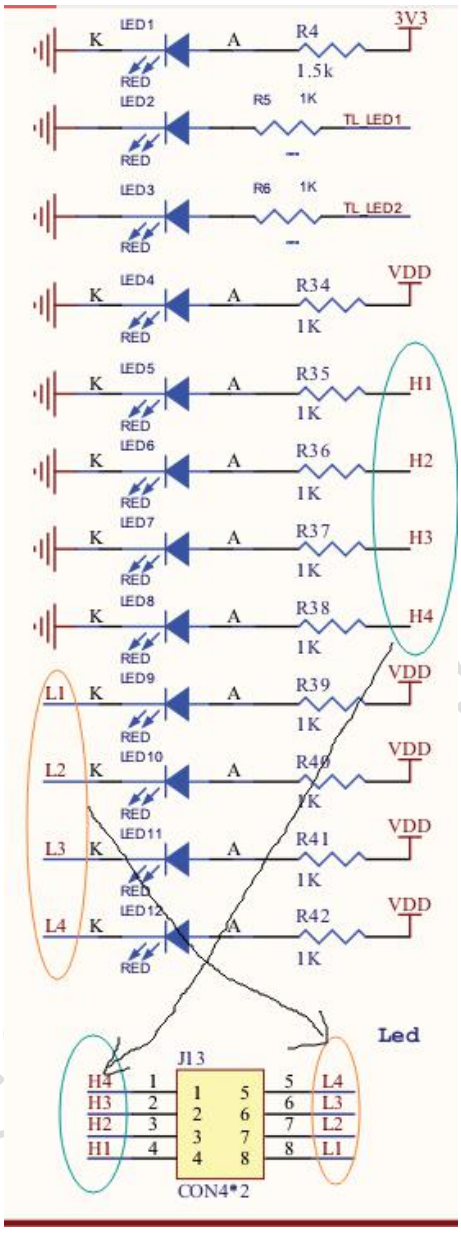
```
14
15  void adc_init ( ) {
16
17      reg_adc_chn_input = ADC_CHNM_ANA_INPUT;// adc chn
18      reg_adc_ref = ADC_CHNM_REF_SRC;  //ref
19
20      reg_adc_sample_clk_res = MASK_VAL(FLD_ADC_SAMPLE_CLOCK,ADC_CLOCK_CYCLE_3,
21        FLD_ADC_RESOLUTION,ADC_RES_10_BITS,
22        FLD_ADC_DATA_SIGN,ADC_DATA_SIGN_POSITIVE
23        );
24
25      //set clk  enable
26      reg_adc_step_l = 4;
27      reg_adc_mod = MASK_VAL(FLD_ADC_MOD, 192, FLD_ADC_CLK_EN, 1);
28
29  }
30
31  u16 adc_get () {
32      // Set a run signal
33      reg_adc_outp = FLD_ADC_OUTPUT_CHN_MANU_START;
34
35      // wait for data
36      sleep_us(5);
37
38      // read data
39      return (reg_adc_dat_byp_outp & 0x3FF);
40  }
```

1.设置每 3 个 ADC 时钟周期采样一次，（FLD_ADC_SAMPLE_CLOCK,*ADC_CLOCK_CYCLE_3*），
2.设置采样到的电压值用 10bits 的数据来表示（FLD_ADC_RESOLUTION,*ADC_RES_10_BITS*），
3.设置 ADC 采样数据的正负（FLD_ADC_DATA_SIGN,*ADC_DATA_SIGN_POSITIVE*）
4.设置 ADC 采样模式 192MHz 时钟频率， *(FLD_ADC_MOD*, 192),
5.设置 ADC 时钟使能，*(FLD_ADC_CLK_EN*, 1).

在初始化成功的基础上，只要调用 adc_get() 子程序即可获取采样电压值。

```
u16 bat_min = adc_get();
u16 bat_max = adc_get();
for(u8 i=0;i<18;i++){
    u16 temp_bat = adc_get();
    if(temp_bat > bat_max) {
        bat_max = temp_bat;
    }
    if(temp_bat < bat_min){
        bat_min = temp_bat;
    }
    sum_bat += temp_bat;
}
u16 batt = (u16)((sum_bat- bat_min - bat_max)>>4);
```

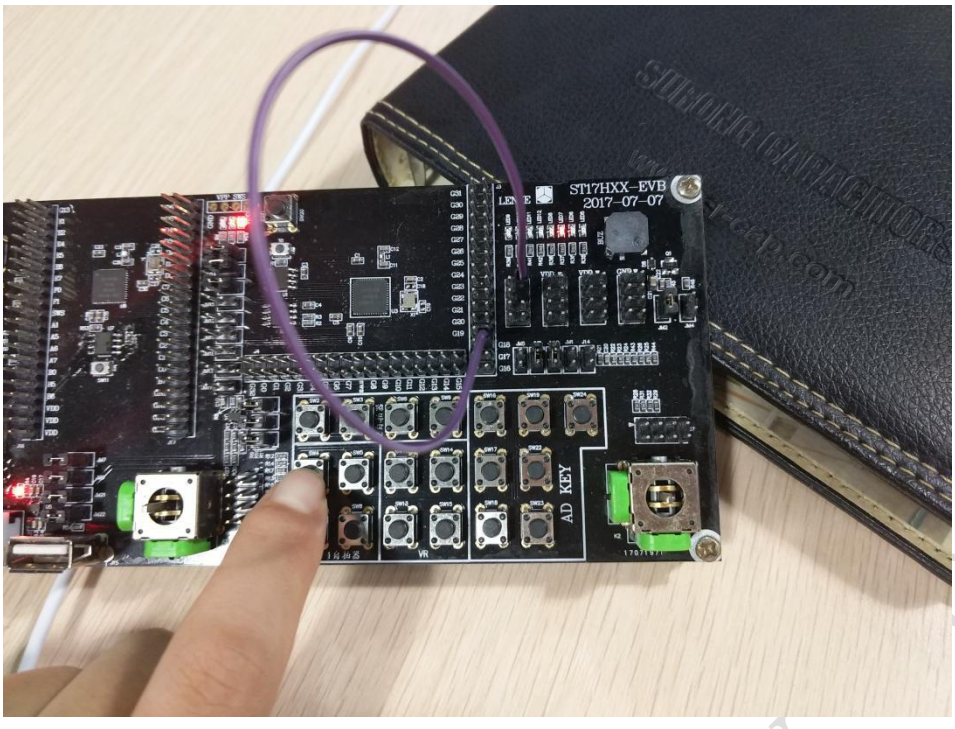
四）硬件测试方法



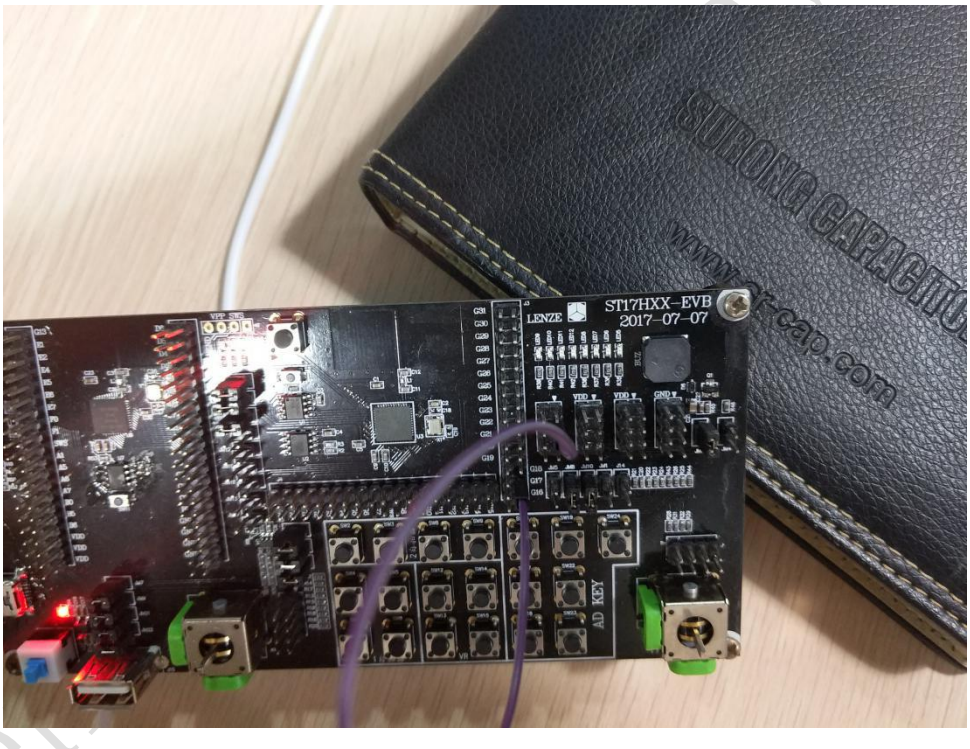
为了检测本例中，我们是否将 GPIO7 口成功设置为上拉电阻。我们将 GPIO7 口连接到上图的 H4 引脚，当按键 SW4 被按下时,GPIO7 的电位被拉高至 VDD，从而接通 LED7；而当按键未被按下时，GPIO7 的电位经内部上拉电阻接到 GND,从而关闭 LED7。

效果展示

按键 SW4 按下后 LED7 点亮

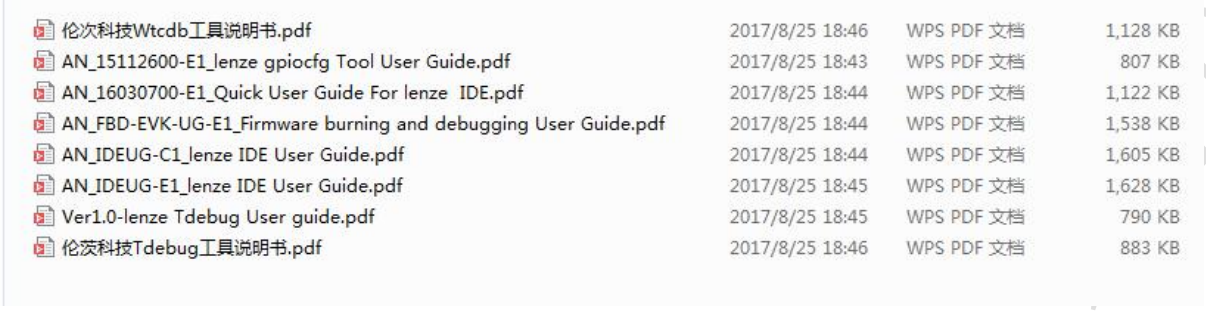


按键 SW4 放开后 LED8 灭。

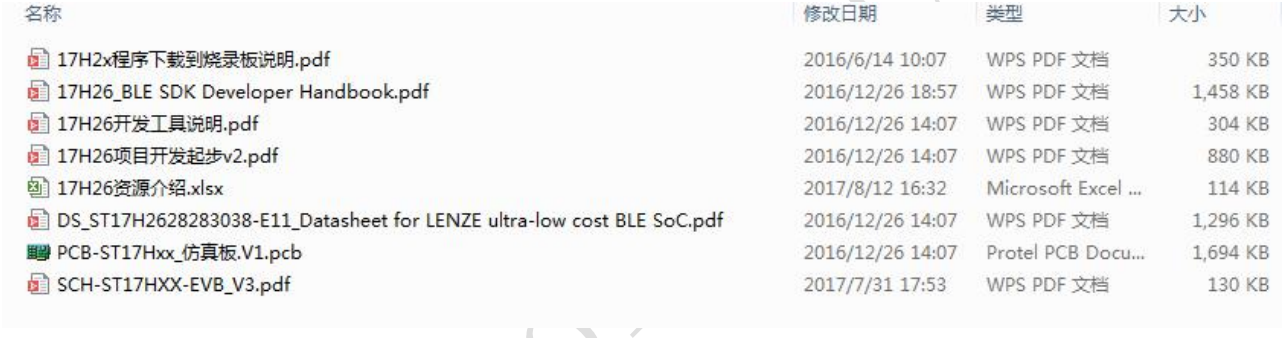


五）参考附录

由于作者写得比较简单，有不够清楚之处，可以查询我们的下列附录，获取更多更精确内容。



伦次科技Wtcdb工具说明书.pdf	2017/8/25 18:46	WPS PDF 文档	1,128 KB
AN_15112600-E1_lenze gpiocfg Tool User Guide.pdf	2017/8/25 18:43	WPS PDF 文档	807 KB
AN_16030700-E1_Quick User Guide For lenze IDE.pdf	2017/8/25 18:44	WPS PDF 文档	1,122 KB
AN_FBD-EVK-UG-E1_Firmware burning and debugging User Guide.pdf	2017/8/25 18:44	WPS PDF 文档	1,538 KB
AN_IDEUG-C1_lenze IDE User Guide.pdf	2017/8/25 18:44	WPS PDF 文档	1,605 KB
AN_IDEUG-E1_lenze IDE User Guide.pdf	2017/8/25 18:45	WPS PDF 文档	1,628 KB
Ver1.0-lenze Tdebug User guide.pdf	2017/8/25 18:45	WPS PDF 文档	790 KB
伦茨科技Tdebug工具说明书.pdf	2017/8/25 18:46	WPS PDF 文档	883 KB



名称	修改日期	类型	大小
17H2x程序下载到烧录板说明.pdf	2016/6/14 10:07	WPS PDF 文档	350 KB
17H26_BLE SDK Developer Handbook.pdf	2016/12/26 18:57	WPS PDF 文档	1,458 KB
17H26开发工具说明.pdf	2016/12/26 14:07	WPS PDF 文档	304 KB
17H26项目开发起步v2.pdf	2016/12/26 14:07	WPS PDF 文档	880 KB
17H26资源介绍.xlsx	2017/8/12 16:32	Microsoft Excel ...	114 KB
DS_ST17H2628283038-E11_Datasheet for LENZE ultra-low cost BLE SoC.pdf	2016/12/26 14:07	WPS PDF 文档	1,296 KB
PCB-ST17Hxx_仿真板.V1.pcb	2016/12/26 14:07	Protel PCB Docu...	1,694 KB
SCH-ST17HXX-EVB_V3.pdf	2017/7/31 17:53	WPS PDF 文档	130 KB

预祝使用愉快！