摘 要

本技术文档旨在帮助客户快速配置 MCU CS32F03x 低功耗模式。 Deepsleep 模式是应用较为广泛的一个低功耗模式，需要达到理想的低电流值，需要正确的配置 MCU 内部外设和 GPIO 的电平状态。文档中提到了在低功耗模式下 MCU 配置的基本原则，以及如何处理 MCU Deepsleep 电流一致性的问题。

版 本

历史版本	修改内容	日期
V1.0	初版生成	2022-09-20

目 录

1 硬件介绍和低功耗参数

评估板：CS32F03xx EVB(红框中的 J6 可以串入万用表测试电流)

调试器：J-Link

万用表 ：

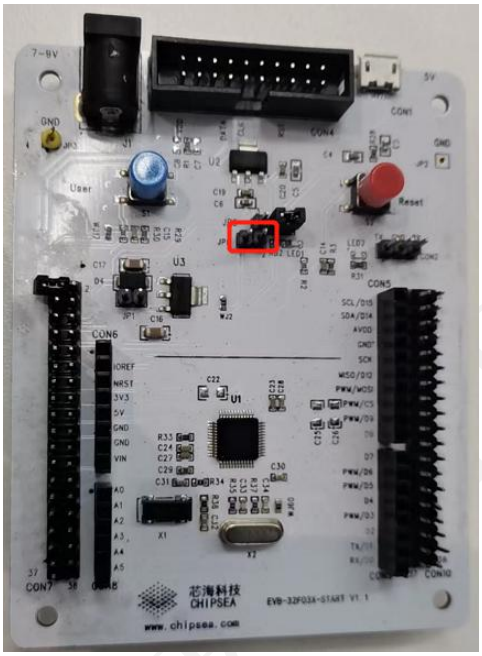

图 1 CS32F03x 开发板 CS32F03x 开发板

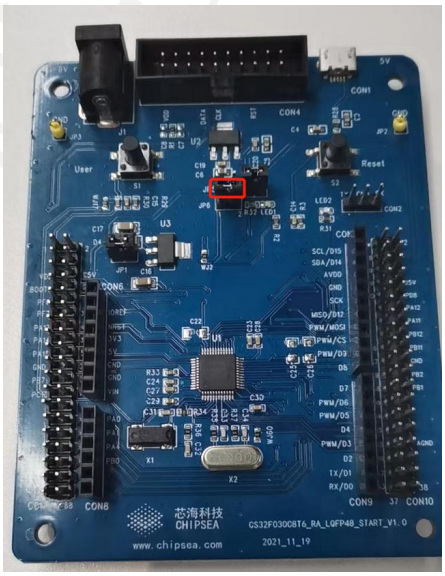

图 2 CS32F03x-RA 开发板

图 3　J-Link 调试器

图 4　万用表

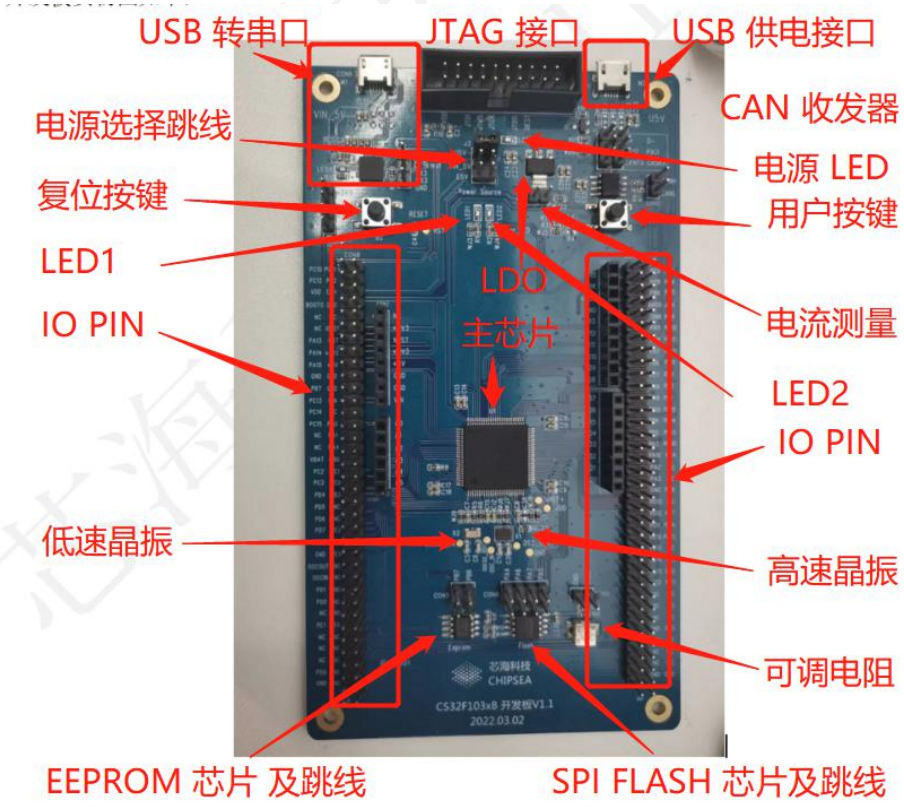

图 5　CS32F103xB 开发板 V1.1

CS32F03X 低功耗模式，支持睡眠，深度睡眠 1，深度睡眠 2 和待机模式 四种模式。深度睡眠模式 2 对应 Deepsleep，保留 RTC 和 EXIT 外部中断唤醒功能，这是最常用的低功耗模式。

参考规格书， Deepsleep 模式下理想电流是 6.81uA(5.6 uA+ 1.21uA),考虑到实际情况，实际测量电流在 7uA ~ 12uA 都是正常值。

表 1　深度睡眠和待机模式功耗参数

运行模式	代码执行位置	条件	f_{HCLK} (MHz)	IVDD (外设打开) (μA)			IVDD (外设关闭) (μA)			IVDDA (μA)		
				Typ(1)	Max(2)	Max(3)	Typ(1)	Max(2)	Max(3)	Typ(1)	Max(2)	Max(3)
深睡模式 1	-	VDDA monitor 打开	-	21.3	42	-				2.73	15.8	
深睡模式 1	-	VDDA monitor 关闭	-	21.3	42	-				1.24		
深睡模式 2	-	VDDA monitor 打开	-	5.6	21	-				2.71	15.8	
深睡模式 2	-	VDDA monitor 关闭	-	5.6	21	-				1.21		
待机模式	-	LRC 打开, FWDT 打开	-	1.2	-	-				3.6		
待机模式	-	VDDA monitor 打开	-	1.0	-	-				2.3		
待机模式	-	LRC 关闭, FWDT 关闭	-	1.2	-	-				2.5		

如果需要功耗更低，可以使用待机模式 Standby. 待机模式下，内部 Regulator 调压器关闭，因此整个 1.5 V 域将断电。进入待机模式后，除 RTC 域和待机电路中的寄存器外，SRAM 和寄存器的内容都将消失。 PLL、 HSI 和 HSE 晶振也会关闭。当发生外部复位（NRST 引脚）、 IWDG 复位、 WKUP 引脚上出现上升沿或者触发 RTC 事件时，器件退出待机模式。PA0 支持 WKUP1，PC13 支持 WKUP2(仅 LQFP48 封装有 PC13)。 另外睡眠模式和待机模式下的唤醒时间也不相同，请参考下表。

表 2　低功耗模式下的唤醒时间参数

符号	描述	最小值	典型值.	最大值	单位
$T_{wk\text{-}sleep}$	睡眠模式的唤醒时间	-	5 system clk	-	μS
$T_{wk\text{-}deepsleep1}$	深度睡眠 1 模式下的唤醒时间	-	3	5.3	μS
$T_{wk\text{-}deepsleep2}$	深度睡眠 2 模式下的唤醒时间	-	4	7.2	μS
$T_{wk\text{-}powerdown}$	待机模式下的唤醒时间	-	57	157	μS

2 基于 SDK 2.0.5 测试

PMU Deepsleep 例程在以下路径：

CS32F03x_DFP.2.0.5\Boards\EVB_32F03x_START\Examples\PMU\PMU_DeepSleep\MDK-ARM

例程编译下载， EVB 上 LED2 会持续快速闪烁。 将万用表串入 开发板上 J6 测量电流，大概在 10mA 左右跳动。 按下开发板上的 S1 按键， LED off， MCU 进入 Deepsleep 模式，万用表测量电流大概是 10uA. 按下 S1 按键后，MCU 从 Deepsleep 模式唤醒， LED2 会持续闪烁，再次按下 S1 按键， MCU 会再次进入 Deepsleep 模式。

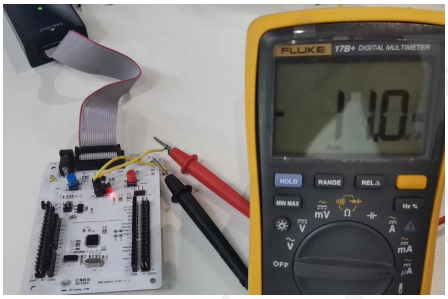

图 6 CS32F03x 开发板 Deepsleep 电流测量

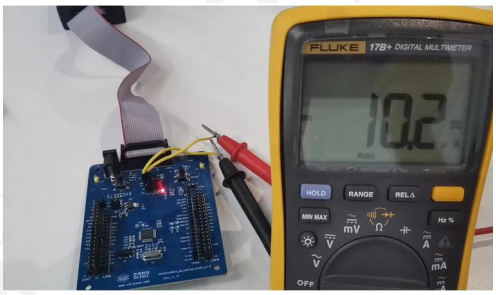

图 7 CS32F03x-RA 开发 Deepsleep 电流测量

```
 - Hardware Description
    CS32F03X
    --------------
    |          |
    |        PA5|-->LED2
    |          |
    |       PC13|<--KEY S1
    |          |
    --------------

int main(void)
{
   /* Configures Systick interrupt at 1ms as hardware delay source */
   SysTick_Config(SystemCoreClock / 1000);

   while (1)
   {
              /* Configure LED */
              cs_eval_led_init();
               cs_eval_led_on();

              /* Configure KEY S1 as external interrupt generator */
```

```
            cs_eval_key_init();

            cs_eval_key_exti_init();

           key_status = 0;

            /* Wait until KEY S1 is pressed to enter the Low Power mode.
            In the meantime, LED2 is blinks */
             while(key_status == 0)
             {
                    cs_eval_led_toggle();
                    delay(100);
             }

              /* Make sure LED2 is turned off to reduce low power mode consumption */
              cs_eval_led_off();

              /* Configures peripherals and enter deep sleep */
             deep_sleep_measure();

             /* Configures system clock after wake-up from deep sleep */
              sysclk_config_from_deep_sleep();
  }
}
```

GPIO 状态配置如下：

```
__RCU_AHB_CLK_ENABLE(RCU_AHB_PERI_GPIOA | RCU_AHB_PERI_GPIOB | RCU_AHB_PERI_GPIOC | RCU_AHB_PERI_GPIOF);

                        // MAX GPIO 39
                        // SWD PA13 PA14
                        // HXT_IN HXT_OUT PF0 PF1
                        // LXT_IN LXT_OUT PC13 PC15

                        // PA0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
                        // PB0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
                        // PC13 14 15
                        // PF0 1 6 7

  //          gpio_mode_set(GPIOA,GPIO_PIN_ALL,GPIO_MODE_ANALOG);
  //          gpio_mode_set(GPIOB,GPIO_PIN_ALL,GPIO_MODE_ANALOG);
  //          gpio_mode_set(GPIOC,GPIO_PIN_ALL,GPIO_MODE_ANALOG);
  //          gpio_mode_set(GPIOF,GPIO_PIN_ALL,GPIO_MODE_ANALOG);
              gpio_mode_set(GPIOA,GPIO_PIN_ALL,GPIO_MODE_OUT_PP(GPIO_SPEED_LOW));
              gpio_mode_set(GPIOB,GPIO_PIN_ALL,GPIO_MODE_OUT_PP(GPIO_SPEED_LOW));
              gpio_mode_set(GPIOC,GPIO_PIN_ALL,GPIO_MODE_OUT_PP(GPIO_SPEED_LOW));
              gpio_mode_set(GPIOF,GPIO_PIN_ALL,GPIO_MODE_OUT_PP(GPIO_SPEED_LOW));

              __GPIO_PIN_RESET(GPIOA, GPIO_PIN_ALL);
              __GPIO_PIN_RESET(GPIOB, GPIO_PIN_ALL);
              __GPIO_PIN_RESET(GPIOC, GPIO_PIN_13 | GPIO_PIN_14 | GPIO_PIN_15);
              __GPIO_PIN_RESET(GPIOF, GPIO_PIN_0 | GPIO_PIN_1 | GPIO_PIN_6 | GPIO_PIN_7);

      /* Closes all peripherals. */
              __RCU_AHB_CLK_DISABLE(RCU_AHB_PERI_GPIOA|RCU_AHB_PERI_GPIOB|RCU_AHB_PERI_GPIOC|RCU_AHB_PERI_GP
IOF|RCU_AHB_PERI_CRC|RCU_AHB_PERI_DMA);

              __RCU_APB1_CLK_DISABLE(RCU_APB1_PERI_TIM2|RCU_APB1_PERI_TIM3|RCU_APB1_PERI_TIM6|RCU_APB1_PERI_TIM
14|RCU_APB1_PERI_WWDT|
      RCU_APB1_PERI_SPI2|RCU_APB1_PERI_USART2|RCU_APB1_PERI_I2C1|RCU_APB1_PERI_I2C2|RCU_APB1_PERI_PMU);

              __RCU_APB2_CLK_DISABLE(RCU_APB2_PERI_SYSCFG|RCU_APB2_PERI_USART6|RCU_APB2_PERI_USART7|RCU_APB2_
PERI_USART8|RCU_APB2_PERI_ADC|

RCU_APB2_PERI_TIM1|RCU_APB2_PERI_SPI1|RCU_APB2_PERI_USART1|RCU_APB2_PERI_TIM15|RCU_APB2_PERI_TIM16|RCU_APB
2_PERI_TIM17|RCU_APB2_PERI_DBG);
```

关闭不用的时钟，如下：

```
/* In order to close PLL, set sysclk source is HRC*/
  rcu_sysclk_config(RCU_SYSCLK_SEL_HRC);

  /* Closes other no used clock. */
   __RCU_FUNC_DISABLE(HXT_CLK);
  __RCU_FUNC_DISABLE(HXT_BYPASS);
  while(__RCU_FLAG_STATUS_GET(HXT_STAB) == SET);

   __RCU_FUNC_DISABLE(LXT_CLK);
   __RCU_FUNC_DISABLE(LXT_BYPASS);
  while(__RCU_FLAG_STATUS_GET(LXT_STAB) == SET);

  __RCU_FUNC_DISABLE(LRC_CLK);
  while(__RCU_FLAG_STATUS_GET(LRC_STAB) == SET);

   __RCU_FUNC_DISABLE(HRC14_CLK);
  while(__RCU_FLAG_STATUS_GET(HRC14_STAB) == SET);

  __RCU_FUNC_DISABLE(PLL_CLK);
  while(__RCU_FLAG_STATUS_GET(PLL_STAB) == SET);

   __RCU_FUNC_DISABLE(RTC_CLK);

//              cs_eval_key_init();
//   cs_eval_key_exti_init();

   __RCU_APB1_CLK_ENABLE(RCU_APB1_PERI_PMU); // Enable the PMU clock
  pmu_deep_sleep_mode_enter(PMU_LDO_LOW_POWER, PMU_DSM_ENTRY_WFI); // enter STOP mode
```

3 基于 SDK 1.0.8 测试

将如下 HAL_Examples 下的 4 个文件 复制替换到工程目录下：

CS32F03X_SDK.1.0.8\Boards\EVB_32F03x_START\HAL_Examples\PMU\PMU_DeepSleep

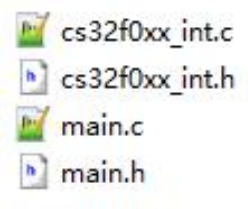

图 8　需要复制的 4 个文件

复制到如下路径：

CS32F03X_SDK.1.0.8\Boards\EVB_32F03x_START\EVB_32F03x_START_DEMO\source\User_Project

编译会提供报错：

```
.\Objects\cs32f0xx_demo.axf: Error: L6200E: Symbol delay multiply defined (by systick.o and main.o).
```

删除下图中的 c 文件后，重新编译后不会再报错：

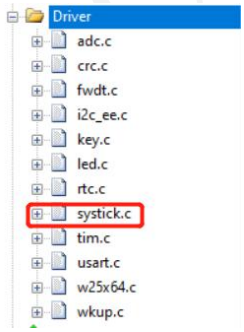

图 9　编译报错，需要删除的 c 文件

Main 函数代码如下：

```
int main(void)
{
	exti4_15_config();
	cs_eval_led2_init();
	SysTick_Config(SystemCoreClock / 1000);//SysTick interrupt each 1 ms

	rtc_config();
	cs_eval_led2_on();


	gpio_pmu_config();

	while(1)
	{
		delay(1000);
		rtc_alarm_config();  //Set alarm in 20s
		cs_eval_led2_off();

		pmu_deep_sleep_mode_enter(PMU_LDO_LOW_POWER, PMU_DSM_ENTRY_WFI); // enter STOP mode
		cs_eval_led2_on();
```

```
                //Disable the RTC Alarm interrupt
                rtc_interrupt_config(RTC_INTERRUPT_ALR, DISABLE);
                rtc_alarm_enable_ctrl(RTC_ALARM_ENABLE, DISABLE);

                config_sysclk_from_deep_sleep();  // Configures system clock after wake-up from STOP
        }
}
```

程序编译下载到开发板，测试电流测量大概 14uA.

增加 GPIO 状态配置后，重新编译下载， 功耗测量大概是 10uA.

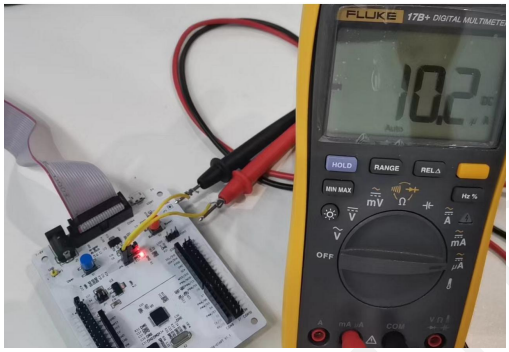

图 10　CS32F03x 开发板 Deepsleep 电流测量

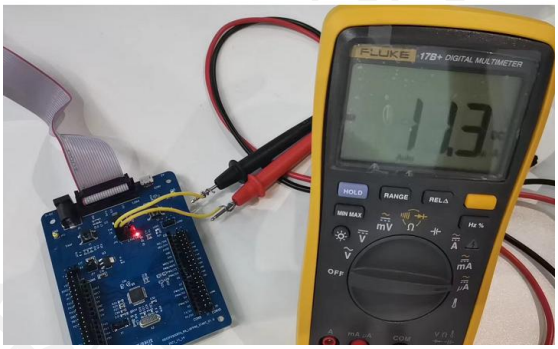

图 11　CS32F03x-RA 开发板 Deepsleep 电流测量

GPIO 电平状态配置函数如下：

```
static void gpio_pmu_config(void)
{
  gpio_config_t gpio_config_struct;

                // Enable GPIO Clock
                rcu_ahb_periph_clock_enable_ctrl(RCU_AHB_PERI_PORTA, ENABLE);
                rcu_ahb_periph_clock_enable_ctrl(RCU_AHB_PERI_PORTB, ENABLE);
                rcu_ahb_periph_clock_enable_ctrl(RCU_AHB_PERI_PORTC, ENABLE);
                rcu_ahb_periph_clock_enable_ctrl(RCU_AHB_PERI_PORTD, ENABLE);
                rcu_ahb_periph_clock_enable_ctrl(RCU_AHB_PERI_PORTE, ENABLE);
                rcu_ahb_periph_clock_enable_ctrl(RCU_AHB_PERI_PORTF, ENABLE);

                // MAX GPIO 39
                // SWD PA13 PA14
                // HXT_IN HXT_OUT PF0 PF1
                // LXT_IN LXT_OUT PC13 PC15

                // PA0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
                // PB0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
                // PC13 14 15
                // PF0 1 6 7
```

```
                    //gpio_config_struct.gpio_pin = GPIO_PIN_0 | GPIO_PIN_1  | GPIO_PIN_2 | GPIO_PIN_3 | GPIO_PIN_4 | GPIO_PIN_5 |
GPIO_PIN_6 | GPIO_PIN_7 | GPIO_PIN_8 | GPIO_PIN_9 | GPIO_PIN_10 | GPIO_PIN_11 | GPIO_PIN_12 | GPIO_PIN_13 | GPIO_PIN_14 |
GPIO_PIN_15;
                    gpio_config_struct.gpio_pin = GPIO_PIN_0 | GPIO_PIN_1  | GPIO_PIN_2 | GPIO_PIN_3 | GPIO_PIN_4 | GPIO_PIN_5 |
GPIO_PIN_6 | GPIO_PIN_7 | GPIO_PIN_8 | GPIO_PIN_9 | GPIO_PIN_10 | GPIO_PIN_11 | GPIO_PIN_12 | GPIO_PIN_12 | GPIO_PIN_12 |
GPIO_PIN_15;
                    gpio_config_struct.gpio_mode = GPIO_MODE_OUTPUT;
                    gpio_config_struct.gpio_speed = GPIO_SPEED_HIGH;
                    gpio_config_struct.gpio_out_type = GPIO_OUTPUT_PP;
                    gpio_config_struct.gpio_pull = GPIO_PULL_NO_PULL;
                    gpio_init(GPIOA, &gpio_config_struct);

                    // Config AIN0~AIN7
                    gpio_config_struct.gpio_pin = GPIO_PIN_0 | GPIO_PIN_1  | GPIO_PIN_2 | GPIO_PIN_3 | GPIO_PIN_4 | GPIO_PIN_5 |
GPIO_PIN_6 | GPIO_PIN_7;
                    gpio_config_struct.gpio_mode = GPIO_MODE_AN;
                    gpio_config_struct.gpio_pull = GPIO_PULL_NO_PULL;
                    gpio_init(GPIOA, &gpio_config_struct);

                    gpio_bits_reset(GPIOA,GPIO_PIN_0 | GPIO_PIN_1  | GPIO_PIN_2 | GPIO_PIN_3 | GPIO_PIN_4 | GPIO_PIN_5 |
GPIO_PIN_6 | GPIO_PIN_7 | GPIO_PIN_8 | GPIO_PIN_9 | GPIO_PIN_10 | GPIO_PIN_11 | GPIO_PIN_12 | GPIO_PIN_13 | GPIO_PIN_14 |
GPIO_PIN_15);
//                  gpio_bits_reset(GPIOA,GPIO_PIN_0 | GPIO_PIN_1  | GPIO_PIN_2 | GPIO_PIN_3 | GPIO_PIN_4 | GPIO_PIN_5 |
GPIO_PIN_6 | GPIO_PIN_7 | GPIO_PIN_8 | GPIO_PIN_9 | GPIO_PIN_10 | GPIO_PIN_11 | GPIO_PIN_12 | GPIO_PIN_12 | GPIO_PIN_12 |
GPIO_PIN_15);


                    gpio_config_struct.gpio_pin = GPIO_PIN_0 | GPIO_PIN_1  | GPIO_PIN_2 | GPIO_PIN_3 | GPIO_PIN_4 | GPIO_PIN_5 |
GPIO_PIN_6 | GPIO_PIN_7 | GPIO_PIN_8 | GPIO_PIN_9 | GPIO_PIN_10 | GPIO_PIN_11 | GPIO_PIN_12| GPIO_PIN_13 | GPIO_PIN_14 |
GPIO_PIN_15;
                    gpio_config_struct.gpio_mode = GPIO_MODE_OUTPUT;
                    gpio_config_struct.gpio_speed = GPIO_SPEED_HIGH;
                    gpio_config_struct.gpio_out_type = GPIO_OUTPUT_PP;
                    gpio_config_struct.gpio_pull = GPIO_PULL_NO_PULL;
                    gpio_init(GPIOB, &gpio_config_struct);

                    gpio_bits_reset(GPIOB,GPIO_PIN_0 | GPIO_PIN_1  | GPIO_PIN_2 | GPIO_PIN_3 | GPIO_PIN_4 | GPIO_PIN_5 |
GPIO_PIN_6 | GPIO_PIN_7 | GPIO_PIN_8 | GPIO_PIN_9 | GPIO_PIN_10 | GPIO_PIN_11 | GPIO_PIN_12 | GPIO_PIN_13 | GPIO_PIN_14 |
GPIO_PIN_15);

                    // Config AIN8~AIN9
                    gpio_config_struct.gpio_pin = GPIO_PIN_0 | GPIO_PIN_1;
                    gpio_config_struct.gpio_mode = GPIO_MODE_AN;
                    gpio_config_struct.gpio_pull = GPIO_PULL_NO_PULL;
                    gpio_init(GPIOB, &gpio_config_struct);

                    gpio_config_struct.gpio_pin = GPIO_PIN_13 | GPIO_PIN_14 |GPIO_PIN_15;
                    gpio_config_struct.gpio_mode = GPIO_MODE_OUTPUT;
                    gpio_config_struct.gpio_speed = GPIO_SPEED_HIGH;
                    gpio_config_struct.gpio_out_type = GPIO_OUTPUT_PP;
                    gpio_config_struct.gpio_pull = GPIO_PULL_NO_PULL;
                    gpio_init(GPIOC, &gpio_config_struct);

                    gpio_bits_set(GPIOC,GPIO_PIN_13 | GPIO_PIN_14 |GPIO_PIN_15);


                    gpio_config_struct.gpio_pin = GPIO_PIN_0 | GPIO_PIN_1 | GPIO_PIN_6 | GPIO_PIN_7;
                    gpio_config_struct.gpio_mode = GPIO_MODE_OUTPUT;
                    gpio_config_struct.gpio_speed = GPIO_SPEED_HIGH;
                    gpio_config_struct.gpio_out_type = GPIO_OUTPUT_PP;
                    gpio_config_struct.gpio_pull = GPIO_PULL_NO_PULL;
                    gpio_init(GPIOF, &gpio_config_struct);

                    gpio_bits_reset(GPIOF,GPIO_PIN_0 | GPIO_PIN_1 | GPIO_PIN_3 | GPIO_PIN_6 | GPIO_PIN_7);

                     //   rcu_ahb_periph_clock_enable_ctrl(RCU_AHB_PERI_PORTA, DISABLE);
                    rcu_ahb_periph_clock_enable_ctrl(RCU_AHB_PERI_PORTB, DISABLE);
                    rcu_ahb_periph_clock_enable_ctrl(RCU_AHB_PERI_PORTC, DISABLE);
                    rcu_ahb_periph_clock_enable_ctrl(RCU_AHB_PERI_PORTD, DISABLE);
                    rcu_ahb_periph_clock_enable_ctrl(RCU_AHB_PERI_PORTE, DISABLE);
                    rcu_ahb_periph_clock_enable_ctrl(RCU_AHB_PERI_PORTF, DISABLE);
}
```

关闭不用的时钟，如下：

```
        rcu_hxt_config(RCU_HXT_OFF);
        rcu_hrc14_enable_ctrl(DISABLE);
        rcu_hrc14_opening_enable_ctrl(DISABLE);
        rcu_lxt_config(RCU_LXT_OFF);
        rcu_hxt_monitor_enable_ctrl(DISABLE);
        rcu_pll_enable_ctrl(DISABLE);
//      rcu_hrc_enable_ctrl(DISABLE);
        rcu_lrc_enable_ctrl(DISABLE);
```

4 低功耗外设配置原则和注意事项

低功耗外设配置，有两个基本原则：

◆ 芯片内部不用的外设，不用的功能要全部关掉。

◆ 芯片 GPIO 要正确配置，避免电流流进或者流出 MCU.

◆ 低功耗调试步骤：

先在开发板，或者芯片 Socket 上调试，或者一个空的 PCBA，只有一个 MCU，这种情况下 GPIO 状态配置较为简单，只要把相关不用的外设关闭，按照上面提到的 GPIO 配置状态，一般只要 MCU 成功进入 Deepsleep 模式，功耗就会降低到比较理想的值。

```
        rcu_hxt_config(RCU_HXT_OFF);
        rcu_hrc14_enable_ctrl(DISABLE);
        rcu_hrc14_opening_enable_ctrl(DISABLE);
        rcu_lxt_config(RCU_LXT_OFF);
        rcu_hxt_monitor_enable_ctrl(DISABLE);
        rcu_pll_enable_ctrl(DISABLE);
//      rcu_hrc_enable_ctrl(DISABLE);
        rcu_lrc_enable_ctrl(DISABLE);
```

然后在目标 PCBA 上按具体硬件的情况，逐个配置 GPIO 状态。建议把 GPIO 列一个表，根据硬件电路确定 GPIO 是设置成 ADC 输入功能，还是设置成 GPIO 输出功能，应该是输出 0 还是输出 1，每个 GPIO 都需要配置，不能让它处于一个默认状态。让 GPIO 处于默认状态，可能会有功耗一致性不好的问题，比如实际生产测试时，可能会有 5% 的 PCBA 功耗偏大。如果有一个 GPIO 配置不正确，可能会有 大概 100uA 的漏电。

MCU 进入 Deep sleep 前，建议把没有用到的外设都关掉一次，包括 PLL， ADC，HIRC, HIRC14 等。部分外设可能是在 运行 ROM ISP 程序中有打开，在进入到 main() 前没有去关掉，所以用户需要主动再把这些外设关一次。

MCU GPIO 配置原则是，有 ADC 功能的，把 GPIO 设置成 模拟输入功能，没有 ADC 功能的， 把 GPIO 设置成输出方向， 输出 0 还是输出 1，要看具体硬件电路，要确保电流不能流进也不会流出 MCU。

注意事项：

CS32F03X 有很多个型号的封装，在配置 GPIO 的状态时，要全部按照 LQFP48 的 GPIO 来配置，不同封装用的是同一个晶圆，小封装的型号， GPIO 数量也是同样的，只是没有引

出到 芯片 PIN 外部来。所以也要给一个确定的电压状态，输出 0 或者输出 1 都可以。如果有一个 IO 口没有配置好，可能会有 大概 100uA 的漏电流。

对于有 MCU 需要进入 Deep sleep 的应用，建议 SWD 调试接口把 NRST PIN 拉出来，和 SWDIO SWDCLK 一起接到调试器。 如果 MCU 进入 Deep sleep 模式， NRST 又没有拉出来， 调试器 J-Link 可能会连接不上 MCU 了。

没有模拟输入功能的 GPIO， 如果不给一个确定的电平状态， 可能会有功耗不一致问题，比如生产 1000PCS， 可能会有 10PCS 功耗大到 100uA 左右的问题发生。同样， 如果是 QFN32 封装的芯片，没有去按 LQFP48 的 GPIO 配置，批量生产时，也可能会有小比例的功耗不一致问题发生。

5 总结

MCU 低功耗的代码配置工作量比较大，基于硬件设计理想的前提下才能完善软件代码。 低功耗配置代码，和硬件相关性较大，在开发板上电流测试理想的代码，放在其它硬件 PCBA 上功耗未必同样理想。针对每个不硬件环境，都需要耐心正确的配置每个 GPIO 的状态，关闭芯片内部已经打开的外设功能，才能达到理想的低功耗电流值。