说明

CH32 系列微控制器提供了多种低功耗模式，本应用文档列举了不同低功耗模式的区别并提供了部分唤醒源将系统从低功耗模式唤醒的使用示例，介绍了从软件配置、硬件设计及测量误差等维度优化低功耗模式下功耗电流的方法，以及睡眠唤醒中的注意事项和使用技巧。

适用范围

适用范围	系列
MCU	CH32 通用 MCU

目录

表格索引

图片索引

第 1 章 低功耗模式

1.1 低功耗模式简介

CH32 系列微控制器提供 SLEEP\STOP\STANDBY 等多种低功耗模式，并提供不同的唤醒方法让系统跳出此状态。当系统不需继续运行时，可以根据系统使用需求选择不同的低功耗模式来节省功耗。

表 1-1 低功耗模式一览

模式	进入	唤醒	描述
SLEEP	WFI	任意中断唤醒	内核时钟关闭，其他时钟无影响所有 I/O 引脚保持，唤醒后继续运行
	WFE	任意事件唤醒	
STOP (1)	SLEEPDEEP 置 1 PDDS 置 0 WFI 或 WFE	任意外部中断/事件 (1)	关闭 HSE、HSI、PLL 和外设时钟，所有 I/O 引脚保持，唤醒后继续运行
STANDBY (1)	SLEEPDEEP 置 1 PDDS 置 1 WFI 或 WFE	任意事件等 (1)	关闭HSE、HSI、PLL和外设时钟，唤醒后继续运行或复位 (1)

注：1.部分产品未提供 STOP 或 STANDBY 模式，不同产品唤醒源有所差别，部分产品 STANDBY 模式唤醒后系统会继续执行，具体细节请查阅芯片手册。

1.1.1 SLEEP 模式

通过执行 WFI 或 WFE 指令进入睡眠状态。在 SLEEP 模式，所有的 I/O 引脚都保持它们在运行模式时的状态。所有的外设时钟都正常，所以进入睡眠模式前，尽量关闭无用的外设时钟，以减低功耗。该模式唤醒所需时间最短。

1.1.2 STOP 模式

此模式高频时钟（HSE/HSI/PLL）域被关闭，SRAM和寄存器内容保持，I/O引脚状态保持。该模式唤醒后系统可以继续运行，HSI为默认系统时钟。如果正在进行闪存编程，直到对内存访问完成，系统才进入停止模式；如果正在进行对PB的访问，直到对PB访问完成，系统才进入停止模式。STOP模式在功耗与灵活性之间取得了平衡，适合大多数低功耗应用。

1.1.3 STANDBY 模式

待机模式下，电压调节器关闭，除唤醒电路和后备域电路之外的电路将断电，实现最低功耗，在指定的唤醒条件下退出后，微控制器将被复位，并执行的是电源复位（部分芯片如CH32V00x、CH32X03x系列，在STANDBY模式下，SRAM和寄存器内容保持，I/O引脚状态保持，系统唤醒后继续执行，HSI默认为系统时钟）。STANDBY模式功耗最低，但通常唤醒后需要重启系统，适用于对实时性要求不高的场景。

1.2 部分唤醒源的使用

1.2.1 RTC 闹钟唤醒

RTC 可以实现无需外部中断的情况下自动唤醒。通过对时间基数进行编程，可周期性地从 STOP 或 STANDBY 模式下唤醒。

以 CH32F/V20x_V30x 为例，可选择精准的外部低频 32.768kHz 晶振 LSE 作为 RTC 时钟源，也可以选择内部 LSI 振荡器作为 RTC 时钟源，LSI 的精度和功耗指标要差于 LSE。如果使用事件唤醒，需要使能外部中断通道 17 的事件请求信号；如果使用中断唤醒，则需要使能外部中断通道 17 的中断请求信号，并且使能闹钟外部中断，编写中断服务函数。STOP 模式支持用事件或中断唤醒，唤醒后系统继续运行。而 STANDBY 模式只能用事件唤醒，且无需配置外部中断线 17，唤醒后系统复位。

表 1-2 RTC 闹钟唤醒配置示例

```
void RTC_Alarm_WakeUp( u32 SetAlarm )
{
    EXTI_InitTypeDef EXTI_InitStructure = {0};

    /* Enable PWR and BKP clocks */
    RCC_PB1PeriphClockCmd(RCC_PB1Periph_PWR | RCC_PB1Periph_BKP, ENABLE);
    PWR_BackupAccessCmd(ENABLE);
    RTC_ClearITPendingBit(RTC_IT_ALR);
    RTC_ClearITPendingBit(RTC_IT_SEC);
    /* Enable LSE oscillator */
    RCC_LSEConfig(RCC_LSE_ON);
    while(RCC_GetFlagStatus(RCC_FLAG_LSERDY) != SET);
    RCC_RTCCLKConfig(RCC_RTCCLKSource_LSE);
    /* Enable the RTC clock */
    RCC_RTCCLKCmd(ENABLE);
    RTC_WaitForLastTask();
    RTC_WaitForSynchro();
    /* Enable alarm interrupt */
    RTC_ITConfig( RTC_IT_ALR, ENABLE );
    RTC_ITConfig( RTC_IT_OW, ENABLE );
    RTC_WaitForLastTask();
    RTC_EnterConfigMode();
    /* Set the RTC prescaler value */
    RTC_SetPrescaler(32767);
    RTC_WaitForLastTask();
    RTC_SetCounter(0);
    RTC_WaitForLastTask();
    RTC_ExitConfigMode();

#if(Enter_MODE == Enter_WFI)
    /* Enable the RTC alarm interrupt */
     NVIC_SetPriority(RTCAlarm_IRQn, 1);
     NVIC_EnableIRQ(RTCAlarm_IRQn);
#endif
    /* Configure EXTI line */
    EXTI_InitStructure.EXTI_Line = EXTI_Line17;
#if(Enter_MODE == Enter_WFI)
    EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
#elif(Enter_MODE == Enter_WFE)
    EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Event;
#endif
    EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Rising_Falling;
    EXTI_InitStructure.EXTI_LineCmd = ENABLE;
    EXTI_Init(&EXTI_InitStructure);

    /* Set the RTC alarm value */
    RTC_SetAlarm(RTC_GetCounter()+10);
    RTC_WaitForLastTask();
    /* Enters STOP mode */
```

```
#if(Enter_MODE == Enter_WFI)
    /* Match with wake-up by interrupt */
    PWR_EnterSTOPMode(PWR_Regulator_LowPower,PWR_STOPEntry_WFI);
#elif(Enter_MODE == Enter_WFE)
    /* Match with wake-up by event */
    PWR_EnterSTOPMode(PWR_Regulator_LowPower,PWR_STOPEntry_WFE);
#endif
}
```

使能 PWR 和 BKP 时钟，配置 RTC 时钟源，待 LSE/LSI 时钟稳定后使能 RTC 时钟和闹钟中断，设置 RTC 时钟分频系数和闹钟值，执行 STOP 模式，10s 后产生闹钟中断唤醒系统，需要注意的是醒来 HSI 为默认系统时钟，为保证程序正常运行，建议重新进行系统时钟初始化。

表 1-3 RTC 闹钟唤醒中断示例

```
void RTCAlarm_IRQHandler(void)
{
    /* After waking up, HSI is the default system clock.To ensure the normal operation of
the system, it is recommended to perform clock initialization */
    SystemInit();
   if(EXTI_GetITStatus(EXTI_Line17)!=RESET)
   {
        /* Clears EXTI line and alarm clock flag */
        EXTI_ClearITPendingBit(EXTI_Line17);
        RTC_ClearITPendingBit(RTC_IT_ALR);
         RTC_WaitForLastTask();
   }
}
```

上述示例中使用了 RTC 闹钟中断函数 RTCAlarm_IRQHandler()，并没有用到 RTC 全局中断 RTC_IRQHandler()。如果两个中断函数同时使用的话，我们必须这样设置才不会有漏洞，RTCAlarm_IRQHandler()函数的优先级一定要高于 RTC_IRQHandler()。原因如下：

产生闹钟中断的前一瞬间，一定产生了秒中断，那么会先执行 RTC_IRQHandler()中断函数，在 RTC_IRQHandler()执行的过程中，闹钟中断标志又被挂起，由于 RTC_IRQHandler()是全局中断函数，必须清除所有的中断标志，程序才能退出该函数，假如 RTC_IRQHandler()和 RTCAlarm_IRQHandler()是同样的优先级，要想让程序退出 RTC_IRQHandler()函数，那么必须清除闹钟中断标志（如果不清除闹钟中断标志，程序会死在 RTC_IRQHandler()，这样问题又出现了，清除闹钟中断标志后，程序就不会进入 RTCAlarm_IRQHandler()，那么 RTCAlarm_IRQHandler()函数永远也不会被执行。

设置闹钟中断函数 RTCAlarm_IRQHandler()的优先级高于全局中断函数 RTC_IRQHandler()，在执行全局中断函数 RTC_IRQHandler()的时候，如果产生闹钟中断，那么中断嵌套去执行 RTCAlarm_IRQHandler()，执行完毕 RTCAlarm_IRQHandler()后，再去执行 RTC_IRQHandler()。

1.2.2 WKUP 引脚唤醒

CH32F/V10x_20x_V30x,CH32L103 等系列具有从低功耗模式唤醒 MCU 的专用硬件接口 WKUP 引脚(PA0)，使能后 WKUP 引脚强制配置为输入下拉状态，WKUP 上升沿用于把 MCU 从 STANDBY 状态下唤醒。建议在初始化时检测唤醒标志（PWR_GetFlagStatus(PWR_FLAG_WU)），区分是否为唤醒启动。

表 1-4 WKUP 引脚唤醒示例

```
void WakeUp_Config(void)
{
    RCC_PB1PeriphClockCmd(RCC_PB1Periph_PWR, ENABLE);

    if(PWR_GetFlagStatus(PWR_FLAG_WU) == SET)
    {
        printf("wake up.. \r\n");
    }
    else
    {
        printf("start standby... \r\n");
```

```
        /* Enable the WKUP Pin functionality */
        PWR_WakeUpPinCmd(ENABLE);
        /* Enters STANDBY mode */
        PWR_EnterSTANDBYMode();
    }
}
```

1.2.3 AWU 自动唤醒

对于 RISC-V 系列 MCU 中没有 RTC 功能的 MCU 如 CH32V00x、CH32X03x 系列，在低功耗模式下需要自动唤醒时，提供了 AWU 功能。AWU 功能支持中断或者事件模式唤醒 MCU，在事件模式唤醒 MCU 时代码更简单，flash 占用也更少，对于小容量 MCU 来说是比较好的选择。

AWU 模块是一个 6 位自加型计数器，当计数器计数到与写进去的值相等时，会从 STOP 模式或 STANDBY 模式下唤醒。

AWU 时间计算公式：

T = Windows Value * (Prescaler / AWU_CLK) (S)

以 CH32V00x 为例，设置 10240 分频，窗口值设置 25：内部低频 128kHz 时钟振荡器 LSI 作为自动唤醒计数时基，10240 分频之后计数一次时间为 1/12.5Hz，则唤醒时间间隔为 25/12.5=2S。CH32X035 计算方式与 CH32V00x 一致，但 AWU 模块时钟源不同，CH32X035 AWU 模块时钟源为内部高速时钟 HSI 的 47KHz 分频时钟。

AWU 事件唤醒的配置方法:首先配置 AWU 对应的外部中断中断线为事件模式，使能 PWR 时钟，使能 LSI，待 LSI 稳定后配置 AWU 的预分频值与窗口比较值，配置完成后，再使能 AWU 功能，并通过 WFE 命令让 MCU 进入低功耗模式，即可实现通过事件唤醒低功耗模式下 MCU 的目的。而且 AWU 仅需配置一次，不需要每次唤醒后重新配置。

AWU 中断唤醒的配置方法：首先配置 AWU 对应的外部中断线为中断模式（CH32V00x 因为 AWU 连接到外部中断线 9，同时外部中断只有 0-7，所以需要使能 AWU 中断，在中断服务函数中清除外部中断 9 的中断标志位，对于 CH32X035 则直接进对应的外部中断，并清除相关的中断标志位即可），使能相应中断，并配置中断优先级，使能 PWR 外设时钟，配置 AWU 预分频器，配置 AWU 窗口比较值，使能 AWU，执行 WFI 进入低功耗模式，此时就可以实现 AWU 的中断方式定时唤醒 MCU。

表 1-5 AWU 自动唤醒示例

```
void AWU_WakeUp(void)
{
    EXTI_InitTypeDef EXTI_InitStructure = {0};
    RCC_PB2PeriphClockCmd(RCC_PB2Periph_AFIO, ENABLE);

    EXTI_InitStructure.EXTI_Line = EXTI_Line9;
    EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Event;
    EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Falling;
    EXTI_InitStructure.EXTI_LineCmd = ENABLE;
    EXTI_Init(&EXTI_InitStructure);

    RCC_LSICmd(ENABLE);
    while(RCC_GetFlagStatus(RCC_FLAG_LSIRDY) == RESET);
    /* Set the Auto Wake up Prescaler */
    PWR_AWU_SetPrescaler(PWR_AWU_Prescaler_10240);
    /* Set the SWU window value */
    PWR_AWU_SetWindowValue(25);
    /* Enable the Auto WakeUp functionality */
    PWR_AutoWakeUpCmd(ENABLE);
    /* Enter STANDBY mode */
    PWR_EnterSTANDBYMode(PWR_STANDBYEntry_WFE);
    USART_Printf_Init(115200);
    printf("Auto wake up!!!!!\r\n");
}
```

1.2.4 LPTIM 唤醒

CH32L103系列LPTIM外设是专为低功耗应用场景设计的低功耗定时器，LPTIM是一个16位上行计数的定时器，具有多种可选的时钟源（可选内部时钟源：LSE、LSI、HSI或PB1时钟，外部时钟源：LPTIM输入上的外部时钟），使得LPTIM能在除待机模式外的所有电源模式下运行。LPTIM在没有内部时钟源的情况下也能运行，依此可以将LPTIM 当作“脉冲计数器”使用。除此之外，LPTIM能够将系统从低功耗模式唤醒，所以LPTIM很适合以极低的功耗实现“超时功能”。

低功耗模式对LPTIM的影响:

SLEEP模式：无影响，LPTIM中断会导致设备退出睡眠模式。

STOP模式：LPTIM外围设备在由LSE或LSI计时时处于活动状态，LPTIM中断导致设备退出停止模式。

STANDBY模式：LPTIM外围设备已断电，必须在退出待机模式后重新初始化。

如果使用事件唤醒，需要使能外部中断通道 21 的事件请求信号；如果使用中断唤醒，则需要使能外部中断通道 21 的中断请求信号，并且使能 LPTIM 外部中断，编写中断服务函数。

表 1-6 LPTIM 唤醒示例

```
void LPTIM_Config(u16 arr)
{
    NVIC_InitTypeDef NVIC_InitStructure = {0};
    EXTI_InitTypeDef EXTI_InitStructure = {0};
    LPTIM_TimeBaseInitTypeDef   LPTIM_TimeBaseInitStruct = {0};
    RCC_PB2PeriphClockCmd(RCC_PB2Periph_GPIOB|RCC_PB2Periph_AFIO, ENABLE);
    RCC_PB1PeriphClockCmd(RCC_PB1Periph_PWR|RCC_PB1Periph_LPTIM, ENABLE);
    /* Enbale LPTIM EXTI-line */
    EXTI_InitStructure.EXTI_Line = EXTI_Line21;
    EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
    EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Rising;
    EXTI_InitStructure.EXTI_LineCmd = ENABLE;
    EXTI_Init(&EXTI_InitStructure);
    /* Enbale LPTIM_WakeUp interrupt */
    NVIC_InitStructure.NVIC_IRQChannel = LPTIMWakeUp_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 1;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);
    /* Enable LPTIM */
    LPTIM_Cmd(ENABLE);

/* INWAKEUP Mode:LSI provide clock for LPTIM in low power mode */
#if(WKMODE==INWAKEUP)
    RCC_LSICmd(ENABLE);
    while(RCC_GetFlagStatus(RCC_FLAG_LSIRDY)!=SET);
#endif
/* EXWAKEUP Mode:PB12 Provide clock for LPTIM in low power mode */
#if(WKMODE==EXWAKEUP)
    LPTIM_TimeBaseInitStruct.LPTIM_ClockSource = LPTIM_ClockSource_Ex;
    LPTIM_TimeBaseInitStruct.LPTIM_CountSource = LPTIM_CountSource_External;
    LPTIM_TimeBaseInitStruct.LPTIM_ClockPrescaler = LPTIM_TClockPrescaler_DIV8;
    LPTIM_TimeBaseInitStruct.LPTIM_InClockSource = LPTIM_InClockSource_PCLK1;
#elif(WKMODE==INWAKEUP)
    LPTIM_TimeBaseInitStruct.LPTIM_ClockSource = LPTIM_ClockSource_In;
    LPTIM_TimeBaseInitStruct.LPTIM_CountSource = LPTIM_CountSource_Internal;
    LPTIM_TimeBaseInitStruct.LPTIM_ClockPrescaler = LPTIM_TClockPrescaler_DIV128;
    LPTIM_TimeBaseInitStruct.LPTIM_InClockSource = LPTIM_InClockSource_LSI;
#endif
    LPTIM_TimeBaseInitStruct.LPTIM_ClockPolarity = LPTIM_ClockPolarity_Falling;
    LPTIM_TimeBaseInitStruct.LPTIM_ClockSampleTime = LPTIM_ClockSampleTime_0T;
    LPTIM_TimeBaseInitStruct.LPTIM_TriggerSampleTime = LPTIM_TriggerSampleTime_0T;
    LPTIM_TimeBaseInitStruct.LPTIM_ExTriggerPolarity = LPTIM_ExTriggerPolarity_Disable;
    LPTIM_TimeBaseInitStruct.LPTIM_TimeOut = ENABLE;
    LPTIM_TimeBaseInitStruct.LPTIM_OutputPolarity = LPTIM_OutputPolarity_High;
    LPTIM_TimeBaseInitStruct.LPTIM_UpdateMode = LPTIM_UpdateMode0;
```

```
    LPTIM_TimeBaseInitStruct.LPTIM_Encoder = DISABLE;
    LPTIM_TimeBaseInitStruct.LPTIM_ForceOutHigh = DISABLE;
    LPTIM_TimeBaseInitStruct.LPTIM_SingleMode = DISABLE;
    LPTIM_TimeBaseInitStruct.LPTIM_ContinuousMode = ENABLE;
    LPTIM_TimeBaseInitStruct.LPTIM_PWMOut = DISABLE;
    LPTIM_TimeBaseInitStruct.LPTIM_CounterDirIndicat = DISABLE;
    LPTIM_TimeBaseInitStruct.LPTIM_Pulse = 0;
    LPTIM_TimeBaseInitStruct.LPTIM_Period = arr;
    LPTIM_TimeBaseInit( & LPTIM_TimeBaseInitStruct);
    LPTIM_ITConfig(LPTIM_IT_ARRM, ENABLE);
    /* Enter STOP mode */
    PWR_EnterSTOPMode(PWR_Regulator_LowPower, PWR_STOPEntry_WFI);
}
```

示例中，在对外部中断线和 LPTIM 的初始化结束后，设备进入 STOP 模式，使用 LPTIM-WKUP 中断唤醒设备，醒来 HSI 为默认系统时钟，为保证程序正常运行，建议重新进行系统时钟初始化。

```
void LPTIMWakeUp_IRQHandler(void) __attribute__((interrupt("WCH-Interrupt-fast")));
void LPTIMWakeUp_IRQHandler(void)
{
    if(EXTI_GetITStatus(EXTI_Line21)!=RESET)
    {
        EXTI_ClearITPendingBit(EXTI_Line21);
        SystemInit();
    }
}
```

1.2.5 PVD 唤醒

可编程电压监视器 PVD，主要被用于监控系统主电源的变化，与电源控制寄存器 PWR_CTLR 的 PLS[2:0]所设置的门槛电压相比较，配合外部中断寄存器（EXTI）设置，可产生相关事件中断，从 SLEEP 或 STOP 模式下唤醒。

配置方法如下:使能 PWR 时钟，然后设置电压监视阈值，随后使能 PVDE 开启电源电压监视功能。PVD 功能内部连接 EXTI 模块的第 16 线的上升/下降边沿触发设置，配置 EXTI 相关寄存器，当 VDD 下降到 PVD 阀值以下或上升到 PVD 阀值之上时就会产生 PVD 事件或中断。SANDBY 模式不支持 PVD 唤醒。

表 1-7 PVD 唤醒示例

```
void PVD_Config(void)
{
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_PWR, ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO, ENABLE);

    EXTI_InitTypeDef EXIT_InitStructure = {0};
    NVIC_InitTypeDef NVIC_InitStructure = {0};
    /* Enable PVD EXTI-line */
    EXIT_InitStructure.EXTI_Line = EXTI_Line16;
    EXIT_InitStructure.EXTI_LineCmd = ENABLE;
#if(Enter_MODE == Enter_WFI)
    EXIT_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
#elif(Enter_MODE == Enter_WFE)
    EXIT_InitStructure.EXTI_Mode = EXTI_Mode_Event;
#endif
    EXIT_InitStructure.EXTI_Trigger = EXTI_Trigger_Rising_Falling;
    EXTI_Init(&EXIT_InitStructure);
#if(Enter_MODE == Enter_WFI)
    NVIC_InitStructure.NVIC_IRQChannel = PVD_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);
#endif
    /* Configures the voltage threshold detected by the PVD */
    PWR_PVDLevelConfig(PWR_PVDLevel_MODE7);
    /* Enable the Power Voltage Detector(PVD) */
```

```
    PWR_PVDCmd(ENABLE);
    /* Enters STOP mode */
    PWR_EnterSTOPMode(PWR_Regulator_LowPower, PWR_STOPEntry_WFE);
    SystemInit();
    printf("Wake up!!\r\n");
}
```

第 2 章 低功耗注意事项

2.1 如何优化芯片睡眠电流

2.1.1 MCU 引脚配置

● 在进入低功耗模式前，需要将所有 I/O 配置成为上拉/下拉输入或模拟输入，防止芯片 I/O 浮空产生漏电流。注意芯片引脚是否支持上拉/下拉输入或模拟输入功能（否则配置不生效，默认浮空态），根据具体情况选择。

● 对于芯片小封装型号，相较最大封装，未封装出的引脚，建议配置为上拉/下拉输入或模拟输入，否则可能影响电流指标。

● 释放 SWD 调试接口作为 GPIO 功能，并配置为上拉/下拉输入或模拟输入(唤醒后恢复 SWD 功能) GPIO_PinRemapConfig(GPIO_Remap_SWJ_Disable, ENABLE)。

2.1.2 关闭外设降低主频

● 在 SLEEP 模式下内核停止，但所有外设（包含内核私有外设）仍在运行，如果系统时钟频率越高，使能的外设越多，功耗就越大，建议睡眠前将未使用的外设禁用，关闭 PLL 切换低速时钟，实现节能。此外部分产品可配置 HSI 进入低功耗模式，如 CH32L103 系列 HSI 使能低功耗模式后，输出频率从 8MHz 降为 1MHz（使用 UART 时 HSI 为 1MHz，波特率支持低于 62.5Kbps）。

● 避免外设关闭不充分，STOP 模式下或某些芯片的 STANDBY 模式，电压调节器未关闭，例如睡眠前使能了 ADC，睡眠后会自动关闭 ADC 外设时钟，但 ADC 内部电路仍会存在部分电流消耗，在睡眠前需失能外设。

2.1.3 开启电压调节器与 RAM 节能模式

以 CH32F/V20x_V30x 为例:

在 STOP 模式下，可选 LPDS 位，LPDS=0，电压调节器工作在正常模式；LPDS=1，电压调节器工作在低功耗模式。在低功耗模式下，可以通过配置 PWR_CTLR 寄存器的 RAMLV=1，使能 RAM 低电压模式，降低功耗。

在 STANDBY 模式下，当正常供电时，通过配置 PWR_CTLR 寄存器的 R2KSTY=1 控制 2K 字节 RAM 不掉电，R30KSTY=1 控制 30K 字节 RAM 不掉电；当使用 VBAT 供电时，通过配 PWR_CTLR 寄存器的 R2KVBAT=1 控制 2K 字节 RAM 不掉电，R32K_VBATEN=1 控制 30K 字节 RAM 不掉电。在该基础之上，可以通过配置 PWR_CTLR 寄存器的 RAMLV=1，使能 RAM 低电压模式，降低功耗。

2.1.4 代码从 SRAM 中运行

以 CH32L103 系列为例，SLEEP 或 STOP 模式下可将数据处理代码放 SRAM 中运行，同时配置 flash 进入低功耗，调节 LDOTRIM 降低数字内核电压，以降低功耗。

2.1.5 检查外围电路

● 注意外部电源转换器件漏电流：某些驱动、电源转换芯片如果是由软件使能，进入低功耗模式之后建议关闭，因为本身会有工作电流消耗；如果是由硬件使能，则需要查看芯片手册看静态工作电流是多少，如果比较高，只能修改硬件电路或者用低功耗芯片代替。

● 外围是否有闭合回路：比如电压采集分压电阻未断开，虽然 ADC 已经关闭，但是这个分压电路还是有电流消耗的，大小一般取决于电阻和电源电压；传感器、通信模块、LED 指示灯等外围器件

在进入低功耗前应切断供电。

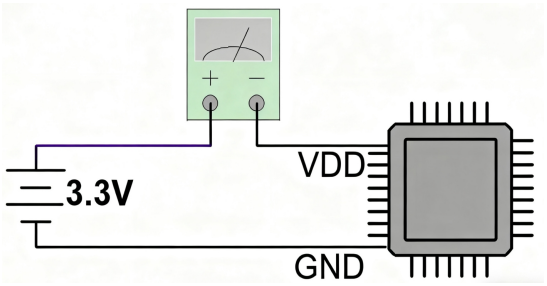
● GPIO 状态是否冲突：根据设备原理图查看 I/O 引脚外部的连接电路，当引脚通过外围电路电阻接地被拉低时，如果引脚设置为上拉输入，则在芯片内部的上拉电阻和外围的下拉电阻构成回路，电流消耗取决于这两个电阻；当引脚通过外围电路电阻接电源被拉高时，如果引脚设置为下拉输入，则在芯片内部的下拉电阻和外围的上拉电阻构成回路，电流消耗也取决于这两个电阻。

2.1.6 减小电流测量误差

● 测量功耗电流时，断开外围模块供电（如传感器、通信芯片，LED、USB 转串口芯片），避免外部器件漏电影响，仅保留 MCU 最小系统。

● 实测电流时要完全断开 SWD 调试器连接，仅保留供电线，万用表测量 uA 级静态电流时档位需切换至 uA 档，避免量程过大丢失精度。

图 2-1 电流测量示意图



2.2 睡眠唤醒注意事项

2.2.1 正确睡眠唤醒

● 睡眠前使能 PWR 外设时钟。不同的低功耗模式，唤醒资源是不一样的，通过查阅应用手册来保证低功耗的模式和唤醒源是匹配关系。

● 避免唤醒源持续触发，如持续中断会阻止睡眠，中断标志要清除彻底。

● 唤醒后时钟重建，退出停止模式后 HSI 自动作为系统时钟，若原用 HSE/PLL 需手动切换（未重配时钟导致外设失效），唤醒后调用时钟配置函数。

● 检查低功耗模式、进入睡眠方式（WFI/WFE）和唤醒方式（事件/中断）是否匹配，EXTI 线路是否正确映射，并确认触发边沿（上升沿/下降沿）与实际信号一致，信号满足沿要求。

2.2.2 STANDBY 模式引脚电平维持方法

通常情况下，芯片进入 STANDBY 模式，芯片引脚并不会保持原来的状态，I/O 默认为模拟输入。若希望芯片进入 STANDBY 模式且引脚电平保持，可通过如下方法实现：

第一步：配置引脚为上拉/下拉输入

第二步：打开 AFIO 时钟

第三步：将需要保持电平的引脚配置为外部中断输入引脚，操作 AFIO_EXTICR[x]寄存器（以 PA3 为例，AFIO->EXTICR[1]=0）

第四步：使能外部中断通道的事件请求，操作 EXT_EVENR 寄存器（以 PA3 为例，EXTI->EVENR|=(1<<3)）

注意：仅支持上/下拉输入的电平保持，不支持输出保持。该功能因为 AFIO_EXTICR[x]，例 PA3,PB3,操作的是 AFIO_EXTICR[x]寄存器的相同位，所以不能同时保持相同 EXTI 通道引脚的电平。

2.2.3 WFE 事件唤醒源获取方法

WFE 方式唤醒无需中断控制器介入，若使用 WFE 方式唤醒设备，且需要获取 WFE 唤醒源时，可通过如下方法：

第一步：配置外部或内部的 EXTI 线为事件模式。

第二步：使能相应 EXTI 线的中断请求信号（EXTI_INTENR）。

第三步：执行 WFE，唤醒后查询中断标志位寄存器（EXTI_INTFR），判断事件唤醒源。

2.2.4 STANDBY 模式 RAM 数据保持方法

通常情况STANDBY模式下，电压调节器关闭，除唤醒电路和后备域电路之外的电路将断电，默认所有RAM数据会丢失，在指定的唤醒条件下退出后，微控制器将被复位。

以CH32F/V20x_V30x为例，在待机模式下，当正常供电时，可以通过配置PWR_CTLR寄存器的R2KSTY=1控制2K字节RAM不掉电，R30KSTY=1控制30K字节RAM不掉电；当使用VBAT供电时，通过配置PWR_CTLR寄存器的R2KVBAT=1控制2K 字节RAM不掉电，R32K_VBATEN=1控制30K字节RAM不掉电。在该基础之上，可以通过配置PWR_CTLR 寄存器的RAMLV=1，使能RAM低电压模式，以降低功耗。

历史版本

更新内容

日期	版本	变更内容
2025/9/10	V1.0	初版发行

声明

任何未经允许擅自更改本产品手册中的内容与南京沁恒微电子股份有限公司无关。

南京沁恒微电子股份有限公司所提供的说明文档只作为相关产品的使用参考，不包含任何对特殊使用目的的担保。南京沁恒微电子股份有限公司保留更改和升级本产品手册以及手册中涉及的产品或软件的权利。

参考手册中可能包含少量由于疏忽造成的错误。已发现的会定期勘误，并在再版中更新和避免出现此类错误。