低功耗操作实验

【实验目的】

测量 STM32 在各种状态下的功耗，包括在不同时钟频率下（32M、8M、1M、100K、10K）、不同振荡器（内部、外部）、不同模式（活动、睡眠、停机、待机）的电流消耗，弄清楚在不同低功耗模式下的唤醒方式。

【实验要求】

1．编程要求：利用 C 语言，调用 STM32 的库函数，完成对各种工作模式的操作。
2．实现功能：测试不同状态下功耗。
3．实验现象：用万用表测试电流消耗。

【硬件电路】

测试时电路连接如图 3-1 所示。

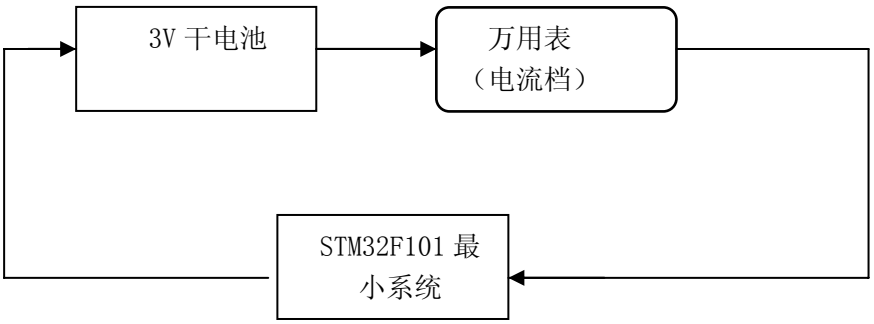

图 1 测试电路连接图

【实验原理】

在系统或电源复位以后，微控制器处于运行状态。当 CPU 不需继续运行时，可以利用多种低功耗模式来节省功耗，例如等待某个外部事件时。用户需要根据最低电源消耗、最快速启动时间和可用的唤醒源等条件，选定一个最佳的低功耗模式。

STM32F10xxx 有三种低功耗模式，每种模式的进入退出条件如图 2 所示。

- 睡眠模式(Cortex-M3 内核停止，所有外设包括 Cortex-M3 核心的外设，如 NVIC、系统时钟(SysTick)等仍在运行)
- 停止模式(所有的时钟都已停止)
- 待机模式(1.8V 电源关闭)

此外，在运行模式下，可以通过以下方式中的一种降低功耗：

- 降低系统时钟
- 关闭 APB 和 AHB 总线上未被使用的外设时钟。

模式	进入	唤醒	对1.8V区域时钟的影响	对V_{DD}区域时钟的影响	电压调节器
睡眠(SLEEP-NOW或SLEEP-ON-EXIT)	WFI	任一中断	CPU时钟关，对其他时钟和ADC时钟无影响	无	开
	WFE	唤醒事件			
停机	PDDS和LPDS位+SLEEPDEEP位+WFI或WFE	任一外部中断(在外部中断寄存器中设置)	关闭所有1.8V区域的时钟	HSI 和HSE的振荡器关闭	开启或处于低功耗模式(依据电源控制寄存器(PWR_CR)的设定)
待机	PDDS位+SLEEPDEEP位+WFI或WFE	WKUP引脚的上升沿、RTC闹钟事件、NRST引脚上的外部复位、IWDG复位			关

图 2　各种模式的进入退出条件

进入睡眠模式

通过执行 WFI 或 WFE 指令进入睡眠状态。根据 Cortex-M3 系统控制寄存器中的 SLEEPONEXIT 位的值，有两种选项可用于选择睡眠模式进入机制：

● SLEEP-NOW：如果 SLEEPONEXIT 位被清除，当 WRI 或 WFE 被执行时，微控制器立即进入睡眠模式。

● SLEEP-ON-EXIT：如果 SLEEPONEXIT 位被置位，系统从最低优先级的中断处理程序中退出时，微控制器就立即进入睡眠模式。 在睡眠模式下，所有的 I/O 引脚都保持它们在运行模式时的状态。

退出睡眠模式

如果执行 WFI 指令进入睡眠模式，任意一个被嵌套向量中断控制器响应的外设中断都能将系统从睡眠模式唤醒。如果执行 WFE 指令进入睡眠模式，则一旦发生唤醒事件时，微处理器都将从睡眠模式退出。唤醒事件可以通过下述方式产生：

● 在外设控制寄存器中使能一个中断，而不是在 NVIC(嵌套向量中断控制器)中使能，并且在 Cortex-M3 系统控制寄存器中使能 SEVONPEND 位。当 MCU 从 WFE 中唤醒后，外设的中断挂起位和外设的 NVIC 中断通道挂起位(在 NVIC 中断清除挂起寄存器中)必须被清除。

● 配置一个外部或内部的 EXIT 线为事件模式。当 MCU 从 WFE 中唤醒后，因为与事件线对应的挂起位未被设置，不必清除外设的中断挂起位或外设的 NVIC 中断通道挂起位。 该模式唤醒所需的时间最短，因为没有时间损失在中断的进入或退出上。

停止模式

停止模式是在 Cortex-M3 的深睡眠模式基础上结合了外设的时钟控制机制，在停止模式下电压调节器可运行在正常或低功耗模式。此时在 1.8V 供电区域的的所有时钟都被停止，PLL、HSI 和 HSE RC 振荡器的功能被禁止，SRAM 和寄存器内容被保留下来。 在停止模式下，所有的 I/O 引脚都保持它们在运行模式时的状态。关于如何进入停止模式，详见图 3。

电源控制寄存器(PWR_CR) 在停止模式下，通过设置 的 LPDS 位使内部调节器进入低功耗模式，能够降低更多的功耗。 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 APB 的访问，直到对 APB 访问

完成，系统才进入停止模式。 可以通过对独立的控制位进行编程，可选择以下功能：

● 独立看门狗(IWDG)：可通过写入看门狗的键寄存器或硬件选择来启动 IWDG。一旦启动了独立看门狗，除了系统复位，它不能再被停止。

● 实时时钟(RTC)：通过备份域控制寄存器 (RCC_BDCR)的 RTCEN 位来设置。

● 内部 RC 振荡器(LSI RC)：通过控制/状态寄存器 (RCC_CSR)的 LSION 位来设置。

● 外部 32.768kHz 振荡器(LSE)：通过备份域控制寄存器 (RCC_BDCR)的 LSEON 位设置。

在停止模式下，如果在进入该模式前 ADC 和 DAC 没有被关闭，那么这些外设仍然消耗电流。通过设置寄存器 ADC_CR2 的 ADON 位和寄存器 DAC_CR 的 ENx 位为 0 可关闭这 2 个外设。

当一个中断或唤醒事件导致退出停止模式时，HSI RC 振荡器被选为系统时钟。 当电压调节器处于低功耗模式下，当系统从停止模式退出时，将会有一段额外的启动延时。如果在停止模式期间保持内部调节器开启，则退出启动时间会缩短，但相应的功耗会增加。

停止模式	说明
进入	在以下条件下执行WFI(等待中断)或WFE(等待事件)指令： – 设置Cortex-M3系统控制寄存器中的SLEEPDEEP位 – 清除电源控制寄存器(PWR_CR)中的PDDS位 – 通过设置PWR_CR中LPDS位选择电压调节器的模式 注：为了进入停止模式，所有的外部中断的请求位(挂起寄存器(EXTI_PR))和RTC的闹钟标志都必须被清除，否则停止模式的进入流程将会被跳过，程序继续运行。
退出	如果执行WFI进入停止模式： 设置任一外部中断线为中断模式(在NVIC中必须使能相应的外部中断向量)。参见中断向量表(表54)。 如果执行WFE进入停止模式： 设置任一外部中断线为事件模式。参见唤醒事件管理(第9.2.3节)。
唤醒延时	HSI RC唤醒时间 + 电压调节器从低功耗唤醒的时间。

图 3 停机模式的进入退出条件

待机模式

待机模式可实现系统的最低功耗。该模式是在 Cortex-M3 深睡眠模式时关闭电压调节器。整个 1.8V 供电区域被断电。PLL、HSI 和 HSE 振荡器也被断电。SRAM 和寄存器内容丢失。只有备份的寄存器和待机电路维持供电。

关于如何进入待机模式，详见图 4。 可以通过设置独立的控制位，选择以下待机模式的功能：

● 独立看门狗(IWDG)：可通过写入看门狗的键寄存器或硬件选择来启动 IWDG。一旦启动了独立看门狗，除了系统复位，它不能再被停止。

● 实时时钟(RTC)：通过备用区域控制寄存器(RCC_BDCR)的 RTCEN 位来设置。

● 内部 RC 振荡器(LSI RC)：通过控制/状态寄存器(RCC_CSR)的 LSION 位来设置。

● 外部 32.768kHz 振荡器(LSE)：通过备用区域控制寄存器(RCC_BDCR)的 LSEON 位设置。

当一个外部复位(NRST 引脚)、IWDG 复位、WKUP 引脚上的上升沿或 RTC 闹钟事件的上升沿发生时，微控制器从待机模式退出。从待机唤醒后，除了 ： 电源控制/状态寄存器(PWR_CSR)，所有寄存器被复位。

从待机模式唤醒后的代码执行等同于复位后的执行(采样启动模式引脚、读取复位向量等)。电源控制/状态寄存器(PWR_CSR)将会指示内核由待机状态退出。待机模式下的输入/输出端口状态 在待机模式下，所有的 I/O 引脚处于高阻态，除了以下的引脚：

- 复位引脚(始终有效)
- 当被设置为防侵入或校准输出时的 TAMPER 引脚
- 被使能的唤醒引脚

待机模式	说明
进入	在以下条件下执行WFI(等待中断)或WFE(等待事件)指令： - 设置Cortex™-M3系统控制寄存器中的SLEEPDEEP位 - 设置电源控制寄存器(PWR_CR)中的PDDS位 - 清除电源控制/状态寄存器(PWR_CSR)中的WUF位
退出	WKUP引脚的上升沿、RTC闹钟事件的上升沿、NRST引脚上外部复位、IWDG复位。
唤醒延时	复位阶段时电压调节器的启动。

图 4　待机模式的进入退出条件

【实验步骤】

1. 学习 STM32 电源控制相关知识，熟悉所调用的库函数。
2. 搭建测试电路。
3. 编写程序，测试电流消耗，对照数据手册，对没有达到的指标进行分析。

【程序代码】

```
#include "stm32f10x.h"
#include "stm32_eval.h"

GPIO_InitTypeDef GPIO_InitStructure;
EXTI_InitTypeDef EXTI_InitStructure;

void EXTI_Config_owl(void);
void RCC_Config_HSI_32M_owl(void);
void RCC_Config_HSE_32M_owl(void);
void TIM2_Base_Config_owl(void);
void TIM2_PWM_Config_owl(void);
void USART1_Config_owl(void);//配置 USART1 口
void RTC_Config_owl(void);//RTC 配置

int main(void)
{
  RCC_Config_HSI_32M_owl();//配置内部时钟到 32M
  // RCC_Config_HSE_32M_owl();//选择 HSE 倍频做为时钟
```

```
  EXTI_Config_owl();                                  //对接收中断的初始化
  USART1_Config_owl();//配置 USART1 口
  TIM2_Base_Config_owl();
  RTC_Config_owl();                                   //对 RTC 配置

  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA ,ENABLE);//时钟使能
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB ,ENABLE);//时钟使能

  GPIO_InitStructure.GPIO_Pin=GPIO_Pin_All;
  GPIO_InitStructure.GPIO_Mode= GPIO_Mode_Out_OD   ;
  GPIO_InitStructure.GPIO_Speed=GPIO_Speed_50MHz;

  GPIO_Init(GPIOA, &GPIO_InitStructure);
  GPIO_Init(GPIOB, &GPIO_InitStructure);

  GPIO_InitStructure.GPIO_Pin=GPIO_Pin_1;
  GPIO_InitStructure.GPIO_Mode=GPIO_Mode_IPU;
  GPIO_InitStructure.GPIO_Speed=GPIO_Speed_50MHz;

  GPIO_Init(GPIOA, &GPIO_InitStructure);

  PWR_EnterSTOPMode(PWR_Regulator_ON, PWR_STOPEntry_WFE); //进入停机模式
    // PWR_EnterSTANDBYMode(); //进入待机模式

 while (1)
  {
  }
}
//////////////////////////////////////////////////////////////////////////////

void EXTI_Config_owl(void)
{
  NVIC_InitTypeDef NVIC_InitStructure;//NVIC 配置结构体

  RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO,ENABLE);//中断时钟使能

  //中断向量控制 reg 初始化
  NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1);//对优先级的配置，还需要简单了解

  //使能 1 中断
  NVIC_InitStructure.NVIC_IRQChannel =    EXTI1_IRQn;
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
```

```
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);

    //选择做中断的引脚 A1
    GPIO_EXTILineConfig(GPIO_PortSourceGPIOA, GPIO_PinSource1);

    //外中断初始化设置
    EXTI_InitStructure.EXTI_Line = EXTI_Line1;//第 1 线
    EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt; //中断模式
    EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Falling;//下降沿触发
    EXTI_InitStructure.EXTI_LineCmd = ENABLE;//使能
    EXTI_Init(&EXTI_InitStructure);//操作

    EXTI_GenerateSWInterrupt(EXTI_Line1);//类似于允许中断

    EXTI_ClearITPendingBit(EXTI_Line1);//退出之前一定要记得清标志位

}
//配置时钟使用 HSI，32M
void RCC_Config_HSI_32M_owl(void)
{
    RCC_DeInit();//复位到初始值
    RCC_HSICmd(ENABLE);//使能 HSI
    RCC_AdjustHSICalibrationValue(0x10);//校正值
    FLASH_PrefetchBufferCmd(FLASH_PrefetchBuffer_Enable);//使能预取指缓存区
    FLASH_SetLatency(FLASH_Latency_1);//设置 flash 操作等待
    RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK);//选择 PLL 作为系统时钟
    RCC_HCLKConfig(RCC_SYSCLK_Div1);//主时钟不分频
    RCC_PCLK1Config(RCC_HCLK_Div1);//APB1 由 AHB 不分频得到
    RCC_PCLK2Config(RCC_HCLK_Div1);//APB2 由 AHB 不分频得到
    RCC_ITConfig(RCC_IT_HSIRDY , ENABLE );//HSI 中断允许
    //RCC_USBCLKConfig(RCC_USBCLKSource_PLLCLK_Div1);//USB 时钟为系统不分频
    RCC_ADCCLKConfig(RCC_PCLK2_Div8);//AD 时钟为 AOB2 的 2 分频
    RCC_AHBPeriphClockCmd(RCC_AHBPeriph_SRAM | RCC_AHBPeriph_FLITF,
ENABLE);//使能 SRAM、FLITF 时钟
    RCC_PLLConfig(RCC_PLLSource_HSI_Div2, RCC_PLLMul_8);//PLL 时钟源及倍频因子
设定
    RCC_PLLCmd(ENABLE);//使能 PLL

    while (RCC_GetSYSCLKSource() != 0x08)//检查时钟是否配置正确
    {
    }
}
```

```
//配置时钟使用 HSE，32M
void RCC_Config_HSE_32M_owl(void)
{
  ErrorStatus HSEStartUpStatus;

  RCC_DeInit();//复位到初始值
  RCC_HSEConfig(RCC_HSE_ON);;//使能 HSE
  HSEStartUpStatus = RCC_WaitForHSEStartUp();//等待 HSE 准备好

  if (HSEStartUpStatus == SUCCESS)
  {
    FLASH_PrefetchBufferCmd(FLASH_PrefetchBuffer_Enable);//使能预取指缓存区
    FLASH_SetLatency(FLASH_Latency_1);//设置 flash 操作等待
    RCC_HCLKConfig(RCC_SYSCLK_Div1);//主时钟不分频
    RCC_PCLK1Config(RCC_HCLK_Div2);//APB1 由 AHB2 分频得到
    RCC_PCLK2Config(RCC_HCLK_Div1);//APB2 由 AHB 不分频得到
    RCC_PLLConfig(RCC_PLLSource_HSE_Div1, RCC_PLLMul_4);//PLL 时钟源及倍频因
子设定
    RCC_PLLCmd(ENABLE);//使能 PLL

    while (RCC_GetFlagStatus(RCC_FLAG_PLLRDY) == RESET)//等待 PLL 启动
    {
    }

    RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK);//选择 PLL 作为系统时钟
    //RCC_USBCLKConfig(RCC_USBCLKSource_PLLCLK_Div1);//USB 时钟为系统不分频
    RCC_ADCCLKConfig(RCC_PCLK2_Div2);//AD 时钟为 AOB2 的 2 分频
    RCC_AHBPeriphClockCmd(RCC_AHBPeriph_SRAM | RCC_AHBPeriph_FLITF,
ENABLE);//使能 SRAM、FLITF 时钟
    while(RCC_GetSYSCLKSource() != 0x08)
    {
    }
  }
}
//////////////////////////////////////////////////////////////////////////////
// 程序名 : void TIM2_Base_Config_owl(void)
// 作用 :  通用定时器 2 配置函数，配置定时器 2 工作在普通计时模式
// 输入参数:无
// 输出参数:无
// 说明:  配置定时器 2 工作在普通计时模式，定时器 UP 模式，使能 up 中断，
//
//////////////////////////////////////////////////////////////////////////////
void TIM2_Base_Config_owl(void)
{
```

```
   NVIC_InitTypeDef NVIC_InitStructure;

   TIM_TimeBaseInitTypeDef    TIM_TimeBaseStructure;


   RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO,ENABLE);//中断时钟使能 为什么要使能这个时钟？

   NVIC_SetVectorTable(NVIC_VectTab_FLASH, 0x0);//中断向量在 flash 中

   //中断控制器中选中 TIM2
   NVIC_InitStructure.NVIC_IRQChannel = TIM2_IRQn;
   NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
   NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
   NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;

   NVIC_Init(&NVIC_InitStructure);

   RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2,ENABLE);

   //定时器基本配置
   TIM_TimeBaseStructure.TIM_Period = 0x4000;//重载计数值
   TIM_TimeBaseStructure.TIM_Prescaler  =  TIM_PSCReloadMode_Update;//估计这个只是对分频因子何时装载进行设置
   //TIM_TimeBaseStructure.TIM_Prescaler = 0x0002;
   TIM_TimeBaseStructure.TIM_ClockDivision = TIM_CKD_DIV1 ;//还不清楚作用
   TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;//向上模式

   TIM_TimeBaseInit(TIM2, &TIM_TimeBaseStructure);//操作

   TIM_PrescalerConfig(TIM2,  0x0000,  TIM_PSCReloadMode_Immediate);//定时器使用的时钟的分频设置
   TIM_ITConfig(TIM2,    TIM_IT_Update , ENABLE);//对定时器中哪个中断源进行设置。
   TIM_Cmd(TIM2, ENABLE);//定时器 2 允许工作
}
////////////////////////////////////////////////////////////////////////////////
// 程序名 : void USART1_Config_owl(void)
// 作用 :      配置 USART1 口工作在普通异步传输模式
// 输入参数:无
// 输出参数:无
// 说明:
//
//
////////////////////////////////////////////////////////////////////////////////
```

```
void USART1_Config_owl(void)
{
   USART_InitTypeDef USART_InitStructure;//配置 USART 数据结构
   NVIC_InitTypeDef NVIC_InitStructure;//NVIC 配置结构体
    GPIO_InitTypeDef GPIO_InitStructure;//GPIO 配置结构体

   //GPIO 配置
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA ,ENABLE);//时钟使能
   //TX 配置
   GPIO_InitStructure.GPIO_Pin=GPIO_Pin_9;
   GPIO_InitStructure.GPIO_Mode=GPIO_Mode_AF_PP;
   GPIO_InitStructure.GPIO_Speed=GPIO_Speed_50MHz;
   GPIO_Init(GPIOA, &GPIO_InitStructure);
   //RX 配置
   GPIO_InitStructure.GPIO_Pin=GPIO_Pin_10;
   GPIO_InitStructure.GPIO_Mode=GPIO_Mode_IN_FLOATING;
   GPIO_InitStructure.GPIO_Speed=GPIO_Speed_50MHz;
   GPIO_Init(GPIOA, &GPIO_InitStructure);

   RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1 ,ENABLE);//时钟使能

   USART_InitStructure.USART_BaudRate=57600; //波特率
   USART_InitStructure.USART_WordLength=USART_WordLength_8b;//8 位数据
   USART_InitStructure.USART_StopBits=USART_StopBits_1;//1 个停止位
   USART_InitStructure.USART_Parity=USART_Parity_No;//无校验
   USART_InitStructure.USART_Mode=USART_Mode_Rx | USART_Mode_Tx; //模式

USART_InitStructure.USART_HardwareFlowControl=USART_HardwareFlowControl_None;//
硬件流控制失能
   USART_Init(USART1, &USART_InitStructure);//操作

   //对 USART1 中断的配置，这是可选项，很多时候可以不使用中断
   RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO,ENABLE);//中断时钟使能 为什么要
使能这个时钟？
   //中断向量控制 reg 初始化
   NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1);//对优先级的配置，还需要简单了解
   //使能对 USART1 中断
   NVIC_InitStructure.NVIC_IRQChannel = USART1_IRQn;
   NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
   NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
   NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
   NVIC_Init(&NVIC_InitStructure);

   ////USART_ITConfig(USART1,  USART_IT_TC  | USART_IT_RXNE, ENABLE);//使能发
```

```
送完成中断和接收中断
   USART_ITConfig(USART1, USART_IT_RXNE, ENABLE);//使能接收中断
  // USART_ITConfig(USART1, USART_IT_TC, ENABLE);//使能发送完成中断

   USART_Cmd(USART1, ENABLE);
}

//////////////////////////////////////////////////////////////////////////////
// 程序名 : void RTC_Config_owl(void)
// 作用 :      配置 RTC 和 BKP，RTC 开启秒中断。
// 输入参数:无
// 输出参数:无
// 说明:
//
//
//////////////////////////////////////////////////////////////////////////////
void RTC_Config_owl(void)
{
  NVIC_InitTypeDef NVIC_InitStructure;//NVIC 配置结构体
     EXTI_InitTypeDef EXTI_InitStructure;

   //对 RTC 中断的配置
   RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO,ENABLE);//中断时钟使能
   //中断向量控制 reg 初始化
   NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1);//对优先级的配置，还需要简单了解
   // NVIC_InitStructure.NVIC_IRQChannel = RTC_IRQn;//RTC 中断
   NVIC_InitStructure.NVIC_IRQChannel = RTCAlarm_IRQn;//闹钟中断
   NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
   NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
   NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
   NVIC_Init(&NVIC_InitStructure);


   //闹钟中断接到第 17 线，视为 17 线外引脚中断，
   EXTI_ClearITPendingBit(EXTI_Line17);
   EXTI_InitStructure.EXTI_Line = EXTI_Line17;
   EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;//中断模式
   EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Rising;
   EXTI_InitStructure.EXTI_LineCmd = ENABLE;
   EXTI_Init(&EXTI_InitStructure);



   RCC_APB1PeriphClockCmd(RCC_APB1Periph_PWR , ENABLE);//电源管理部分时钟开启
```

```
  RCC_APB1PeriphClockCmd(RCC_APB1Periph_BKP, ENABLE);//BKP 部分使能
  PWR_BackupAccessCmd(ENABLE);//使能或者失能 RTC 和后备寄存器访问
  //RCC_LSEConfig(RCC_LSE_ON);//使能 LSE
  RCC_LSICmd(ENABLE);//使能 LSI
  //等待晶振启动（有内、外之分）
  while (RCC_GetFlagStatus(RCC_FLAG_LSIRDY) == RESET)
  {}
  RCC_RTCCLKConfig(RCC_RTCCLKSource_LSI);//设置时钟为内部低振
  RCC_RTCCLKCmd(ENABLE);
 //若在读取 RTC 寄存器时，RTC 的 APB1 接口曾经处于禁止状态，
 //则软件首先必须等待 RTC_CRL 寄存器中的 RSF 位(寄存器同步标志)被硬件置’1’。
  RTC_WaitForSynchro();
  RTC_WaitForLastTask();//等待最近一次对 RTC 寄存器的写操作完成
  // RTC_ITConfig(RTC_IT_SEC);//使能秒中断
  RTC_ITConfig( RTC_IT_ALR, ENABLE);//使能闹钟中断
  RTC_WaitForLastTask();
  RTC_SetPrescaler(32767); //分频系数
  RTC_WaitForLastTask();
  RTC_SetCounter(0x0); //初始计数值
  RTC_WaitForLastTask();
  RTC_SetAlarm(5);//闹钟变量
  RTC_WaitForLastTask();
}
闹钟中断服务函数
////////////////////////////////////////////////////////////////////////////////
// 程序名 : void RTCAlarm_IRQHandler(void)
// 作用 :    本函数是闹钟中断处理函数
// 输入参数:无
// 输出参数:无
// 说明:
//
////////////////////////////////////////////////////////////////////////////////
void RTCAlarm_IRQHandler(void)
{
 //若在读取 RTC 寄存器时，RTC 的 APB1 接口曾经处于禁止状态，
 //则软件首先必须等待 RTC_CRL 寄存器中的 RSF 位(寄存器同步标志)被硬件置’1’。
  RTC_WaitForSynchro();
  if(RTC_GetITStatus(RTC_IT_ALR) != RESET)
  {

    EXTI_ClearITPendingBit(EXTI_Line17);//清 17 线中断标志
    //检查 Wake-Up 标志置位
    if(PWR_GetFlagStatus(PWR_FLAG_WU) != RESET)
    {
```

```
        PWR_ClearFlag(PWR_FLAG_WU);//清标志
      }
      RTC_WaitForLastTask();
      RTC_SetAlarm(RTC_GetCounter()+ 3);   //重设闹钟标志
      RTC_WaitForLastTask();
      RTC_ClearITPendingBit(RTC_IT_ALR);//清闹钟标志位
      RTC_WaitForLastTask();
    }
}
```

【实验总结】

各种模式下，理论数据和实测数据如下表所示，对于理论值的详细描述见图5-图9，其中图5是停机和待机模式下电流消耗，图6是活动模式下各个外设电流消耗，图7是睡眠模式下不同时钟频率下电流消耗，图8是运行模式下不同时钟频率下电流消耗，图9是从停机和待机模式下唤醒所需的时间。

由于从待机模式下唤醒时，系统唤醒后的代码执行等同于复位后的执行(采样启动模式引脚、读取复位向量等)。因为所有的寄存器都没有保存数据，所以需要重新初始化外设（中断、定时器、SPI 等等），所以实际从唤醒到系统实际开始正常工作的时间，从几百微秒到几十毫秒不等，取决于使用的外设数目和外部器件，本次试验用的代码初始化时间约200uS。

表 1　各种模式下理论数据和实测数据

时钟模式	工作模式	电压 V	理论电流	实测电流	状态	备注 1	备注 2
外部时钟	活动，32M	3.3	14.8 ；19mA	18.3 mA	RTC 开启，外中断开启，定时器开启，USART 开启		红色的数值是不开任何外设的数值，蓝色的数值开启所以外设的数值。
	活动，8M	3.3	4.6 ；5.5mA	7.2 mA			
	活动，1M	3.3	1.45 ；1.6mA	2.9 mA			
	活动，125k	3.3	1.06 ；1.08mA	2.0 mA			
	活动，15k	3.3	--------	2.0 mA			
	CPU 停止，8M，外设活动	3.3	1.2 ；2.1mA	1.2 mA			
	CPU 停止，1M，外设活动	3.3	0.89 ；1.11mA	1.0 mA			
	停止模式，所有外设	3.3	24uA	21uA		寄存器值不丢	

	停止						
	待机模式	3.3	3.2uA	3uA		只开低振，关看门狗	
内部时钟	活动，32M	3.3	14.1 ；18.3mA	15.8 mA			
	活动，8M	3.3	4 ；4.9mA	7.5 mA			
	活动，1M	3.3	1 ；1.02mA	2.0 mA			
	活动，125k	3.3	0.48 ；0.5mA	1.3 mA			
	活动，10k	3.3	--------	1.3 mA			
	CPU 停止，8M，外设活动	3.3	0.6 ；1.6mA	1.1 mA			
	CPU 停止，1M，外设活动	3.3	0.44 ；0.56mA	0.8 mA			
	停止模式，所有外设停止	3.3	24uA	21uA		寄存器值不丢	
	待机模式	3.3	3.2uA	3uA		只开低振，关看门狗	

停机模式与待机模式各项指标对比：

工作模式	具体模式	状态	电流消耗	唤醒时间	唤醒后初始化时间（RTC 唤醒）
停机模式	电压调节器关闭	RAM、寄存器值保存	14uA	3.6uS	50 uS -100uS
	电压调节器开启		24uA	5.4uS	
待机模式	RTC 运行。看门狗关闭	仅保存电源控制寄存器值，所有的 I/O 引脚处于高阻态	3.2uA	50uS	一般 200uS 从几百微秒到几十毫秒不等，取决于使用的外设数目和外部器件

Table 15. Typical and maximum current consumptions in Stop and Standby modes

Symbol	Parameter	Conditions	Typ(1)		Max	Unit
			V_{DD}/V_{BAT} = 2.4 V	V_{DD}/V_{BAT} = 3.3 V	T_A = 85 °C(2)	
I_{DD}	Supply current in Stop mode	Regulator in Run mode, Low-speed and high-speed internal RC oscillators and high-speed oscillator OFF (no independent watchdog)	23.5	24	200	µA
		Regulator in Low-Power mode, Low-speed and high-speed internal RC oscillators and high-speed oscillator OFF (no independent watchdog)	13.5	14	180	
	Supply current in Standby mode	Low-speed internal RC oscillator and independent watchdog ON	2.6	3.4	-	
		Low-speed internal RC oscillator ON, independent watchdog OFF	2.4	3.2	-	
		Low-speed internal RC oscillator and independent watchdog OFF, low-speed oscillator and RTC OFF	1.7	2	4	
I_{DD_VBAT}	Backup domain supply current	Low-speed oscillator and RTC ON	1.1	1.4	1.9	

1. Typical values are measured at T_A = 25 °C.
2. Based on characterization, not rested in production.

图 5 停机和待机模式下电流消耗

Table 18. Peripheral current consumption

Peripheral		Typical consumption at 25 °C(1)	Unit
APB1	TIM2	0.6	mA
	TIM3	0.6	
	TIM4	0.6	
	SPI2	0.08	
	USART2	0.21	
	USART3	0.21	
	I2C1	0.18	
	I2C2	0.18	
APB2	GPIO A	0.21	
	GPIO B	0.21	
	GPIO C	0.21	
	GPIO D	0.21	
	GPIO E	0.21	
	ADC1(2)	1.4	
	SPI1	0.24	
	USART1	0.35	

1. f_{HCLK} = 36 MHz, $f_{APB1} = f_{HCLK}/2$, $f_{APB2} = f_{HCLK}$, default prescaler value for each peripheral.
2. Specific conditions for ADC: f_{HCLK} = 28 MHz, $f_{APB1} = f_{HCLK}/2$, $f_{APB2} = f_{HCLK}$, $f_{ADCCLK} = f_{APB2}/2$, ADON bit in the ADC_CR2 register is set to 1.

图 6 活动模式下各个外设电流消耗

Table 17. Typical current consumption in Sleep mode, code with data processing code running from Flash or RAM

Symbol	Parameter	Conditions	f_{HCLK}	Typ(1) All peripherals enabled(2)	Typ(1) All peripherals disabled	Unit
I_{DD}	Supply current in Sleep mode	External clock(3)	36 MHz	7.6	3.1	mA
			24 MHz	5.3	2.3	
			16 MHz	3.8	1.8	
			8 MHz	2.1	1.2	
			4 MHz	1.6	1.1	
			2 MHz	1.3	1	
			1 MHz	1.11	0.98	
			500 kHz	1.04	0.96	
			125 kHz	0.98	0.95	
		Running on High Speed Internal RC (HSI), AHB prescaler used to reduce the frequency	36 MHz	7	2.5	
			24 MHz	4.8	1.8	
			16 MHz	3.2	1.2	
			8 MHz	1.6	0.6	
			4 MHz	1	0.5	
			2 MHz	0.72	0.47	
			1 MHz	0.56	0.44	
			500 kHz	0.49	0.42	
			125 kHz	0.43	0.41	

1. Typical values are measures at T_A = 25 °C, V_{DD} = 3.3 V.
2. Add an additional power consumption of 0.8 mA per ADC for the analog part. In applications, this consumption occurs only while the ADC is on (ADON bit is set in the ADC_CR2 register).
3. External clock is 8 MHz and PLL is on when f_{HCLK} > 8 MHz.

图 7 睡眠模式下不同时钟频率下电流消耗

Table 16. Typical current consumption in Run mode, code with data processing running from Flash

Symbol	Parameter	Conditions	f_{HCLK}	Typ(1) All peripherals enabled(2)	Typ(1) All peripherals disabled	Unit
I_{DD}	Supply current in Run mode	External clock(3)	36 MHz	19	14.8	mA
			24 MHz	12.9	10.1	
			16 MHz	9.3	7.4	
			8 MHz	5.5	4.6	
			4 MHz	3.3	2.8	
			2 MHz	2.2	1.9	
			1 MHz	1.6	1.45	
			500 kHz	1.3	1.25	
			125 kHz	1.08	1.06	
		Running on high speed internal RC (HSI), AHB prescaler used to reduce the frequency	36 MHz	18.3	14.1	
			24 MHz	12.2	9.5	
			16 MHz	8.5	6.8	
			8 MHz	4.9	4	
			4 MHz	2.7	2.2	
			2 MHz	1.6	1.4	
			1 MHz	1.02	0.9	
			500 kHz	0.73	0.67	
			125 kHz	0.5	0.48	

1. Typical values are measures at T_A = 25 °C, V_{DD} = 3.3 V.
2. Add an additional power consumption of 0.8 mA per ADC for the analog part. In applications, this consumption occurs only while the ADC is on (ADON bit is set in the ADC_CR2 register).
3. External clock is 8 MHz and PLL is on when f_{HCLK} > 8 MHz.

图 8 运行模式下不同时钟频率下电流消耗

Table 25. Low-power mode wakeup timings

Symbol	Parameter	Conditions	Typ	Unit
$t_{WUSLEEP}$(1)	Wakeup from Sleep mode	Wakeup on HSI RC clock	1.8	µs
t_{WUSTOP}(1)	Wakeup from Stop mode (regulator in run mode)	HSI RC wakeup time = 2 µs	3.6	µs
	Wakeup from Stop mode (regulator in low-power mode)	HSI RC wakeup time = 2 µs, Regulator wakeup from LP mode time = 5 µs	5.4	
$t_{WUSTDBY}$(1)	Wakeup from Standby mode	HSI RC wakeup time = 2 µs, Regulator wakeup from power down time = 38 µs	50	µs

1. The wakeup times are measured from the wakeup event to the point at which the user application code reads the first instruction.

图 9 从停机和待机模式下唤醒所需的时间