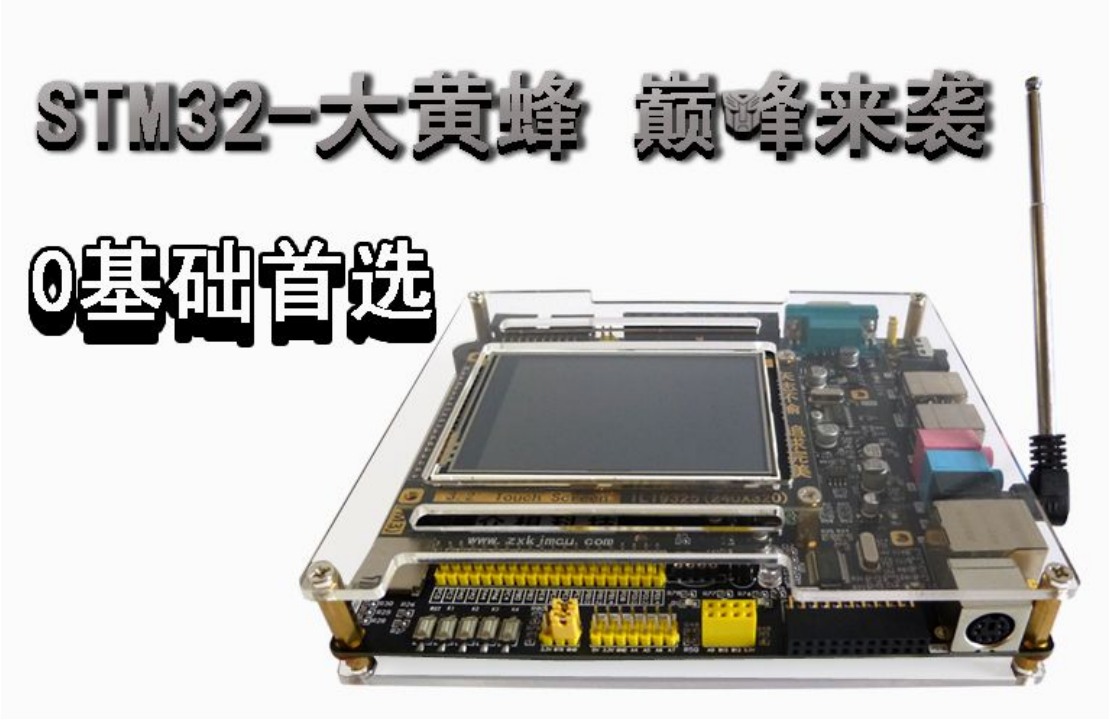
学 ARM 从 STM32 开始

STM32 开发板库函数教程--实战篇



官方网站：http://www.zxkjmcu.com
官方店铺：http://zxkjmcu.taobao.com
官方论坛：http://bbs.zxkjmcu.com
刘洋课堂：http://school.zxkjmcu.com

4.15 STM32 低功耗的工作原理与实验

4.15.1 概述

4.15.1.1 低功耗概念

低功耗就是单片机在空闲时关不不需要启动的功能，以最低的电能消耗维持必要的运行功能的方式。

低功耗的单片机在有些情况下应用很广泛，比如户外手持式的仪表、万用表、卫星、电动汽车等等。

4.15.2 STM32 低功耗模式

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：

- 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：

1、 降低系统时钟

2、 关闭 APB 和 AHB 总线上未被使用的外设时钟。

表 1 低功耗一览表说明

模式	进入	唤醒	对 1.8 区域时钟的影响	对 VDD 区域时钟的影响	电压调节器
睡眠	WFI	任一中断	CPU 时钟关，对	无	开

(SLEEP-NOW 或 SLEEP-ON-EXIT)	WFE	唤醒事件	其他时钟和 ADC 时钟无影响		
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

4.15.2.1 降低系统时钟

在运行模式下，通过对预分频寄存器进行编程，可以降低任意一个系统时钟(SYSCLK、HCLK、PCLK1、PCLK2)的速度。进入睡眠模式前，也可以利用预分频器来降低外设的时钟。

4.15.2.2 外部时钟的控制

在运行模式下，任何时候都可以通过停止为外设和内存提供时钟(HCLK 和 PCLKx)来减少功耗。 为了在睡眠模式下更多地减少功耗，可在执行 WFI 或 WFE 指令前关闭所有外设的时钟。 通过设置 AHB 外设时钟使能寄存器(RCC_AHBENR)、APB2 外设时钟使能寄存器(RCC_APB2ENR)和 APB1 外设时钟使能寄存器(RCC_APB1ENR)来开关各个外设模块的时钟。

4.15.3 待机模式

我们着重讲解待机模式。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。

4.15.3.1 进入待机模式

关于如何进入待机模式，详见表 12。 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。

4.15.3.2 退出待机模式

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(简化的 RTC 框图)，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。

4.15.3.3 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

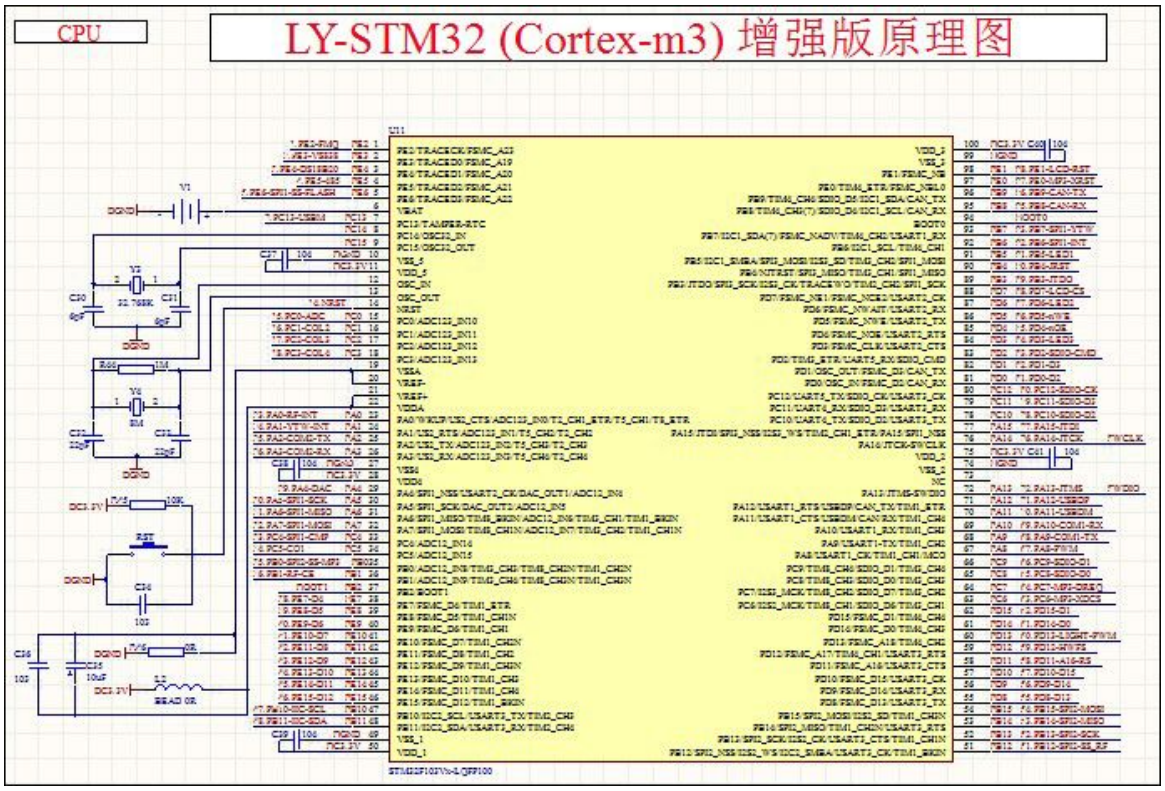
● 被使能的唤醒引脚

调试模式 默认情况下，如果在进行调试微处理器时，使微处理器进入停止或待机模式，将失去调试连接。这是因为 Cortex™M3 的内核失去了时钟。然而，通过设置 DBGMCU_CR 寄存器中的某些配置位，可以在使用低功耗模式下调试软件。

4.15.4 STM32 低功耗实验目的

编写一段程序，倒计时 10 秒后，进入待机模式,通过打印程序把设计好的过程打印到屏幕上。

4.15.5 硬件设计



利用实验板上主芯片 SMT32 本身的功能，通过软件设计设置倒计时时间，进入到待机模式。

4.15.6 低功耗软件设计

4.15.6.1 STM32 库函数文件

```
stm32f10x_gpio.c
stm32f10x_rcc.c
Misc.c              // 中断控制字（优先级设置）库函数
stm32f10x_exti.c // 外部中断库处理函数
stm32f10x_tim.c  // 定时器库处理函数
stm32f10x_usart.c  // 串口通讯函数
stm32f10x_pwr.c  // DMA
```

本节实验及以后的实验我们都是用到库文件，其中 stm32f10x_gpio.h 头文件包含了 GPIO 端口的定义。stm32f10x_rcc.h 头文件包含了系统时钟配置函数以及相关的外设时钟使能函数，所以我们要把这两个头文件对应的 stm32f10x_gpio.c 和 stm32f10x_rcc.c 加到工程中；Misc.c 库函数主要包含了中断优先级的设置，stm32f10x_exti.c 库函数主要包含了外部中断设置参数，tm32f10x_tim.c 库函数主要包含定时器设置，tm32f10x_usart.c 库函数主要包含串行通讯设置，tm32f10x_pwr.c 库函数主要包含电源管理相关设置，这些函数也要添加到函数库中。以上库文件包含了本次实验所有要用到的函数使用功能。

4.15.6.2 自定义头文件

```
pbdata.h
pbdata.c
```

同时我们自己也创建了两个公共的文件，这两个文件主要存放我们自定义的公共函数和全局变量，以方便以后每个功能模块之间传递参数。

4.15.6.3 pbdata.h 文件里的内容是

```
#ifndef _pbdata_H
#define _pbdata_H

#include "stm32f10x.h"
#include "misc.h"
#include "stm32f10x_exti.h"
#include "stm32f10x_tim.h"
#include "stm32f10x_usart.h"
#include "stm32f10x_pwr.h"
```

```
#include "stdio.h"

//定义变量
extern u8 dt;

//定义函数
void RCC_HSE_Configuration(void);
void delay(u32 nCount);
void delay_us(u32 nus);
void delay_ms(u16 nms);

#endif
```

语句 #ifndef、#endif 是为了防止 pbdata.h 文件被多个文件调用时出现错误提示。如果不加这两条语句，当两个文件同时调用 pbdata 文件时，会提示重复调用错误。

4.15.6.4 pbdata.c 文件里的内容是

```
#include "pbdata.h"

u8 dt=0;

void RCC_HSE_Configuration(void) //HSE 作为 PLL 时钟，PLL 作为 SYSCLK
{
   RCC_DeInit(); /*将外设 RCC 寄存器重设为缺省值 */
   RCC_HSEConfig(RCC_HSE_ON);    /*设置外部高速晶振（HSE） HSE 晶振打开(ON)*/

   if(RCC_WaitForHSEStartUp() == SUCCESS) {  /*等待 HSE 起振，  SUCCESS：HSE 晶振稳定且就绪*/

   RCC_HCLKConfig(RCC_SYSCLK_Div1);/*设置 AHB 时钟(HCLK)RCC_SYSCLK_Div1——AHB 时钟 = 系统时*/
   RCC_PCLK2Config(RCC_HCLK_Div1); /*设置高速 AHB 时钟(PCLK2)RCC_HCLK_Div1——APB2 时钟 = HCLK*/
   RCC_PCLK1Config(RCC_HCLK_Div2); /*设置低速 AHB 时钟(PCLK1)RCC_HCLK_Div2——APB1 时钟 = HCLK / 2*/

   RCC_PLLConfig(RCC_PLLSource_HSE_Div1, RCC_PLLMul_9);/*设置 PLL 时钟源及倍频系数*/
   RCC_PLLCmd(ENABLE);    /*使能 PLL */
   while(RCC_GetFlagStatus(RCC_FLAG_PLLRDY) == RESET) ; /*检查指定的 RCC 标志位(PLL 准备好标志)设置与否*/
```

```
   RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK);  /*设置系统时钟（SYSCLK） */
   while(RCC_GetSYSCLKSource() != 0x08);      /*0x08：PLL 作为系统时钟 */
 }
}

void delay(u32 nCount)
{
   for(;nCount!=0;nCount--);
}
/*******************************************************************************
* 名    称：delay_us(u32 nus)
* 功    能：微秒延时函数
* 入口参数：u32  nus
* 出口参数：无
* 说    明：
* 调用方法：无
*******************************************************************************/
void delay_us(u32 nus)
{
    u32 temp;
    SysTick->LOAD = 9*nus;
    SysTick->VAL=0X00;//清空计数器
    SysTick->CTRL=0X01;//使能，减到零是无动作，采用外部时钟源
    do
    {
     temp=SysTick->CTRL;//读取当前倒计数值
    }while((temp&0x01)&&(!(temp&(1<<16))));//等待时间到达

    SysTick->CTRL=0x00; //关闭计数器
    SysTick->VAL =0X00; //清空计数器
}

/*******************************************************************************
* 名    称：delay_ms(u16 nms)
* 功    能：毫秒延时函数
* 入口参数：u16 nms
* 出口参数：无
* 说    明：
* 调用方法：无
*******************************************************************************/
void delay_ms(u16 nms)
{
    //注意 delay_ms 函数输入范围是 1-1864
```

```
    //所以最大延时为 1.8 秒

    u32 temp;
    SysTick->LOAD = 9000*nms;
    SysTick->VAL=0X00;//清空计数器
    SysTick->CTRL=0X01;//使能，减到零是无动作，采用外部时钟源
    do
    {
     temp=SysTick->CTRL;//读取当前倒计数值
    }while((temp&0x01)&&(!(temp&(1<<16))));//等待时间到达
    SysTick->CTRL=0x00; //关闭计数器
    SysTick->VAL =0X00; //清空计数器
}
```

4.15.7 STM32 系统时钟配置 SystemInit()

每个工程都必须在开始时配置并启动 STM32 系统时钟。

4.15.8 GPIO 引脚时钟使能

```
      RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA,ENABLE);
      RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1,ENABLE);//设置串口 1 时钟
使能
      RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO,ENABLE);//功能复用 IO 时钟
使能
```

本节实验用到了 PA 端口，所以要把 PA 的时钟打开；串口 1 时钟打开；因为要与外部芯片通讯，所以要打开功能复用时钟。

4.15.9 GPIO 管脚电平控制函数

在主程序中采用 while(1)循环语句，采用查询的方式等待 ADC 模拟量转换完毕，初始化完成以后要在主程序中采集模拟量，加入滤波、取平均值等措施然后转换送出打印。下面是 while(1)语句中详细的内容。

```
   while(1)
  {
    printf("倒计时：5 秒\r\n");
    delay_ms(1000);
    printf("倒计时：4 秒\r\n");
    delay_ms(1000);
    printf("倒计时：3 秒\r\n");
```

```
    delay_ms(1000);
    printf("倒计时：2 秒\r\n");
    delay_ms(1000);
    printf("倒计时：1 秒\r\n");
    delay_ms(1000);
    printf("进入待机模式\r\n");
    Sys_Standby();
  }
```

4.15.10 stm32f10x_it.c 文件里的内容是

在中断处理 stm32f10x_it.c 文件里中串口 1 子函数非空，进入中断处理函数后，打开串口 1，和外部设备联络好。

```
#include "stm32f10x_it.h"
#include "stm32f10x_exti.h"
#include "stm32f10x_rcc.h"
#include "misc.h"
#include "pbdata.h"

void NMI_Handler(void)
{
}

void USART1_IRQHandler(void)
{
  if(USART_GetITStatus(USART1,USART_IT_RXNE)!=RESET)
  {
      USART_SendData(USART1,USART_ReceiveData(USART1));
      while(USART_GetFlagStatus(USART1,USART_FLAG_TXE)==RESET);
  }
}
```

4.15.12 main.c 文件里的内容是

大家都知道 prinif 重定向是把需要显示的数据打印到显示器上。在这个试验中把待机模式的执行过程通过 prinif 重定向打印到串口精灵上。也就是送到显示器显示出来。下面是待机模式进入和退出的相关说明。

表 2 待机模式

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。

```
#include "pbdata.h"

void RCC_Configuration(void);
void GPIO_Configuration(void);
void NVIC_Configuration(void);
void USART_Configuration(void);
void Sys_Standby(void);

int fputc(int ch,FILE *f)//打印输出函数
{
    USART_SendData(USART1,(u8)ch);
    while(USART_GetFlagStatus(USART1,USART_FLAG_TXE)==RESET);
    return ch;
}

int main(void)
{
   RCC_Configuration(); //系统时钟初始化
   GPIO_Configuration();//端口初始化
   USART_Configuration();
   NVIC_Configuration();

   while(1)
   {
      printf("倒计时：5 秒\r\n");
      delay_ms(1000);
      printf("倒计时：4 秒\r\n");
      delay_ms(1000);
      printf("倒计时：3 秒\r\n");
      delay_ms(1000);
      printf("倒计时：2 秒\r\n");
      delay_ms(1000);
      printf("倒计时：1 秒\r\n");
      delay_ms(1000);
```

```
            printf("进入待机模式\r\n");
            Sys_Standby();
        }
    }

    void Sys_Standby(void)//低功耗待机模式子函数
    {
        //选择待机模式
        NVIC_SystemLPConfig(NVIC_LP_SLEEPDEEP,ENABLE);//深度睡眠使能
        //使能 PWR 外设时钟
        RCC_APB1PeriphClockCmd(RCC_APB1Periph_PWR,ENABLE);
        //使能唤醒管脚
        PWR_WakeUpPinCmd(ENABLE);
        //进入待机模式
        PWR_EnterSTANDBYMode();
    }

    void RCC_Configuration(void)
    {
        SystemInit();//72m
        RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA,ENABLE);
        RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1,ENABLE);
        RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO,ENABLE);
    }

void GPIO_Configuration(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;
    //LED
    GPIO_InitStructure.GPIO_Pin=GPIO_Pin_9;//TX
    GPIO_InitStructure.GPIO_Speed=GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode=GPIO_Mode_AF_PP;
    GPIO_Init(GPIOA,&GPIO_InitStructure);

    GPIO_InitStructure.GPIO_Pin=GPIO_Pin_10;//RX
    GPIO_InitStructure.GPIO_Mode=GPIO_Mode_IN_FLOATING;
    GPIO_Init(GPIOA,&GPIO_InitStructure);
}

void NVIC_Configuration(void)
{
    NVIC_InitTypeDef NVIC_InitStructure;

    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1);
```

```
    NVIC_InitStructure.NVIC_IRQChannel = USART1_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);
}

void USART_Configuration(void)
{
    USART_InitTypeDef  USART_InitStructure;

    USART_InitStructure.USART_BaudRate=9600;
    USART_InitStructure.USART_WordLength=USART_WordLength_8b;
    USART_InitStructure.USART_StopBits=USART_StopBits_1;
    USART_InitStructure.USART_Parity=USART_Parity_No;
    USART_InitStructure.USART_HardwareFlowControl=USART_HardwareFlowControl_
None;
    USART_InitStructure.USART_Mode=USART_Mode_Rx|USART_Mode_Tx;

    USART_Init(USART1,&USART_InitStructure);
    USART_ITConfig(USART1,USART_IT_RXNE,ENABLE);
    USART_Cmd(USART1,ENABLE);
    USART_ClearFlag(USART1,USART_FLAG_TC);
}
```

4.15.13 应用函数

查 STM32 固件使用手册列出了这节课程主要的应用函数，方便大家学习。

4.15.13.1 NVIC_SystemLPConfig

表 3 描述了函数 NVIC_SystemLPConfig

函数名称	NVIC_SystemLPConfig
函数原型	Void NVIC_SystemLPConfig(u8 LowPowerMode,FunctionalState NewState)
功能描述	选择系统进入低功耗模式的条件
输入参数 1	LowPowerMode：系统进入低功耗模式的新模式 参阅 Section：LowPowerMode 查阅更多该参数允许取值范围
输入参数 2	NewState：LP 条件的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

附表. LowPowerMode 值

LowPowerMode	描述
NVIC_LP_SEVONPEND	根据待处理请求唤醒
NVIC_LP_SLEEPDEEP	深度睡眠使能
NVIC_LP_SLEEPONEXIT	退出 ISR 后睡眠

4.15.13.2 PWR_WakeUpPinCmd

表 4 描述了函数 PWR_WakeUpPinCmd

函数名称	PWR_WakeUpPinCmd
函数原型	void PWR_WakeUpPinCmd(FunctionalState NewState)
功能描述	使能或者失能唤醒管脚功能
输入参数	NewState: 唤醒管脚功能的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

4.15.13.3 PWR_EnterSTANDBYMode

表 4 描述了函数 PWR_EnterSTANDBYMode

函数名称	PWR_EnterSTANDBYMode
函数原型	void PWR_EnterSTANDBYMode(void)
功能描述	进入待命（STANDBY）模式
输入参数	NewState: 唤醒管脚功能的新状态 这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	__WFI(),

4.15.14 程序下载

请根据下图所指向的 7 个重点区域配置。其中（1）号区域根据自己机器的实际情况选择，我的机器虚拟出来的串口号是 COM3。（2）号区域请自己选择程序所在的文件夹。（7）号区域当程序下载完后，进度条会到达最右边，并且提示一切正常。（4、5、6）号区域一定要按照上图显示的设置。当都设置好以后就可以直接点击（3）号区域的开始编程按钮下传程序了。

本节实验的源代码在光盘中：（LY-STM32 光盘资料\1.课程\1,基础篇\基础篇 28. STM32 低功耗工作原理和实验\程序）

4.15.15 实验效果图

程序写入实验板后，使用公司开发的多功能监视系统，在串口调试界面中的接收区就会就会接收到程序定时 5 秒后自动进入待机模式。