# 实验2：待机唤醒实验

## 实验目的

* 结合数据手册，理解待机模式进入操作和唤醒方法；
* 深入理解封装库函数设计原理，包括函数的用途和与相关寄存器之间的关系。
* 学习代码设计逻辑和书写规范，复习嵌入式C语言；

## 实验内容

* 参考实验例程、《STM32F1开发指南-库函数版本\_V3.0》和《STM32数据手册》，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，断电整个1.8V供电区域（CPU核心、存储器、内置数字外设）， 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（包含唤醒逻辑）。
* 进入待机模式配置包括：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，在配置好上述寄存器后，运行PWR\_EnterSTANDBYMode()函数，执行WFI或WFE命令进入待机状态。退出待机模式：在配置好PWR\_CSR寄存器的基础上，给予WKUP引脚上升沿唤醒或通过RTC闹钟事件的上升沿。

从待机模式唤醒后的代码执行等同于复位后的执行。

* 进入停止模式配置包括：设置Cortex™-M3系统控制寄存器中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，在配置好上述寄存器后，运行PWR\_EnterSTOPMode ()函数，执行WFI命令进入停止状态。

退出待机模式：通过RTC闹钟中断的上升沿唤醒。

从停止模式唤醒后的先执行RTC闹钟事件中断函数，再继续从进入停止模式的地方继续执行代码。

## 程序框图

 

图1 待机模式主要程序设计流程图 图2 停止模式主要程序设计流程图

## 主要程序

针对STM32进入待机模式和停止模式的配置流程、唤醒方式和唤醒后系统时钟的选择做如下详细介绍（主要参考STM32官方标准库和《STM32F1开发指南-库函数版本\_V3.0》文件）：

1. **Sys\_Enter\_Standby ()函数做详细分析：**
2. **void** Sys\_Enter\_Standby(**void**)
3. {
4. RCC\_APB2PeriphResetCmd(0X01FC,DISABLE);
5. Sys\_Standby();
6. }

* **RCC\_APB2PeriphResetCmd()函数：**

1. **void**RCC\_APB2PeriphResetCmd(uint32\_t RCC\_APB2Periph,FunctionalState NewState)
2. {
3. assert\_param(IS\_RCC\_APB2\_PERIPH(RCC\_APB2Periph));
4. assert\_param(IS\_FUNCTIONAL\_STATE(NewState));
5. **if** (NewState != DISABLE)
6. RCC->APB2RSTR |= RCC\_APB2Periph;
7. **else**
8. RCC->APB2RSTR &= ~RCC\_APB2Periph;
9. }

由前面的实验解析可得，RCC->APB2RSTR所指向的地址为：0x4002100C，对应于数据手册RCC\_APB2RSTR寄存器地址。为该寄存器写入0X01FC即对IO端口A-G进行复位操作，将GPIO外设关闭（此步骤操作仅以关闭GPIO外设为例，若开启了其他外设，同样需要关闭，以达到最低功耗）。

* **Sys\_Standby ()函数：**

1. **void** Sys\_Standby(**void**)
2. {
3. RCC\_APB1PeriphClockCmd(RCC\_APB1Periph\_PWR, ENABLE);
4. PWR\_WakeUpPinCmd(ENABLE);
5. PWR\_EnterSTANDBYMode();
6. }

 RCC\_APB1PeriphClockCmd()函数配置地址为0x4002101C的RCC\_APB1ENR寄存器（因为要配置电源控制寄存器，所以必须先使能挂载在APB1上的电源时钟）。

* **PWR\_WakeUpPinCmd ()函数：**

1. **void** PWR\_WakeUpPinCmd(FunctionalState NewState)
2. {
3. /\* Check the parameters \*/
4. assert\_param(IS\_FUNCTIONAL\_STATE(NewState));
5. \*(\_\_IO uint32\_t \*) CSR\_EWUP\_BB = (uint32\_t)NewState;
6. }

根据定义#define CSR\_EWUP\_BB (PERIPH\_BB\_BASE + (CSR\_OFFSET \* 32) + (EWUP\_BitNumber \* 4))可得 CSR\_EWUP\_BB所指向的地址为0x420E00A0，对该位赋值1即对PWR\_CSR寄存器的第8位置1，即使能WKUP引脚。（位带操作）

* **PWR\_EnterSTANDBYMode ()函数：（待机模式）**

1. **void** PWR\_EnterSTANDBYMode(**void**)
2. {
3. /\* Clear Wake-up flag \*/
4. PWR->CR |= PWR\_CR\_CWUF;
5. /\* Select STANDBY mode \*/
6. PWR->CR |= PWR\_CR\_PDDS;
7. /\* Set SLEEPDEEP bit of Cortex System Control Register \*/
8. SCB->SCR |= SCB\_SCR\_SLEEPDEEP;
9. /\* This option is used to ensure that store operations are completed \*/
10. #if defined ( \_\_CC\_ARM   )
11. \_\_force\_stores();
12. #endif
13. /\* Request Wait For Interrupt \*/
14. \_\_WFI();
15. }

对PWR\_CR的CWUF和PDDS位分别置1，清除WUF唤醒位并选择CPU进入深睡眠时进入待机模式。对系统控制寄存器（SCB\_SCR）的第二位置1（设置SLEEPDEEP位）。在完成上述操作后，运行\_\_WFI()函数，使得单片机进入待机状态。（根据定义#define \_\_WFI \_\_wfi ，\_\_wfi是编译器内置函数，函数内部使用调用了相应的汇编指令，使用该指令时需使用\_\_WFI()函数格式。）同样，还有WFE指令，两者区别为：其中WFI指令决定了它需要用中断唤醒，而WFE则决定了它可用事件来唤醒。由数据手册可知，进入待机操作，执行WFI或WFE均可。

* **PWR\_** **EnterSTOPMode ()函数：（停止模式）**

1. **void** PWR\_EnterSTOPMode(uint32\_t PWR\_Regulator, uint8\_t PWR\_STOPEntry)
2. {
3. uint32\_t tmpreg = 0;
4. assert\_param(IS\_PWR\_REGULATOR(PWR\_Regulator));
5. assert\_param(IS\_PWR\_STOP\_ENTRY(PWR\_STOPEntry));
6. tmpreg = PWR->CR;
7. tmpreg &= CR\_DS\_MASK;
8. tmpreg |= PWR\_Regulator;
9. PWR->CR = tmpreg;
10. SCB->SCR |= SCB\_SCR\_SLEEPDEEP;
11. **if**(PWR\_STOPEntry == PWR\_STOPEntry\_WFI)
12. \_\_WFI();
13. **else**
14. \_\_WFE();
15. SCB->SCR &= (uint32\_t)~((uint32\_t)SCB\_SCR\_SLEEPDEEP);
16. }

该函数首先对PWR\_CR寄存器的PDDS、LPSD位清空，然后根据用户选择的调节器状态进行配置，PWR\_Regulator参数可选PWR\_Regulator\_ON和 PWR\_Regulator\_LowPower两个参数，分别对应在停机模式下电压调节器处于开启或者低功耗模式。之后对系统控制寄存器（SCB\_SCR）的第二位置1（设置SLEEPDEEP位）。当RTC中断事件唤醒之后复位系统控制寄存器的SLEEPDEEP位。

1. **RTC时钟唤醒停机模式和待机模式详细分析：**
2. **void** EXTI4\_IRQHandler(**void**)
3. {
4. delay\_ms(10);
5. **if**(!GPIO\_ReadInputDataBit(GPIOE, GPIO\_Pin\_4))
6. {
7. EXTI\_ClearITPendingBit(EXTI\_Line4);
8. RTC\_SetAlarm(RTC\_GetCounter()+3);
9. RTC\_ITConfig(RTC\_IT\_ALR, ENABLE);
10. RTC\_WaitForLastTask();
11. Sys\_Enter\_Standby();
12. }
13. }

该试验采用KEY0按键触发中断，进入待机或者停止模式。待机和停止模式通过Sys\_Enter\_Standby() 函数进行选择。

此处测试RTC闹钟事件唤醒两种低功耗模式，在进入低功耗模式之前，通过RTC\_SetAlarm()函数实现自定义时间后进行唤醒，之后通过RTC\_ITConfig()函数使能RTC闹钟中断，RTC\_WaitForLastTask()函数等待写操作完成，Sys\_Enter\_Standby()函数进入所选低功耗模式。

1. **RTCAlarm\_IRQHandler()函数详细分析：**
2. **void** RTCAlarm\_IRQHandler(**void**)
3. {
4. **if**(RTC\_GetFlagStatus(RTC\_IT\_ALR))
5. {
6. EXTI\_ClearITPendingBit(EXTI\_Line17);
7. EXTI\_ClearITPendingBit(EXTI\_Line4);
8. RTC\_ClearITPendingBit(RTC\_IT\_ALR);
9. RTC\_WaitForLastTask();
10. **if**(PWR\_GetFlagStatus(PWR\_FLAG\_WU) != RESET)
11. {
12. PWR\_ClearFlag(PWR\_FLAG\_WU);
13. }
14. SystemInit();
15. }
16. }

该函数为RTC闹钟中断服务函数，当在停止模式下，产生RTC闹钟中断唤醒后，首先执行RTCAlarm\_IRQHandler()函数内容，在此函数中需要清除所有可能产生的中断标志位，包括RTC闹钟中断标志和之前看起的KEY0外部中断标志，否则可能唤醒失败。值得注意的是，在停止模式下，当被唤醒后，系统首选HIS RC振荡器作为系统时钟（8MHz），为了使得系统恢复72MHz频率，此时需要调用系统初始化函数SystemInit()对系统时钟重新配置。

在待机模式下SRAM和寄存器内容全部丢失，所以唤醒后只能执行复位操作从启动文件中配置系统时钟，同样调用SystemInit()函数进行系统时钟配置。

1. **SystemInit ()函数做详细分析：（该函数的解析穿插在代码注释中）**
2. **void** SystemInit (**void**)
3. {
4. RCC->CR |= (uint32\_t)0x00000001;  //设置RCC\_CR寄存器HSION位，开启内部8MHz振荡器
5. #ifndef STM32F10X\_CL
6. RCC->CFGR &= (uint32\_t)0xF8FF0000;  /\*清除RCC\_CFGR寄存器SW（系统时钟切换）, HPRE（AHB预分频）, PPRE1（低速APB预分频（APB1））, PPRE2（高速APB预分频（APB2））, ADCPRE（ADC预分频）和MCO（微控制器时钟输出） 位\*/
7. #else
8. ~~RCC->CFGR &= (uint32\_t)0xF0FF0000;~~ //该部分代码不执行
9. #endif /\* STM32F10X\_CL \*/
10. RCC->CR &= (uint32\_t)0xFEF6FFFF;        /\* 复位RCC\_CR寄存器HSEON（外部高速时钟使能）, CSSON （时钟安全系统使能）和 PLLON （PLL使能）bits \*/
11. RCC->CR &= (uint32\_t)0xFFFBFFFF;    /\* 复位RCC\_CR寄存器HSEBYP（外部高速时钟旁路） 位\*/
12. RCC->CFGR &= (uint32\_t)0xFF80FFFF;    /\* 复位RCC\_CFGR寄存器PLLSRC（PLL输入时钟源）, PLLXTPRE（HSE分频器作为PLL输入）, PLLMUL（PLL倍频系数）和USBPRE/OTGFSPRE（USB预分频）位 \*/
13. #ifdef STM32F10X\_CL  //该部分代码不执行
14. ~~/\* Reset PLL2ON and PLL3ON bits \*/~~
15. ~~RCC->CR &= (uint32\_t)0xEBFFFFFF;~~
16. ~~/\* Disable all interrupts and clear pending bits  \*/~~
17. ~~RCC->CIR = 0x00FF0000;~~
18. ~~/\* Reset CFGR2 register \*/~~
19. ~~RCC->CFGR2 = 0x00000000;~~
20. ~~#elif defined (STM32F10X\_LD\_VL) || defined (STM32F10X\_MD\_VL) || (defined STM32F10X\_HD\_VL)~~
21. ~~RCC->CIR = 0x009F0000;    /\* Disable all interrupts and clear pending bits  \*/~~
22. ~~/\* Reset CFGR2 register \*/~~
23. ~~RCC->CFGR2 = 0x00000000;~~
24. #else
25. RCC->CIR = 0x009F0000;    /\* 清除CSSC（清除时钟安全系统中断）、PLLRDYC（清除PLL就绪中断）、HSERDYC（清除HSE就绪中断）、HSIRDYC（清除HSI就绪中断）、LSERDYC（清除LSE就绪中断）、LSIRDYC（清除LSI就绪中断）中断标志位\*/
26. #endif /\* STM32F10X\_CL \*/
28. #if defined (STM32F10X\_HD) || (defined STM32F10X\_XL) || (defined STM32F10X\_HD\_VL)
29. #ifdef DATA\_IN\_ExtSRAM
30. ~~SystemInit\_ExtMemCtl();~~ //该部分代码不执行
31. #endif /\* DATA\_IN\_ExtSRAM \*/
32. #endif
33. SetSysClock();  //配置系统时钟频率，HCLK，PCLK2和PCLK1预分频器
34. #ifdef VECT\_TAB\_SRAM   //该部分代码不执行
35. ~~SCB->VTOR = SRAM\_BASE | VECT\_TAB\_OFFSET; // Vector Table Relocation in Internal SRAM.~~
36. #else
37. SCB->VTOR = FLASH\_BASE | VECT\_TAB\_OFFSET; //内部FLASH向量表重新定位
38. #endif
39. }

SystemInit()函数首先对RCC一些寄存器进行复位，清除后续需要配置的相关位，以便后续配置工作。之后调用SetSysClock()函数，对系统时钟进行配置。

* **SetSysClock ()函数：**

1. **static** **void** SetSysClock(**void**)
2. {
3. #ifdef SYSCLK\_FREQ\_HSE
4. ~~SetSysClockToHSE();~~
5. ~~#elif defined SYSCLK\_FREQ\_24MHz~~
6. ~~SetSysClockTo24();~~
7. ~~#elif defined SYSCLK\_FREQ\_36MHz~~
8. ~~SetSysClockTo36();~~
9. ~~#elif defined SYSCLK\_FREQ\_48MHz~~
10. ~~SetSysClockTo48();~~
11. ~~#elif defined SYSCLK\_FREQ\_56MHz~~
12. ~~SetSysClockTo56();~~
13. #elif defined SYSCLK\_FREQ\_72MHz
14. SetSysClockTo72();
15. #endif
16. /\* 如果以上定义均未启用，则将HSI用作系统时钟源（复位后的默认设置） \*/
17. }

根据定义内容，调用SetSysClockTo72()函数。

* **SetSysClockTo72 ()函数：**

1. **static** **void** SetSysClockTo72(**void**)
2. {
3. \_\_IO uint32\_t StartUpCounter = 0, HSEStatus = 0;
4. RCC->CR |= ((uint32\_t)RCC\_CR\_HSEON);    /\* RCC\_CR寄存器 HSEON位置1，开启 HSE振荡器\*/
5. **do**
6. {
7. HSEStatus = RCC->CR & RCC\_CR\_HSERDY;
8. StartUpCounter++;
9. } **while**((HSEStatus == 0) && (StartUpCounter != HSE\_STARTUP\_TIMEOUT));
10. /\* 等待HSE时钟就位或者时间溢出而退出 \*/
11. **if** ((RCC->CR & RCC\_CR\_HSERDY) != RESET)
12. HSEStatus = (uint32\_t)0x01;   //如果HSE时钟就位，则让HSEStatus为1
13. **else**
14. HSEStatus = (uint32\_t)0x00;  //如果HSE时钟就位失败，则让HSEStatus为0
15. **if** (HSEStatus == (uint32\_t)0x01)  //如果HSE时钟就位
16. {
17. FLASH->ACR |= FLASH\_ACR\_PRFTBE;      /\* Enable Prefetch Buffer \*/
18. /\* Flash 2 wait state \*/
19. FLASH->ACR &= (uint32\_t)((uint32\_t)~FLASH\_ACR\_LATENCY);
20. FLASH->ACR |= (uint32\_t)FLASH\_ACR\_LATENCY\_2;
21. RCC->CFGR |= (uint32\_t)RCC\_CFGR\_HPRE\_DIV1;        /\* HCLK = SYSCLK \*/
22. RCC->CFGR |= (uint32\_t)RCC\_CFGR\_PPRE2\_DIV1;      /\* PCLK2 = HCLK=72M \*/
23. RCC->CFGR |= (uint32\_t)RCC\_CFGR\_PPRE1\_DIV2;      /\* PCLK1 = HCLK/2=36M \*/
24. #ifdef STM32F10X\_CL  //此代码不执行
25. ~~RCC->CFGR2 &= (uint32\_t)~(RCC\_CFGR2\_PREDIV2 | RCC\_CFGR2\_PLL2MUL |~~
26. ~~RCC\_CFGR2\_PREDIV1 | RCC\_CFGR2\_PREDIV1SRC);~~
27. ~~RCC->CFGR2 |= (uint32\_t)(RCC\_CFGR2\_PREDIV2\_DIV5 | RCC\_CFGR2\_PLL2MUL8 |~~
28. ~~RCC\_CFGR2\_PREDIV1SRC\_PLL2 | RCC\_CFGR2\_PREDIV1\_DIV5);~~
29. ~~/\* Enable PLL2 \*/~~
30. ~~RCC->CR |= RCC\_CR\_PLL2ON;~~
31. ~~/\* Wait till PLL2 is ready \*/~~
32. **~~while~~**~~((RCC->CR & RCC\_CR\_PLL2RDY) == 0)  ;~~
33. ~~/\* PLL configuration: PLLCLK = PREDIV1 \* 9 = 72 MHz \*/~~
34. ~~RCC->CFGR &= (uint32\_t)~(RCC\_CFGR\_PLLXTPRE | RCC\_CFGR\_PLLSRC | RCC\_CFGR\_PLLMULL);~~
35. ~~RCC->CFGR |= (uint32\_t)(RCC\_CFGR\_PLLXTPRE\_PREDIV1 | RCC\_CFGR\_PLLSRC\_PREDIV1 | RCC\_CFGR\_PLLMULL9);~~
36. #else
37. RCC->CFGR &= (uint32\_t)((uint32\_t)~(RCC\_CFGR\_PLLSRC | RCC\_CFGR\_PLLXTPRE |
38. RCC\_CFGR\_PLLMULL));  //清除PLLSRC（PLL输入时钟源）、 //PLLXTPRE（PREDIV1分频因子的低位）和PLLMULL（PLL倍频系数）位
39. RCC->CFGR |= (uint32\_t)(RCC\_CFGR\_PLLSRC\_HSE | RCC\_CFGR\_PLLMULL9);  //选择HES 8M时钟作为PLL输入源（需在RCC\_CFGR2寄存器PREDIV1SRC位置0），PLL9倍频8M\*9=72M
40. #endif /\* STM32F10X\_CL \*/
41. RCC->CR |= RCC\_CR\_PLLON;        /\*RCC\_CR寄存器PLLON位置1，开启PLL \*/
42. **while**((RCC->CR & RCC\_CR\_PLLRDY) == 0) ;     /\* 等待PLL就位\*/
43. RCC->CFGR &= (uint32\_t)((uint32\_t)~(RCC\_CFGR\_SW));
44. RCC->CFGR |= (uint32\_t)RCC\_CFGR\_SW\_PLL;  //选择PLL输出（72M）作为系统时钟
45. **while** ((RCC->CFGR & (uint32\_t)RCC\_CFGR\_SWS) != (uint32\_t)0x08) ;   /\*等待PLL作为系统时钟源就位\*/
46. }
47. **else**
48. { }
49. }

SystemInit()函数在启动初期就会被调用，将系统时钟配置位72MHz。

## 实验现象

**待机模式**：MCU上电，LED亮四秒后，进入待机模式，LED熄灭，通过按压WKUP按键使得MCU唤醒，（由于采用复位方式唤醒）唤醒以后代码重新执行，LED亮四秒后熄灭，进入待机模式。

**停止模式：**LED亮四秒后，开始不间断闪烁，按下KEY0按键（执行RTC配置代码，三秒后唤醒，并执行进入停止模式代码）进入停止模式，三秒后MCU被唤醒，LED继续闪烁。