实验 17 -功耗管理-PIN 唤醒实验

1. 实验目的

掌握 NRF24LE1 的低功耗模式的配置和使用。
掌握 NRF24LE1 的 PIN 唤醒。

2. 实验内容

通过接收串口命令，配置 NRF24LE1 进入以下 4 种低功耗模式：

- 深度睡眠，唤醒后 NRF24LE1 会复位。
- 存储器维持，定时器关闭，唤醒后 NRF24LE1 会复位。
- 存储器维持，定时器开启，唤醒后 NRF24LE1 会复位。
- 寄存器维持，唤醒后 NRF24LE1 不会复位

通过 PIN 唤醒 NRF24LE1，观察串口输出内容和 NRF24LE1 的运行情况。

3. 实验原理

当 NRF24LE1 复位或上电后，进入活动工作模式，功能由软件进行控制。为进入其中一种低功耗模式，PWRDWN 寄存器必须按照所选定的工作模式写入相应内容。若要从低功耗模式再次进行活动模式，则需要一个唤醒源来激活，本例程演示通过唤醒源 PIN 激活。

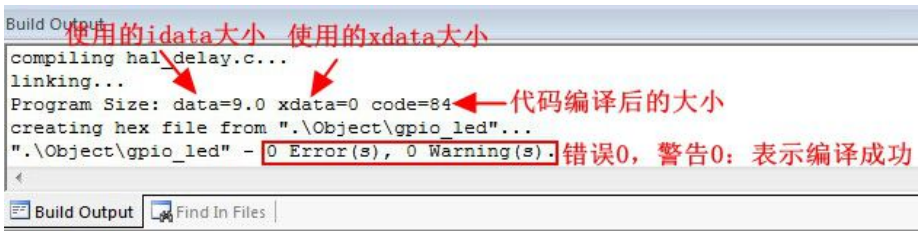
表 1：NRF24LE1 工作模式

模式	简要说明
深度睡眠	❑ 运行的功能：引脚包括唤醒滤波器。 ❑ 唤醒源：外部引脚。 ❑ 启动时间：小于由 RCOSC16M 启动时间，即小于 100us。 ❑ 说明：此模式下的 PIN 唤醒将会导致系统复位（唤醒后，程序从复位向量处开始执行）。
存储器维持、定时器关闭	❑ 运行的功能：比深度睡眠下增加： • 电源管理； • IRAM 和 512 字节数据存储器(数据维持 SRAM) ❑ 唤醒源：外部引脚。 ❑ 启动时间：与深度睡眠相同。 ❑ 说明：此模式下的 PIN 唤醒将会导致系统复位。
存储器维持、定时器开启	❑ 运行的功能：比存储器维持、定时器关闭下增加： • XOSC32K 或 RCOSC32K； • RTC2 和看门狗由 32KHz 驱动； ❑ 唤醒源：外部引脚、定时器或引脚上的电平比较器唤醒 TICK。 ❑ 启动时间： ❑ 由 PIN 唤醒：小于由 RCOSC16M 启动时间，即小于 100us； 由 TICK 唤醒：预启动稳压器和 XOSC16M，系统开启 RTC2 TICK 就绪，为

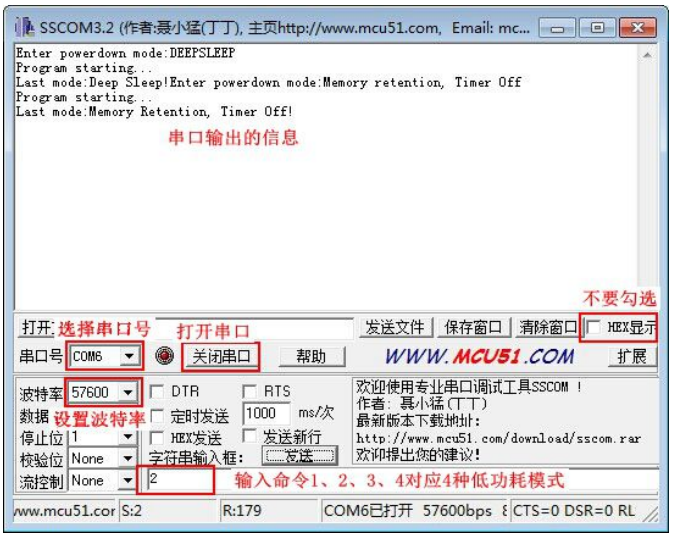
	了降低功耗，当 MCU 在被唤醒（<100us）并等待 TICK 中断时，用户可选择进入待机节能模式。当 XOSC16M 被 CLKCTRL 寄存器的位 5 和位 4 所禁止时，可以获得一个较快的预启动时间（更少的时钟周期）。 说明：此模式下的 PIN 唤醒将会导致系统复位。
寄存器维持、定时器关闭	❑ 运行的功能：比寄存器维持、定时器开启模式下增加： • 所有寄存器； • 其他数据存储器(SRAM) ❑ 唤醒源：如存储器维持、定时器开启。 ❑ 启动时间：同存储器维持、定时器开启模式。 ❑ 说明：唤醒不会导致系统复位（唤醒后，将从当前指令重新开始执行）。
寄存器维持、定时器关闭	❑ 运行的功能：比寄存器维持、定时器关闭模式下增加： 可选 XOSC16M； ❑ 唤醒源：同存储器维持、定时器开启模式。 ❑ 启动时间：同寄存器维持、定时器关闭模式。 如果由 TICK 唤醒，则当 XOSC16M 被 CLKCTRL 寄存器的位 5 和位 4 所禁止或 XOSC16M 在寄存器维持模式时（CLKCTRL 位 7），可以获得一个较快的预启动时间。如果 XOSC16M 运行前进入掉电模式，则快启动模式将不会被使用（这可以查询 CLKLFCTRL 位 3） ❑ 说明：唤醒不会导致系统复位（唤醒后，将从当前指令重新开始执行）。
待机	❑ 运行的功能：比寄存器维持模式下增加： • 程序和数据； • VREG • XOSC16M • 其他数据存储器(SRAM) ❑ 唤醒源：比寄存器维持模式下增加： • RFIRQ 和 MISCIRQ 中断，模拟比较器在此模式下不支持。 ❑ 启动时间：~ 100 ns。 ❑ 说明：处理器在待机，即时钟停止，I/O 功能有效。
	❑ 运行的功能：所有功能。 ❑ 唤醒源：~ ❑ 启动时间：~ ❑ 说明：处理器处于运行状态。

4. 实验步骤

- 在 Keil uVision4 中打开工程“power management.uvproj”工程；
- 编译工程，注意查看编译输出栏，观察编译的结果，如果有错误，修改程序，直到编译成功为止；



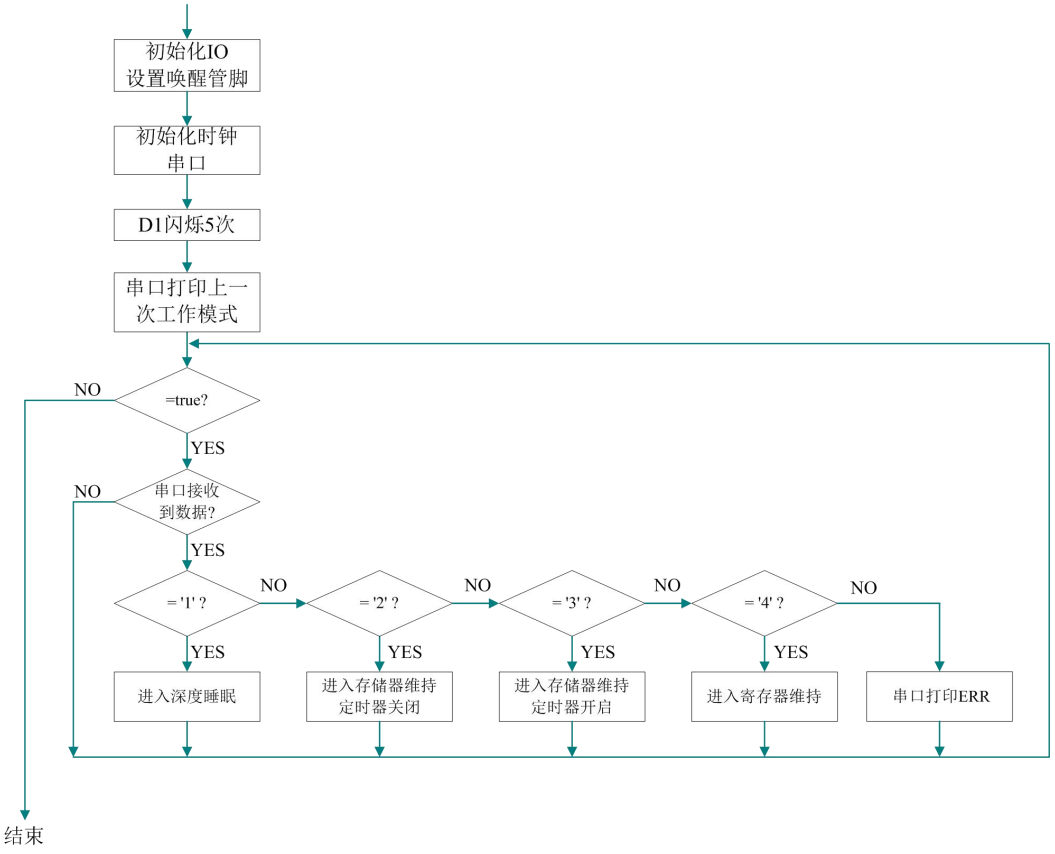
- 将编译生成的 HEX 文件“power management.hex” (该文件位于工程目录下的“Object”文件夹中)通过编程器下载到开发板中运行。
- 打开串口调试助手，选择串口号，设置波特率为 57600，打开串口，注意不要勾选“HEX 显示”。



- 输入字符 1，点击发送，让 NRF24LE1 进入深度睡眠模式，此时应观察到开发板上的指示灯停止闪烁。按下 S1 按键，唤醒 NRF24LE1，此时，会观察到 NRF24LE1 复位，开发板上的 D1 指示灯先闪烁 5 次，程序启动，之后 D2 一直闪烁，程序正常运行。同时，串口会打印出相关信息。
- 依次输入字符 2、3、4，待 NRF24LE1 进入相应的低功耗模式后，按下 S1 按键唤醒，观察开发板的运行情况和串口输出的信息。

5. 实验程序

5.1. 程序流程



5.2. 程序清单

```
#define   D1      P00   //开发板上的指示灯 D1
#define   D2      P01   //开发板上的指示灯 D2
#define   WDSV_10S      1280   //看门狗超时时间：10S

#define   DEEPSLEEP             1    //深度睡眠
#define   MEMRET_TIMEOFF        2    //存储器维持、定时器关闭
#define   MEMRET_TIMEON         3    //存储器维持、定时器开启
#define   REGRET                4    //寄存器维持


/*******************************************************************************
 *描   述 : 配置 IO P0.0 和 P0.1 为输出，驱动 LED。P03 输出: UART TXD，P04:输入 UART
            RXD
 *入   参 : 无
 *返回值 : 无
*******************************************************************************/
void IO_Init(void)
```

```
{
   P0DIR &= ~0x03;        //配置 P0.0 和 P0.1 为输出
   P0DIR &= ~0x08;     //P03:输出  UART TXD
   P0DIR |= 0x10;        //P04:输入  UART RXD
   D1 = 1;     //设置 D1 初始状态为熄灭
   D2 = 1;     //设置 D2 初始状态为熄灭
}
/*******************************************************************************
 *描   述 : 初始化时钟
 *入   参 : 无
 *返回值 : 无
*******************************************************************************/
void ClockInit(void)
{
   hal_rtc_start(false);   //关闭 32.768KHz 时钟
   hal_clklf_set_source(HAL_CLKLF_RCOSC32K); //32.768KHz 的时钟源为内部 RC

   hal_rtc_start(true);     //启动 32.768KHz 时钟

   while((CLKLFCTRL&0x80)==0x80); //等待时钟启动完成
   while((CLKLFCTRL&0x80)!=0x80);
}
/*******************************************************************************
 *描   述 : 串口打印字符串
 *入   参 : 无
 *返回值 : 无
*******************************************************************************/
void PutString(char *s)
{
   while(*s != 0)
      hal_uart_putchar(*s++);
}
/*******************************************************************************
 *描   述 : 设置唤醒管脚
 *入   参 : 无
 *返回值 : 无
*******************************************************************************/
void SetWakeUpPin(void)
{
   OPMCON=0x04;     //开锁、低电平唤醒
   WUOPC0=0x00;     // P0 唤醒的引脚设置:无
   WUOPC1=0x04;     // P1 唤醒的引脚设置:P12
```

```
    P1DIR|=0x04;        //P12 初始化为输入 I/O
    P12     =1;         // P12 初始化为高电平
}

/*******************************************************************************
 *描   述 : 主函数
 *入   参 : 无
 *返回值 : 无
*******************************************************************************/
void main(void)
{
    uint8_t i,ReceDat;
    uint32_t LoopCount = 0;

    IO_Init();          //初始化 IO
    SetWakeUpPin(); //设置唤醒 PIN
    mcu_init();         //初始化时钟

    hal_uart_init(UART_BAUD_57K6); // 初始化 UART，波特率 57600
    while(hal_clk_get_16m_source() != HAL_CLK_XOSC16M) // 等待时钟稳定
    ;
    EA = 1;             // 开启全局中断

    for(i=0;i<10;i++)   //D1 闪烁 5 次，表示程序启动
    {
        D1 = ~D1;
        delay_ms(120);
    }

    PutString("Program starting...\n");
    delay_ms(100);
    GetPrintLastPWM(); //获取上一次低功耗模式并通过串口打印
    delay_ms(100);

    while(1)
    {
        LoopCount++;
        if(LoopCount == 10000)
        {
            D2 = ~D2;
            LoopCount = 0;
        }
```

```
    if( hal_uart_chars_available() )     //串口接收到数据
    {
        ReceDat = hal_uart_getchar(); //读取数据

        switch(ReceDat)
        {
          case '1':   //进入深度睡眠模式
            PutString("Enter powerdown mode:DEEPSLEEP\n");
            delay_ms(20);
            SetPowrDownMode(DEEPSLEEP);
            break;
          case '2':   //进入存储器维持、定时器关闭模式
            PutString("Enter powerdown mode:Memory retention, Timer Off\n");
            delay_ms(20);
            SetPowrDownMode(MEMRET_TIMEOFF);
            break;
          case '3':   //进入存储器维持、定时器开启模式
            PutString("Enter powerdown mode:Memory retention, Timer On\n");
            delay_ms(20);
            SetPowrDownMode(MEMRET_TIMEON);
            break;
          case '4':   //进入寄存器维持模式
            PutString("Enter powerdown mode:Register retention\n");
            delay_ms(20);
            SetPowrDownMode(REGRET);
            break;
          default:   //输入命令错误
            PutString("ERR\n");
            break;
        }
    }
  }
}
```