STM32 串口实验

本次实验采用 STM32F103 系列芯片 USART 串口的异步功能，实现两个设备之间的通讯，具体表现为：主机通过按键控制从机的 LED 灯亮灭。

串口设置的一般步骤可以总结为如下几个步骤：

1) 串口时钟使能和 GPIO 时钟使能

RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1)；

2) GPIO 端口模式设置

根据数据手册得：配置全双工的串口 1，需要将 TX(PA9)管脚需要配置为推挽复用输出，RX(PA10)管脚配置为浮空输入或者带上拉输入；

3) 开启中断并且初始化 NVIC 中断控制器设置优先级

配置 NVIC_InitTypeDef 结构体变量：NVIC_IRQChannel、NVIC_IRQChannel、NVIC_IRQChannelSubPriority 和 NVIC_IRQChannelCmd

具体设置在前期报告中有具体阐述，此处不在赘述。

4) 串口参数初始化

配置 USART_InitTypeDef 结构体变量：USART_BaudRate、USART_WordLength、USART_StopBits、USART_Parity、USART_HardwareFlowControl、USART_Mode

此处串口数据格式设置为：串口波特率：115200、8 位数据、1 位停止位、无奇偶校验位、无硬件数据流控制、收发模式共同使能。

5) 使能串口功能和串口接收中断

USART_Cmd(USART1, ENABLE);

USART_ITConfig(USART1, USART_IT_RXNE, ENABLE);

6) 编写中断处理函数（开启串口接收中断）

USART1_IRQHandle；

通过 USART_GetITStatus(USART1, USART_IT_RXNE)函数判断中断源。

RX 端 usart.c 代码

```
1.  void USART1_IRQHandler(void)                //串口 1 中断服务程序
2.  {
3.      u8 Res;
4.      if(USART_GetITStatus(USART1, USART_IT_RXNE) != RESET)
5.      //接收中断(接收到的数据必须是 0x0d 0x0a 结尾，0x0d 回车键  0x0a 换行符)
6.      {
7.          Res =USART_ReceiveData(USART1);     //读取接收到的数据
8.          if((USART_RX_STA&0x8000)==0)        //接受未完成
9.          {
```

```
        if(USART_RX_STA&0x4000)         //表示已接收到了 0x0d
        {
            if(Res!=0x0a)               //判断 0x0d 紧跟后续不是 0x0a
                USART_RX_STA=0;   //如果不是则判定为结束格式错误,重新开始
            else
                USART_RX_STA|=0x8000;    //如果是，则判定为接收完成
        }
        else
        {
            if(Res==0x0d)
                USART_RX_STA|=0x4000;    //接收 0x0d 时
            else                    //0x0d 和 0x0a（结束标志）未接收到时
            {
                USART_RX_BUF[USART_RX_STA&0X3FFF]=Res ;
                USART_RX_STA++;
                if(USART_RX_STA>(USART_REC_LEN-1))
                    USART_RX_STA=0;//接收数据错误,重新开始接收
            }
        }
    }
}
```

USART 支持 10 个带标志的中断源：CTS 改变、LIN 断开符检测、发送数据寄存器空、发送完成、接收数据寄存器满、检测到总线为空闲、溢出错误、帧错误、噪音错误、校验错误。

当上述任意中断源响应时都会执行 USART1_IRQHandle 函数，所以此时需要使用 USART_GetITStatus(USART1, USART_IT_RXNE)函数判断接收数据中断源。该实验使能串口的接收数据中断，即当有数据传输过来时，本设备立刻执行中断服务函数，将 USART_DR 寄存器（由两个寄存器组成的，TDR 用于发送，RDR 用于接收）中的数据保存下来。

此处还添加了判断数据传输结束的协议：在每次传输的数据结尾需附加发送一个回车键符（0x0d）和一个换行符（0x0a）表示次此数据传输结束。该结束标志的使用可使得下次数据到来时覆盖上一次保存的数据同时对于便于检测当前所保存的数据内容。

RX 端 main.c 代码：

```
if(USART_RX_STA&0x8000)   //USART_RX_STA 清 0 后不再进入接收数据判断
{
```

```
    if(USART_RX_BUF[0] == '0')
    {
        LED0 = 0;
        USART_RX_STA = 0;     //USART_RX_STA 清 0，USART_RX_BUF[0]不变
    }
    if(USART_RX_BUF[0] == '1')
    {
        LED0 = 1;
        USART_RX_STA = 0;
    }
}
```

USART_RX_STA 变量用于保存是否接收到数据传输结束标志。使用 USART_RX_STA&0x8000 的结果判断当前传输数据帧是否结束，当为 1 时，可进行所保存接受到的数据的验证。 USART_RX_STA = 0 表示不再进行接收数据的判断，同时下次数据到来时将覆盖上次已接收的数据。该方法可避免在直接判断接收数据的相关位但是下次数据还未到来的这时间段内条件一直为真，连续执行 if 程序块内代码的情况。

TX 端代码：

```
u8 zero[]= {0x30,0x0d,0x0a};
u8 one[] = {0x31,0x0d,0x0a};

key=KEY_Scan(0);    //得到键值
if(key)
{
    switch(key)
    {
        case KEY1_PRES: //控制 LED1 翻转
            for(t=0;t<3;t++)
            {
                USART_SendData(USART1, zero[t]);//向串口 1 发送数据
                while(USART_GetFlagStatus(USART1,USART_FLAG_TC)!=SET);
                //等待发送结束
            }
            break;
        case KEY0_PRES: //同时控制 LED0,LED1 翻转
            for(t=0;t<3;t++)
            {
```

```
            USART_SendData(USART1, one[t]);//向串口 1 发送数据
            while(USART_GetFlagStatus(USART1,USART_FLAG_TC)!=SET);
            //等待发送结束
        }
        break;
    }
}

```

根据 ASCII 码存储格式，0x30 为字符‘ 0’ ，0x31 为字符‘ 1’ ，0x0d 为回车键符、0x0a 为换行符。

使用 USART_SendData 函数发送数组，接收端可判断 USART_RX_BUF 接收数据数组的第零位进行条件选择。

根据数据手册可得，USART_SR 寄存器的 TC 位置 1 时表示发送完成，因此可用 USART_GetFlagStatus(USART1,USART_FLAG_TC)函数判断当前数据的传输是否完成，等待此处传输结束后再执行其他操作。

SEP8000 验证

- 测试条件和目的：

在不使用 PLL 的情况下，验证从 Normal 模式进入 Sleep1 模式流程和从 Sleep1 模式退出至 Normal 模式流程的正确性。

- 测试步骤：

1、设置 PMU_RCCFG、PMU_CRYCFG 寄存器，分别配置 32KHz 和 20MHz 时钟源的稳定时间。用户手册中指出，Sleep1 模式将系统时钟切换为 32KHz 晶振，所以在进入 Sleep1 模式之前需开启 RC 振荡器。同时在不使用 PLL 的前提下，系统时钟默认选择 20MHz。

2、不使用 PLL，配置 PMCR 寄存器不使能 PLL。

3、配置 PMU_MaskWakeUp 寄存器为 0x02，屏蔽 GPIO 唤醒。

4、Sleep1 模式支持 GPIO 和 Sleeptimer 唤醒，此次针对于 SleepTimer 唤醒进行验证，配置 PMU_SleepTimerSetValue 寄存器，设置 SleepTimer 定时时间。

5、配置 PMU_SleepTimerCfg 寄存器使能 SleepTimer，并设置 SleepTimer 分频系数。

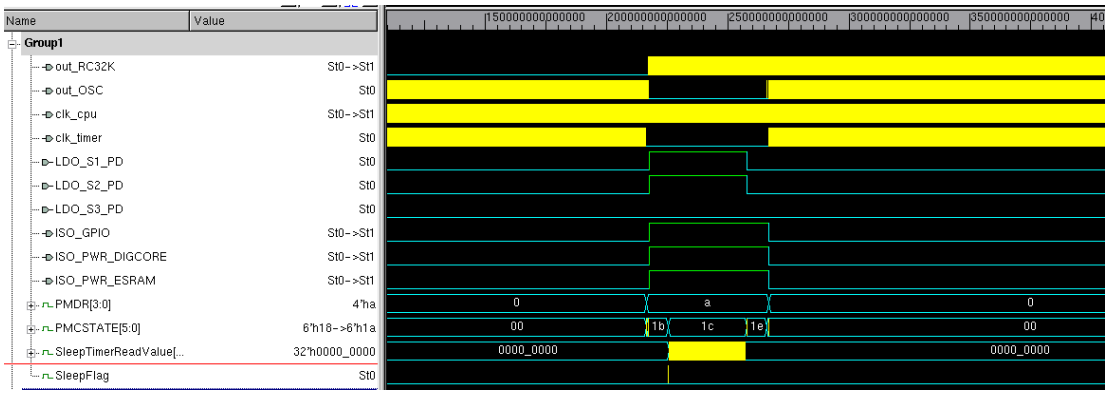
6、配置 PMU_PWROFFCFG 和 PMU_PWRONCFG 寄存器 ，设置进入低功耗模式时，数字核心区的断电等待时间和从低功耗模式下退出时，数字核心区的上电稳定的等待时间。

7、配置 PMDR 寄存器，进入 Sleep1 模式；

8、SleepTimer 计时结束后，Sleep1 模式退出。在不使用 PLL 情况下，系统时钟为 20M 时钟，程序从头重新开始执行，配置 PMU_Int 寄存器为 0x7，写 1 清除所有中断信号；读 PMU_WakeUpRecord 寄存器，可验证唤醒源是否为 SleepTimer；

- 测试结果

根据测试总图可以直观的看到在 Sleep1 模式期间系统时钟的切换和关键信号的变化。

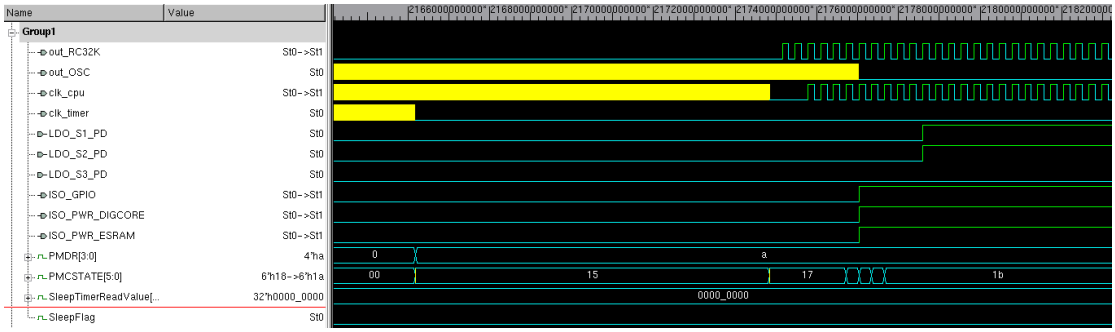


Normal-Sleep1-Normal 模式转换信号总图

1、进入 Sleep1 模式期间信号变化

在进入 Sleep1 模式期间，会关闭所有外设的时钟的门控。此处以 clk_timer 信号为例，在进入 Normal2Sleep1 模式过度期间后，受 PMU 的影响，时钟信号保持为 0。PMCSTATE 信号为 0x15 时，根据状态机描述，表示此时状态为 WaitRCReady，

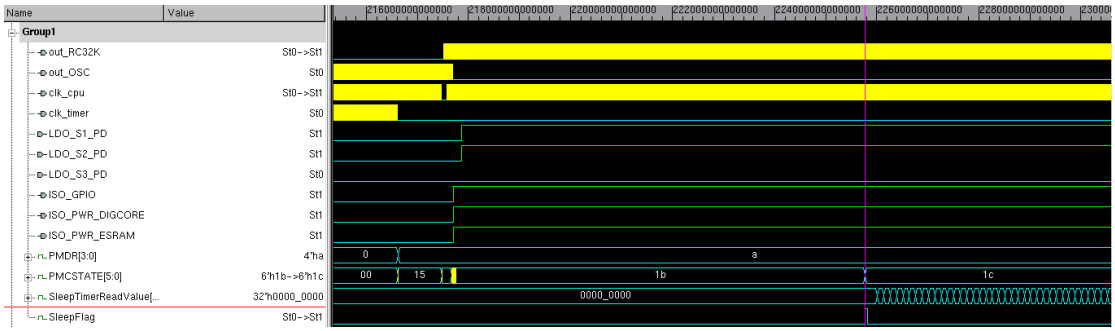
等待外部 RC32KHz 时钟稳定后，clk_cpu 将切换为 32KHz 时钟。系统时钟切换为 32KHz 后关闭 OSC20M 时钟，同时 LDO 控制信号与 ISO 信号变化正常。



Normal 模式进入 Sleep1 模式期间信号图

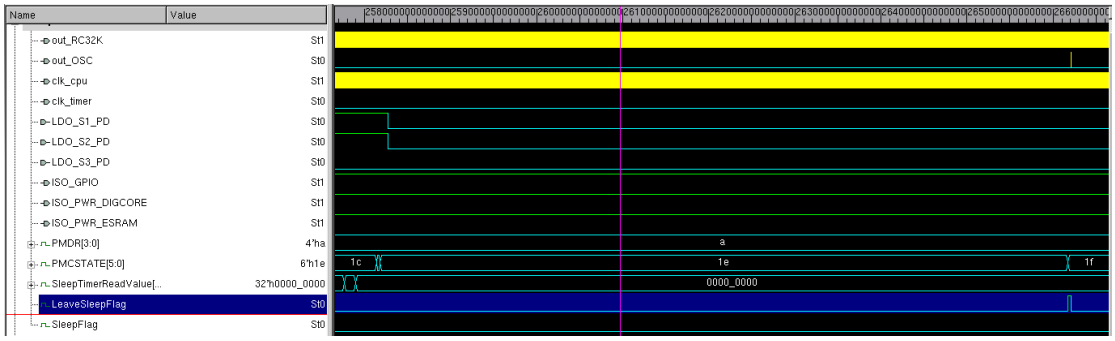
2、 进入 Sleep1 模式后信号变化

PMCSTATE 信号为 0x1B 时，状态为 No2Sl_PowerOff，等待系统掉电计时（由用户配置 PMU_PWROFFCFG 寄存器设定）结束后正式进入 Sleep1 模式。PMCSTATE 信号为 0x1C 时，状态为 Sleep。进入 Sleep 模式后，SleepFlag 信号变化正常，SleepTimer 开始计数。



SleepFlag 和 SleepTimer 信号变化图

当 SleepTimer 计数结束后 LeaveSleepFlag 和 LDO 信号变化正常，进入退出 Sleep1 模式流程。

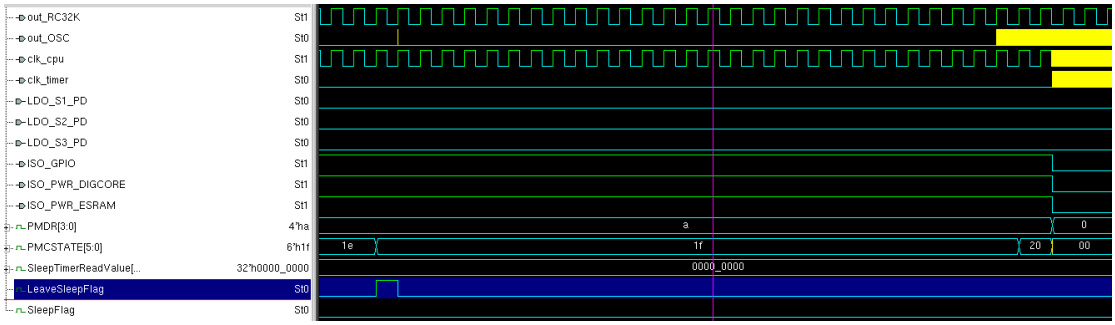


LeaveSleepFlag 和 SleepTimer 信号变化图

3、 退出 Sleep1 模式期间信号变化

PMCSTATE 信号为 0x1E 时，状态为 Sl2No_PowerOn。在等待系统上电计时（由用户配置 PMU_PWRONCFG 寄存器设定）结束后重启 OSC20M 时钟源，并将系统时钟切

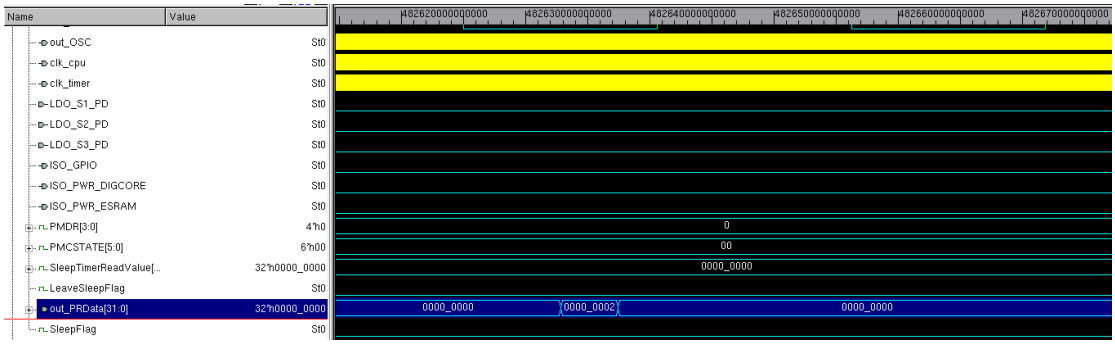
换回 20M 时钟。同时 ISO 信号变化正常，并在回到 Normal 模式后将 PMDR[3:0]置 0 表示当前状态为 Normal 模式。



Sleep1 模式返回 Normal 模式期间信号图

4、 验证唤醒源

根据用户手册，从低功耗模式被唤醒返回 Normal 模式后，内核从第一条指令开始执行。所以在被唤醒之后通过读取 PMU_WakeUpRecord 寄存器的值可验证唤醒源，此时 out_PRData 信号为 0x02，对比用户手册可得唤醒源为 SleepTimer，与验证相符。



读取 PMU_WakeUpRecord 寄存器信号