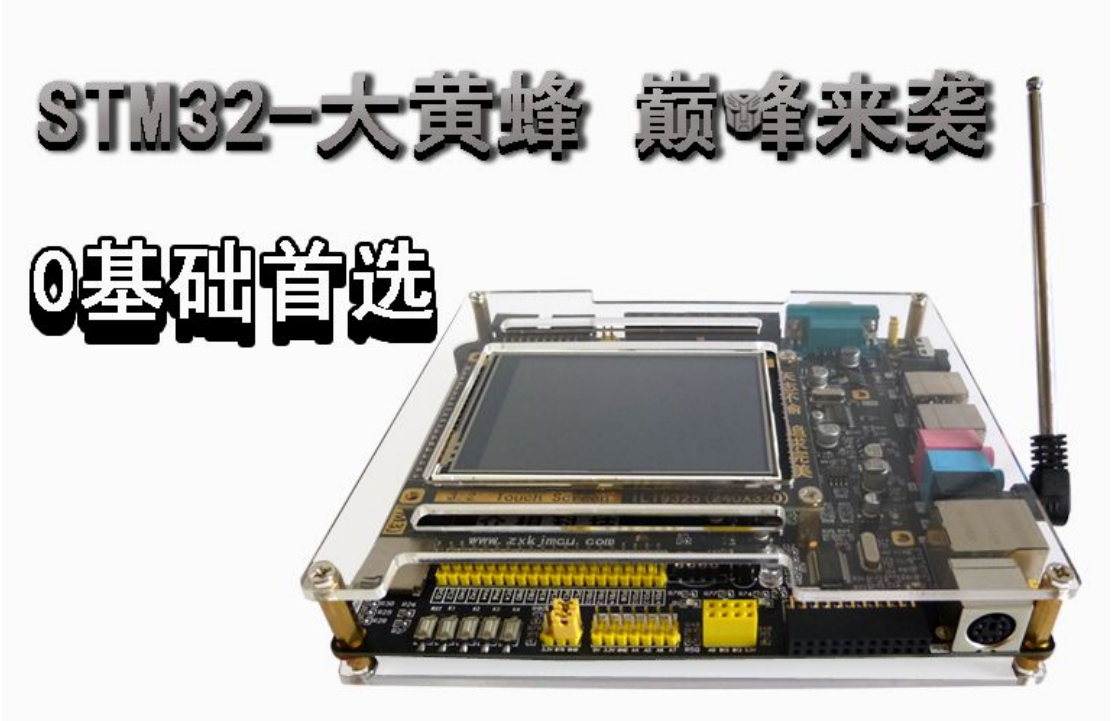
学 ARM 从 STM32 开始

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

4.14 STM32 RTC 时钟和 BKP 的工作原理及程序设计

4.14.1 概述

4.14.1.1 RTC 概念

RTC：RTC 实时时钟是一个独立的定时器。RTC 模块拥有一组连续计数的计数器，在相应软件配置下，可提供时钟日历的功能。修改计数器的值可以重新设置系统当前的时间和日期。 RTC 模块和时钟配置系统(RCC_BDCR 寄存器)处于后备区域，即在系统复位或从待机模式唤醒后，RTC 的设置和时间维持不变。 系统复位后，对后备寄存器和 RTC 的访问被禁止，这是为了防止对后备区域(BKP)的意外写操作。

执行以下操作将使能对后备寄存器和 RTC 的访问：

● 设置寄存器 RCC_APB1ENR 的 PWREN 和 BKPEN 位，使能电源和后备接口时钟。

● 设置寄存器 PWR_CR 的 DBP 位，使能对后备寄存器和 RTC 的访问。

4.14.1.2 BKP 概念

BKP：备份寄存器是 42 个 16 位的寄存器，可用来存储 84 个字节的用户应用程序数据。他们处在备份域里，当 VDD 电源被切断，他们仍然由 VBAT 维持供电。当系统在待机模式下被唤醒，或系统复位或电源复位时，他们也不会被复位。 此外，BKP 控制寄存器用来管理侵入检测和 RTC 校准功能。 复位后，对备份寄存器和 RTC 的访问被禁止，并且备份域被保护以防止可能存在的意外的写操作。

执行以下操作可以使能对备份寄存器和 RTC 的访问。

● 通过设置寄存器 RCC_APB1ENR 的 PWREN 和 BKPEN 位来打开电源和后备接口的时钟

● 电源控制寄存器(PWR_CR)的 DBP 位来使能对后备寄存器和 RTC 的访问。

4.14.2 STM32 RTC 时钟和 BKP 的主要特征

4.14.2.1 RTC 时钟主要特性

● 可编程的预分频系数：分频系数最高为 220；

● 32 位的可编程计数器，可用于较长时间段的测量；

● 2 个分离的时钟：用于 APB1 接口的 PCLK1 和 RTC 时钟(RTC 时钟的频率必须小于 PCLK1 时钟频率的四分之一以上)；

● 可以选择以下三种 RTC 的时钟源：

— HSE 时钟除以 128；

— LSE 振荡器时钟；

— LSI 振荡器时钟；

● 2 个独立的复位类型：

— APB1 接口由系统复位；

— RTC 核心(预分频器、闹钟、计数器和分频器)只能由后备域复位；

● 3 个专门的可屏蔽中断：

— 闹钟中断，用来产生一个软件可编程的闹钟中断。

— 秒中断，用来产生一个可编程的周期性中断信号(最长可达 1 秒)。

— 溢出中断，指示内部可编程计数器溢出并回转为 0 的状态；

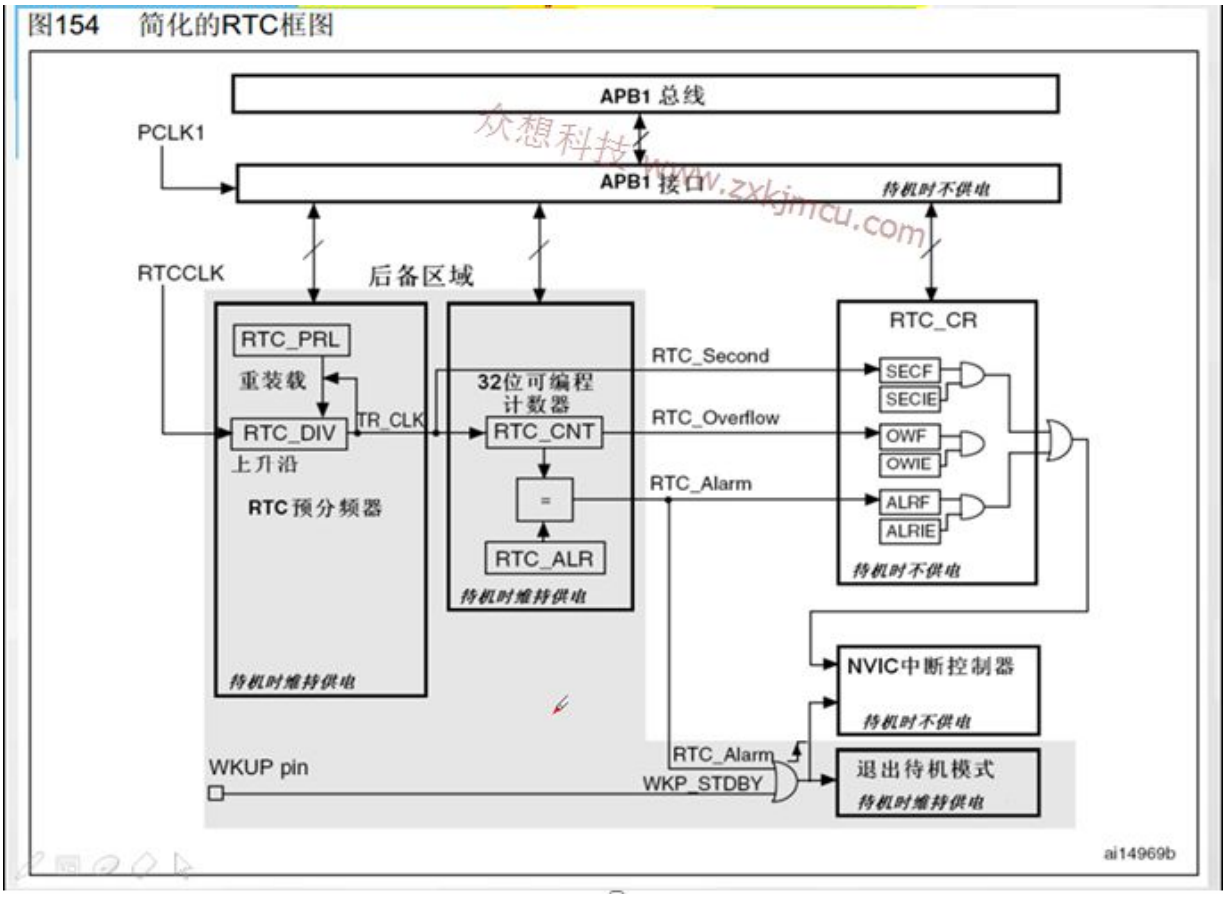
4.14.2.2 BKP 特性

● 20 字节数据后备寄存器(中容量和小容量产品)，或 84 字节数据后备寄存器(大容量和互联型产品)；

● 用来管理防侵入检测并具有中断功能的状态/控制寄存器 ；

● 用来存储 RTC 校验值的校验寄存器；

● 在 PC13 引脚(当该引脚不用于侵入检测时)上输出 RTC 校准时钟，RTC 闹钟脉冲或者秒脉冲；



4.14.3 STM32 读 RTC 寄存器

RTC 核完全独立于 RTC APB1 接口。 软件通过 APB1 接口访问 RTC 的预分频值、计数器值和闹钟值。但是，相关的可读寄存器只在与 RTC APB1 时钟进行重新同步的 RTC 时钟的上升沿被更新。RTC 标志也是如此的。 这意味着，如果 APB1 接口曾经被关闭，而读操作又是在刚刚重新开启 APB1 之后，

则在第一次的内部寄存器更新之前，从 APB1 上读出的 RTC 寄存器数值可能被破坏了(通常读到 0)。下述几种情况下能够发生这种情形：

- 发生系统复位或电源复位；
- 系统刚从待机模式唤醒(参见第 4.3 节：低功耗模式)；
- 系统刚从停机模式唤醒(参见第 4.3 节：低功耗模式)；

所有以上情况中，APB1 接口被禁止时(复位、无时钟或断电)RTC 核仍保持运行状态。 因此，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’。

注： RTC 的 APB1 接口不受 WFI 和 WFE 等低功耗模式的影响。

4.14.4 STM32 配置 RTC 寄存器

配置 RTC 寄存器 必须设置 RTC_CRL 寄存器中的 CNF 位，使 RTC 进入配置模式后，才能写入 RTC_PRL、RTC_CNT、RTC_ALR 寄存器。 另外，对 RTC 任何寄存器的写操作，都必须在前一次写操作结束后进行。可以通过查询 RTC_CR 寄存器中的 RTOFF 状态位，判断 RTC 寄存器是否处于更新中。仅当 RTOFF 状态位是’1’时，才可以写入 RTC 寄存器。 配置过程：

1. 查询 RTOFF 位，直到 RTOFF 的值变为’1’
2. 2. 置 CNF 值为 1，进入配置模式
3. 3. 对一个或多个 RTC 寄存器进行写操作
4. 4. 清除 CNF 标志位，退出配置模式
5. 5. 查询 RTOFF，直至 RTOFF 位变为’1’以确认写操作已经完成。

仅当 CNF 标志位被清除时，写操作才能进行，这个过程至少需要 3 个

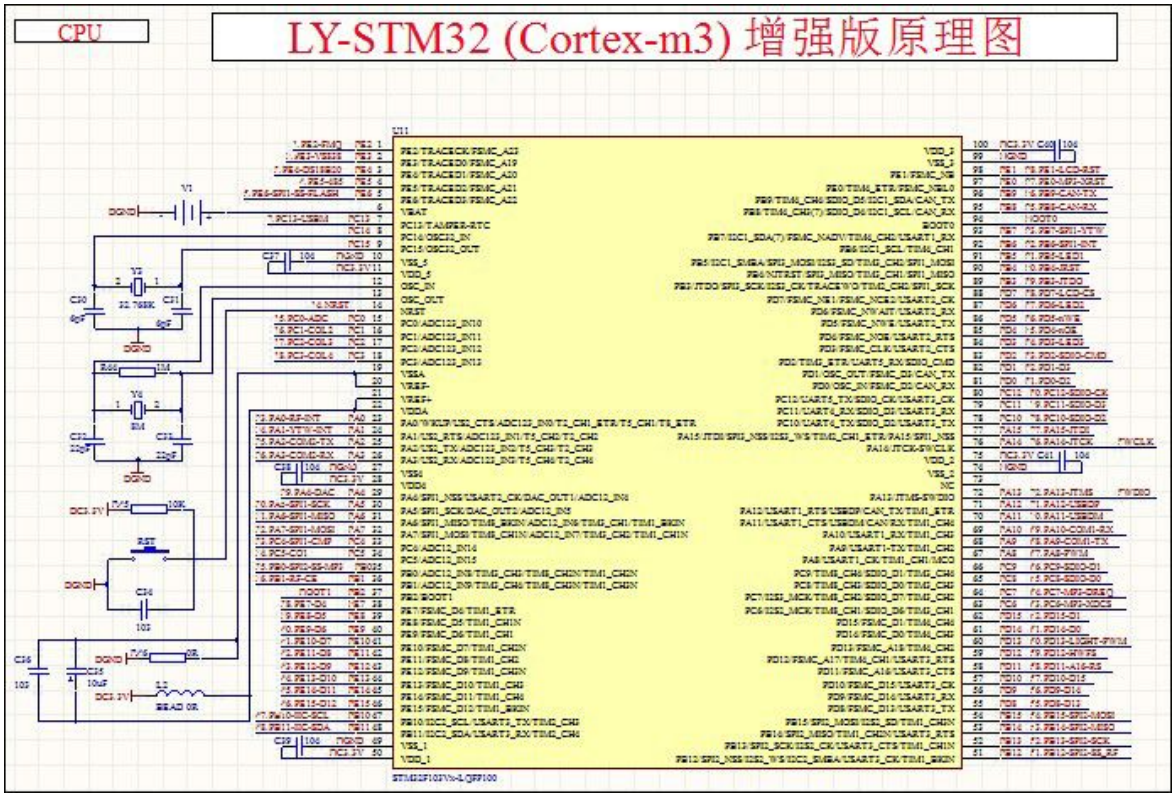
RTCCLK 周期。

4.14.5 STM32 RTC 实验目的

RTC 是一个独立功能模块，RTC 可以通过外部时钟源（纽扣电池）供电，大家可以把 RTC 看做一个独立的芯片在工作。如果 RTC 在正常工作，通过串口打印输出设置好的 RTC 电子时钟。

4.14.6 硬件设计

利用实验板上主芯片 SMT32 本身的功能模拟，通过软件设计设置 RTC 功能，通过 232 串口打印到显示器上。



4.14.7 DMA 模拟量转换软件设计

设置 RTC 功能模块，再通过主程序把时钟数据转送到串口打印输出。

4.14.7.1 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_rtc.c  // RTC 函数
stm32f10x_bkp.c
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_dac.c 库函数主要包含 DAC 模拟量转换设置，tm32f10x_rtc.c 库函数主要包含 RTC 功能模块设置，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。

4.14.7.2 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。

4.14.7.3 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_rtc.h"
#include "stm32f10x_pwr.h"
#include "stm32f10x_bkp.h"

#include "stdio.h"

//定义变量
extern u8 dt;

extern u8 tim_bz;

//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。

4.14.7.4 pbdata.c 文件里的内容是

```
#include "pbdata.h"

u8 dt=0;
u8 tim_bz=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
}
```

4.14.8 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。

4.14.9 GPIO 引脚时钟使能

```
    void RCC_Configuration(void)
    {
        SystemInit();//72m
       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

    }

    void RTC_Configuration(void)//RTC 时钟子函数

    {
       RCC_APB1PeriphClockCmd(RCC_APB1Periph_PWR,ENABLE);//打开 RTC 时钟后备
```

```
电源
        RCC_APB1PeriphClockCmd(RCC_APB1Periph_BKP,ENABLE);/打开 RTC 时钟后备域
存储区

        PWR_BackupAccessCmd(ENABLE);//允许访问 BKP 区域
        BKP_DeInit(); //复位 BKP
        RCC_LSEConfig(RCC_LSE_ON);//使能外部低速晶振 32.768K
        while(RCC_GetFlagStatus(RCC_FLAG_LSERDY)==RESET);//等待外部低速晶振就
绪
        RCC_RTCCLKConfig(RCC_RTCCLKSource_LSE);//选择 RTC 外部时钟为低速晶振
32.768K
        RCC_RTCCLKCmd(ENABLE);//使能 RTC 时钟
        RTC_WaitForSynchro();//等待 RTC 寄存器同步
        RTC_WaitForLastTask();//等待写 RTC 寄存器完成
        RTC_ITConfig(RTC_IT_SEC,ENABLE);//使能 RTC 秒中断
        RTC_WaitForLastTask();//等待写 RTC 寄存器完成
        RTC_SetPrescaler(32767);//设置预分频
        RTC_WaitForLastTask();//等待写 RTC 寄存器完成
    }
```

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；RTC 电子时钟打开，RTC 电子时钟功能模块功能设置。

“RTC_ITConfig(RTC_IT_SEC,ENABLE);//使能 RTC 秒中断”这条很重要，这条语句是打开 RTC 秒中断，这样可以一秒钟产生一次秒中断，使 CPU 产生中断，向 BKP 写入数据。

4.14.10 RTC 电子时钟判断函数

在这里我们要引入一个 RTC 初始化函数，为什么要引入这样一个函数，因为我们在第一次正式工作的时候（也就是系统从来没有启动过 RTC 功能）要初始化 RTC 功能，然后进入正常工作；当某种原因第二次启动（断电、复位等）CPU 工作的时候，就很可能不进入“void RTC_Configuration(void)//RTC 时钟初始化子函数”了，因为第一次初始化完成后，即使这个实验板断电或者复位了，RTC 也有纽扣电池供电，电子

时钟在不停的工作，所以下次实验板送电的时候就完全没有必要进入“void RTC_Configuration(void)//RTC 时钟初始化子函数”了。这样才能实现时钟的连续工作。

注：除非你想重新清除 RTC 设置好的参数，让它重新开始从初始化的时间开始工作。

```
void Clock_Init(void)
{
    if(BKP_ReadBackupRegister(BKP_DR1)!=0xA5A5)
    {
        //第一次运行  初始化设置
        //RTC 初始化
        RTC_Configuration();
        //等待写 RTC 寄存器完成
         RTC_WaitForLastTask();
        //设置时间初值
        RTC_SetCounter(0xA442);
        //等待写 RTC 寄存器完成
         RTC_WaitForLastTask();
        //写配置 0xA5A5 标志
        BKP_WriteBackupRegister(BKP_DR1,0xA5A5);//写入配置 0xA5A5 标志
    }
    else
    {
        //等待 RTC 寄存器同步
        RTC_WaitForSynchro();
        //等待写 RTC 寄存器完成
        RTC_WaitForLastTask();
        //使能 RTC 秒中断
         RTC_ITConfig(RTC_IT_SEC,ENABLE);
          //等待写 RTC 寄存器完成
        RTC_WaitForLastTask();
    }
    RCC_ClearFlag();//清除复位标志;
}
```

4.14.11 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)
  {
    if(tim_bz==1)
    {
      tim_bz=0;

      TimeData=RTC_GetCounter();
      hh= TimeData/3600;
      mm= (TimeData%3600)/60;
      ss=  TimeData%60;

      printf("时间：  %0.2d:%0.2d:%0.2d\r\n",hh,mm,ss);
    }
  }
```

4.14.12 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，先打开串口 1，和外部设备联络好，打开 STC 秒中断，来中断是把 tim_bz=1。

```
#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);
  }
```

```
}

void RTC_IRQHandler(void)
{
   if(RTC_GetITStatus(RTC_IT_SEC)!=RESET)//读取中断标志
   {
       RTC_ClearITPendingBit(RTC_IT_SEC);//清除中断标志
       tim_bz=1;//秒中断标志
   }
}
```

4.14.13 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 RTC 电子时钟数据通过 prinif 重定向打印到串口精灵上。送到显示器显示。

```
#include "pbdata.h"

void RCC_Configuration(void);
void GPIO_Configuration(void);
void NVIC_Configuration(void);
void USART_Configuration(void);
void RTC_Configuration(void);
void Clock_Init(void);

int fputc(int ch,FILE *f)
{
    USART_SendData(USART1,(u8)ch);
    while(USART_GetFlagStatus(USART1,USART_FLAG_TXE)==RESET);
    return ch;
}

int main(void)
{
   u32 TimeData=0,hh=0,mm=0,ss=0;

   RCC_Configuration(); //系统时钟初始化
   GPIO_Configuration();//端口初始化
   USART_Configuration();
   NVIC_Configuration();
   Clock_Init();
```

```
    while(1)
    {
      if(tim_bz==1)//如果中断标志为 1，顺序执行
      {
        tim_bz=0;

        TimeData=RTC_GetCounter();
        hh= TimeData/3600;
        mm= (TimeData%3600)/60;
        ss=  TimeData%60;

        printf("时间：  %0.2d:%0.2d:%0.2d\r\n",hh,mm,ss);
      }
    }
}

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 RTC_Configuration(void)
{
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_PWR,ENABLE);
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_BKP,ENABLE);

    //允许访问 BKP 区域
    PWR_BackupAccessCmd(ENABLE);
    //复位 BKP
    BKP_DeInit();
    //使能外部低速晶振 32.768K
    RCC_LSEConfig(RCC_LSE_ON);
    //等待外部低速晶振就绪
    while(RCC_GetFlagStatus(RCC_FLAG_LSERDY)==RESET);
    //选择 RTC 外部时钟为低速晶振 32.768K
    RCC_RTCCLKConfig(RCC_RTCCLKSource_LSE);
    //使能 RTC 时钟
    RCC_RTCCLKCmd(ENABLE);
    //等待 RTC 寄存器同步
    RTC_WaitForSynchro();
    //等待写 RTC 寄存器完成
```

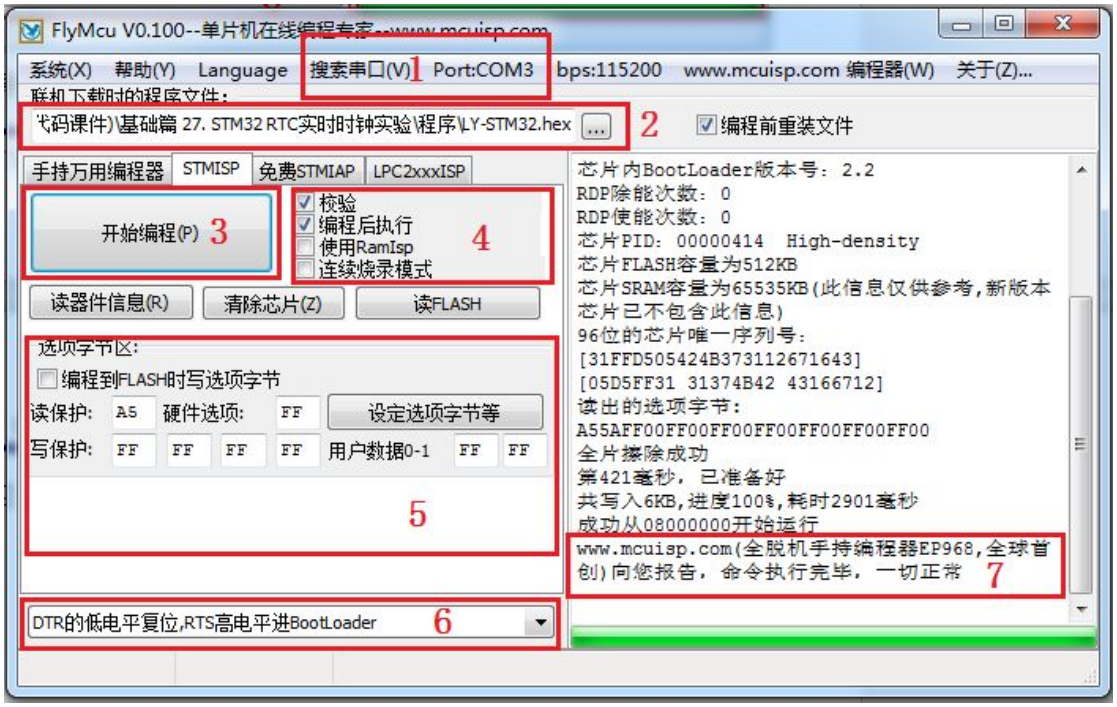
```
    RTC_WaitForLastTask();
    //使能 RTC 秒中断
    RTC_ITConfig(RTC_IT_SEC,ENABLE);
    //等待写 RTC 寄存器完成
    RTC_WaitForLastTask();
    //设置预分频
    RTC_SetPrescaler(32767);
    //等待写 RTC 寄存器完成
    RTC_WaitForLastTask();
}

void Clock_Init(void)
{
    if(BKP_ReadBackupRegister(BKP_DR1)!=0xA5A5)
    {
        //第一次运行   初始化设置
        //RTC 初始化
        RTC_Configuration();
        //等待写 RTC 寄存器完成
         RTC_WaitForLastTask();
        //设置时间初值
        RTC_SetCounter(0xA442);
        //等待写 RTC 寄存器完成
         RTC_WaitForLastTask();
        //写配置标志
        BKP_WriteBackupRegister(BKP_DR1,0xA5A5);
    }
    else
    {
        //等待 RTC 寄存器同步
        RTC_WaitForSynchro();
        //等待写 RTC 寄存器完成
        RTC_WaitForLastTask();
        //使能 RTC 秒中断
         RTC_ITConfig(RTC_IT_SEC,ENABLE);
          //等待写 RTC 寄存器完成
        RTC_WaitForLastTask();
    }

    RCC_ClearFlag();//清除复位标志;
}

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);

    NVIC_InitStructure.NVIC_IRQChannel = RTC_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 2;
    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_HardwareFlowContr
ol_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);
}
```

4.14.14 程序下载

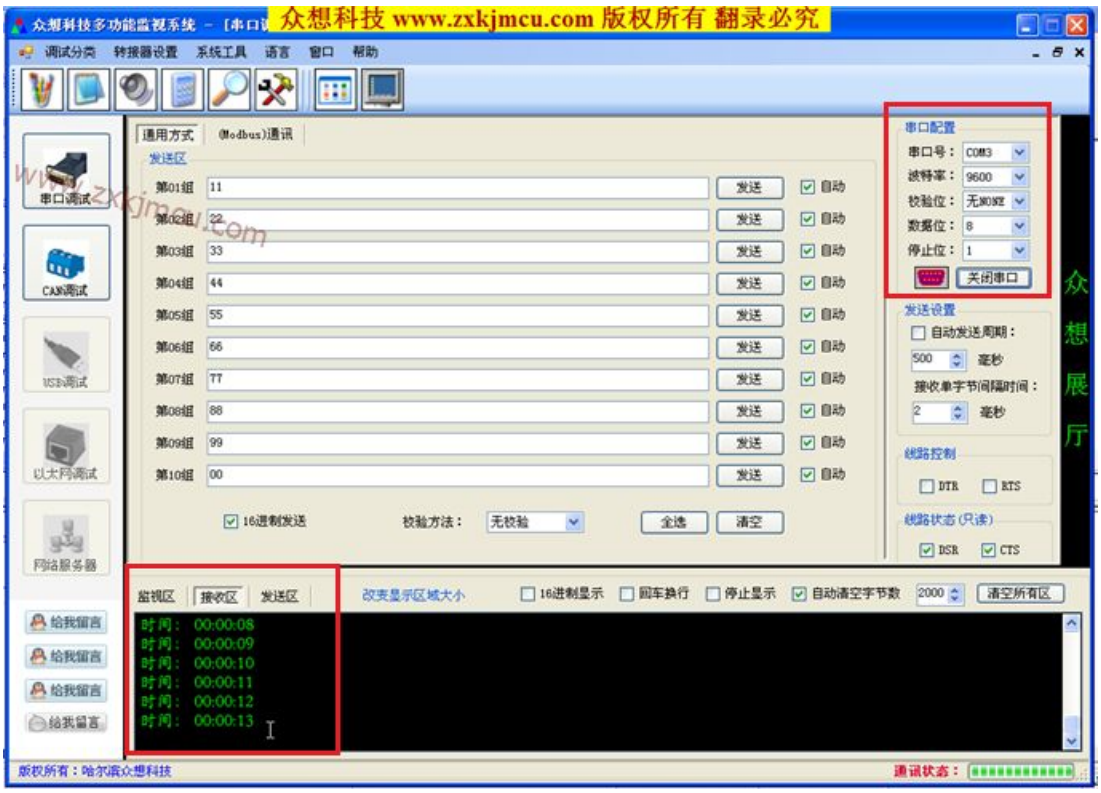
请根据下图所指向的 7 个重点区域配置。其中（1）号区域根据自己机器的实际情况选择，我的机器虚拟出来的串口号是 COM2。（2）号区域请自己选择程序所在的文件夹。（7）号区域当程序下载完后，进度条会到达最右边，并且提示一切正常。（4、5、6）号区域一定要按照上图显示的设置。当都设置好以后就可以直接点击（3）号区域的开始编程按钮下传程序了。



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 27. STM32 RTC 实时时钟实验\程序）

4.14.15 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看到 RTC 实时时钟的数据。


众想科技 www.zxkjmcu.com 版权所有 翻录必究
众想科技多功能监视系统
调试分类
转接器设置
系统工具
语言
窗口
帮助
通用方式
(Modbus)通讯
发送区
串口调试
CAN调试
USB调试
以太网调试
网络服务器
给我留言
第01组 11
第02组 22
第03组 33
第04组 44
第05组 55
第06组 66
第07组 77
第08组 88
第09组 99
第10组 00
发送
自动
16进制发送
校验方法：
无校验
全选
清空
串口配置
串口号：COM3
波特率：9600
校验位：无
数据位：8
停止位：1
关闭串口
发送设置
自动发送周期：
500
毫秒
接收单字节间隔时间：
2
线路控制
DTR
RTS
线路状态(只读)
DSR
CTS
众想展厅
监视区
接收区
发送区
改变显示区域大小
16进制显示
回车换行
停止显示
自动清空字节数
2000
清空所有区
时间：00:00:08
时间：00:00:09
时间：00:00:10
时间：00:00:11
时间：00:00:12
时间：00:00:13
版权所有：哈尔滨众想科技
通讯状态：