STM32 入门系列教程

实时时钟 *RTC* 编程

Revision 0.01

(2010-04-27)

对于单片机转 ARM 的同学来说，RTC 可能比较少接触。提到实时时钟，更经常想到的是 DS1302。当然，在 STM32 里，自己一个 CPU 已经足够，不需要 DS1302。

实际上，RTC 就只一个定时器而已，掉电之后所有信息都会丢失，因此我们需要找一个地方来存储这些信息，于是就找到了备份寄存器。因为它掉电后仍然可以通过纽扣电池供电，所以能时刻保存这些数据。我们在本期教程中将详细讲述 RTC 原理及例程，以引导大家顺利进入 RTC 的世界。

1. STM32 的 RTC 模块

RTC 模块之所以具有实时时钟功能，是因为它内部维持了一个独立的定时器，通过配置，可以让它准确地每秒钟中断一次。下面就来看以下它的组成结构。

1.1 RTC 的组成

RTC 由两个部分组成：APB1 接口部分以及 RTC 核心部分（感觉说了等于没说，因为任何模块都会有接口部分和它自己的核心部分。请注意，权威的 STM32 系列手册是这么说的 ☹ ）。笔者猜想原因可能是 STM32 所有的外设默认时钟无效，使用某个外设时，再开启时钟，用这样的方式来降低功耗。这里的 RTC，APB1 接口由 APB1 总线时钟来驱动。为了突出时钟吧？不过据说 APB1 接口部分还包括一组 16 位寄存器。

RTC 核心部分又分为预分频模块和一个 32 位的可编程计数器。前者可使每个 TR_CLK 周期中 RTC 产生一个秒中断，后者可被初始化为当前系统时间。此后系统时间会按照 TR_CLK 周期进行累加，实现时钟功能。

1.2 对 RTC 的操作

我们对 RTC 的访问，是通过 APB1 接口来进行的。注意，APB1 刚被开启的时候(比如刚上电，或刚复位后)，从 APB1 上读出来的 RTC 寄存器的第一个值有可能是被破坏了的（通常读到 0）。这个不幸，STM32 是如何预防的呢？我们在程序中，会先等待 RTC_CRL 寄存器中的 RSF 位（寄存器同步标志）被硬件置 1，然后才开始读操作，这时候读出来的值就是 OK 的。

那么对 RTC 寄存器的写操作会不会有类似的情况呢？对于写操作，我们只要注意，每一次写操作，必须确保在前一次写操作完成后进行。这个“确保”，

是通过查询 RTC_CR 寄存器中的 RTOFF 状态位，判断 RTC 寄存器是否处于更新中。只有当 RTOFF 状态位是 1，才可以写 RTC 寄存器。

2. RTC 的编程

RTC 的例程，主要是设置 RTC 时钟，使得其在超级终端上显示出当前的时钟。这个时钟的显示是“不停地走”。而且掉电后，重新上电，时钟仍然在走，仍然显示当前的时间。当然，如果感兴趣，您可以让它在 LCD 上显示 —— 那就是一个名副其实的电子钟了。

编程的时候，首先要注意备份寄存器 BKP_DR1，它做了一件关键的事情：判断 RTC 是否已经被设置过。因为 RTC 跟其他计时器不同，它是使用纽扣电池单独供电工作，所以它不会每次上电或者复位都被重置。判断 RTC 是否已经被设置过，可以决定当前是否需要去设置 RTC。如果刚安装电池，第一次上电，自然需要去设置。否则的话，我们只要让它显示当前时钟即可。

当第一次使用 RTC 的时候，可以参考手册。这里对第一次配置需要做的工作进行了总结：

1、打开电源管理和备份寄存器时钟。注意，一定要打开备份寄存器的时钟。我们正是通过在备份寄存器写固定的数据来判断芯片是否第一次实用 RTC，从而在系统运行 RTC 时提示配置时钟的。

2、使能 RTC 和备份寄存器的访问(复位默认是关闭的，以防止可能存在的意外的写操作。)。

3、选择外部低速晶体为 RTC 时钟，并使能时钟。笔者当初调试 RTC 的时候，犯了一个低级错误：由于没有定义如下：

```
#define RTCClockSource_LSE
```

导致程序一直停留在这里：

```
  /* Wait till LSE is ready */
  while(RCC_GetFlagStatus(RCC_FLAG_LSERDY) == RESET)
  {
  }
```

希望大家看完本教程后，能避免这个错误。

4、使能秒中断，程序里在秒中断里置位标志位来通知主程序显示时间数据，同时在 32 位计数器到 23：59：59 时清零；

5、设置 RTC 预分频器值产生 1 秒信号计算公式 fTR_CLK = fRTCCLK/(PRL+1)，我们设置 32767 来产生秒信号。

我们再次强调：所有在对 RTC 寄存器操作之前都要判断读写操作是否完成，即内部是否有读写操作。

下面来看代码：

```
/* System Clocks Configuration */
RCC_Configuration();
/* NVIC configuration */
NVIC_Configuration();
/* Configure the GPIOs */
GPIO_Configuration();
/* Configure the USART1 */
USART_Configuration();
```

以上四个函数调用，虽然最平常不过，但是还是要引起大家的关注。特别是中断 NVIC_Configuration();以及 USART_Configuration();，希望大家仔细查阅具体的函数实现。

与本期教程有关系第一个要点，就是时钟，为避免遗漏，笔者将其代码放在第一位：

```
RCC_APB1PeriphClockCmd(RCC_APB1Periph_PWR | RCC_APB1Periph_PWR, ENABLE);
```

接着我们读取备份寄存器 BKP_DR1 中的值来判断是否是第一次上电，如果不是则直接显示时钟，否则进行时间设置。当 BKP_DR1 的值不为 0xAAAA，说明是第一次上电，此时需要对 RTC 进行初始化。注意初始化的实现函数 RTC_Configuration();，为什么那么写，请参考我们之前给出的“第一次使用 RTC 的配置工作总结”，然后进行时钟设置。注意，因为我们需要进行写操作，所以根据固件库手册，要先调用 RTC_WaitForLastTask()，等待标志位 RTOFF 被设置，保证在前一次写操作结束后才能进行。调用 RTC_SetCounter(Time_Regulate());，将计数值写入 RTC 计数器。

由于后面要通过 BKP_WriteBackupRegister()函数对 BKP_DR1 写操作，因此之前还需要进行一次 RTC_WaitForLastTask()，这样，对时间的设置就完成了。

剩下的代码，比较简单，主要是注意如下：

```
RTCCount = RTC_GetCounter();          //获得计数值并计算当前时钟
/* Compute  hours */
THH = RTCCount/3600;
/* Compute minutes */
TMM = (RTCCount % 3600)/60;
/* Compute seconds */
TSS = (RTCCount % 3600)% 60;
```

这是通过 RTC_GetCounter();函数获取计数值，然后把这个计数值分别用小时、分钟、秒来表示的过程。最后还需要调用 printf 函数把它显示出来。

以上就是整个 RTC 的过程，期待大家拍砖。如需拍砖，请直接前往 www.arm79.com 猛拍，以促进芯达 STM32 进一步改善教程，谢谢！

附：

福州芯达工作室简介

福州芯达工作室成立于 2009 年 9 月，我们专注于嵌入式产品的研发与推广，目前芯达产品涉及 ARM9 系列、STM32 系列。

芯达团队成员均硕士研究生毕业，具有一定研发实力。我们的愿景在于把福州芯达打造成国内一流的嵌入式品牌。或许我们现在做的还不够，但是我们真的努力在做，希望通过我们的努力，能够在您学习和使用芯达产品的过程中带来或多或少的帮助。

这是芯达为了配合 STM32 开发板而推出的入门系列教程。如果您在看了我们的教程后，理清了思路，我们都会倍感欣慰！让我们一起学习，共同进步，在征服嵌入式领域的道路上风雨同行！

官方网站：http://www.arm79.com/

官方淘宝：http://shop36353570.taobao.com/