ARC

实时时钟

支持	www.armrunc.com http://armrunc.taobao.com

Change History

版本号	更改描述	作者	修改时间
1.0.0	第一版	armrunc	2012-3-6

目录

第11章 实时时钟

11.1 实时时钟简介

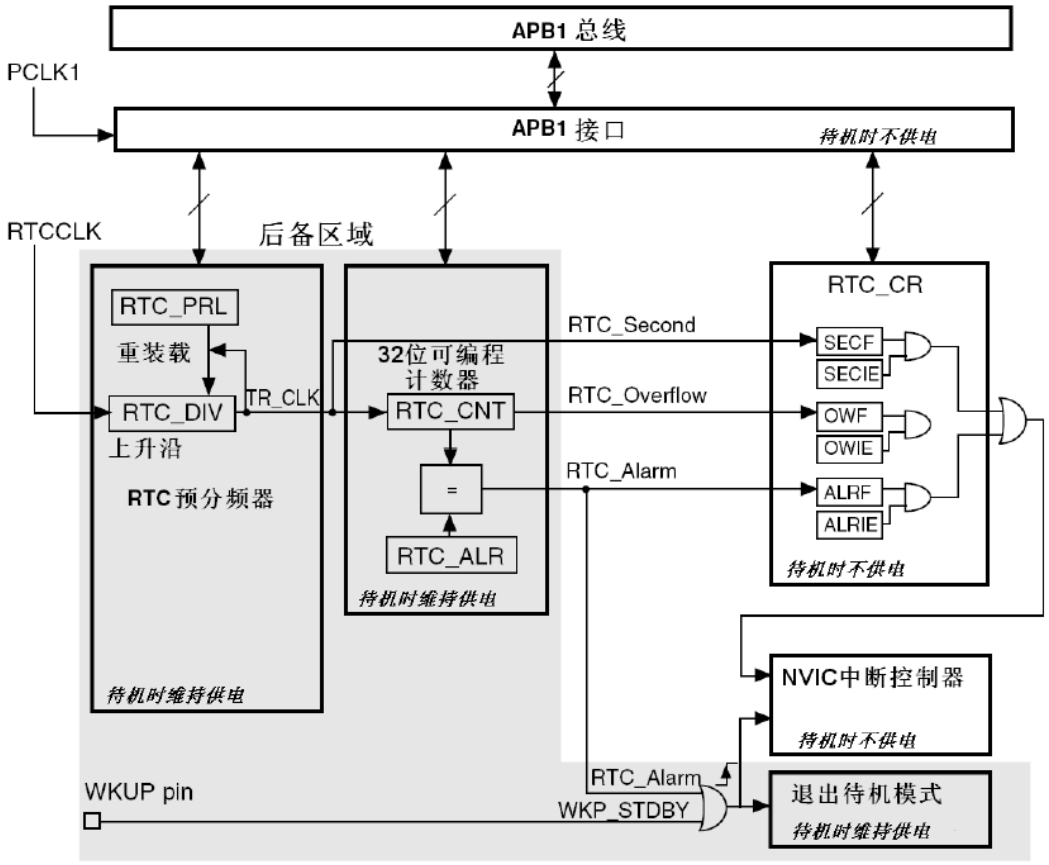
实时时钟是一个独立的定时器。RTC 模块拥有一组连续计数的计数器，在相应软件配置下，可提供时钟日历的功能。修改计数器的值可以重新设置系统当前的时间和日期。RTC 模块和时钟配置系统(RCC_BDCR 寄存器)处于后备区域，即在系统复位或从待机模式唤醒后，RTC 的设置和时间维持不变。 系统复位后，对后备寄存器和 RTC 的访问被禁止，这是为了防止对后备区域(BKP)的意外写操作。执行以下操作将使能对后备寄存器和 RTC 的访问：

- 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。

RTC 的主要特性如下：

- 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。

RTC 由两个主要部分组成(参见下图)。第一部分(APB1 接口)用来和 APB1 总线相连。此单元还包含一组 16 位寄存器，可通过 APB1 总线对其进行读写操作。APB1 接口由 APB1 总线时钟驱动，用来与 APB1 总线接口。 另一部分(RTC 核心)由一组可编程计数器组成，分成两个主要模块。第一个模块是 RTC 的预分频模块，它可编程产生最长为 1 秒的 RTC 时间基准 TR_CLK。RTC 的预分频模块包含了一个 20 位的可编程分频器(RTC 预分频器)。如果在 RTC_CR 寄存器中设置了相应的允许位，则在每个 TR_CLK 周期中 RTC 产生一个中断(秒中断)。第二个模块是一个 32 位的可编程计数器，可被初始化为当前的系统时间。系统时间按 TR_CLK 周期累加并与存储在 RTC_ALR 寄存器中的可编程时间相比较，如果 RTC_CR 控制寄存器中设置了相应允许位，比较匹配时将产生一个闹钟中断。



除了 RTC_PRL、RTC_ALR、RTC_CNT 和 RTC_DIV 寄存器外，所有的系统寄存器都由系统复位或电源复位进行异步复位。RTC_PRL，RTC_ALR，RTC_CNT 和 RTC_DIV 寄存器仅能通过备份域复位信号复位。

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
- 系统刚从停机模式唤醒

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’。

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. 置 CNF 值为 1，进入配置模式 3. 对一个或多个 RTC 寄存

器进行写操作 4. 清除 CNF 标志位，退出配置模式 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。

在每一个 RTC 核心的时钟周期中，更改 RTC 计数器之前设置 RTC 秒标志(SECF)。 在计数器到达 0x0000 之前的最后一个 RTC 时钟周期中，设置 RTC 溢出标志(OWF)。 在计数器的值到达闹钟寄存器的值加 1(RTC_ALR+1)之前的 RTC 时钟周期中，设置 RTC_Alarm 和 RTC 闹钟标志(ALRF)。对 RTC 闹钟的写操作必须使用下述过程之一与 RTC 秒标志同步：

- 使用 RTC 闹钟中断，并在中断处理程序中修改 RTC 闹钟和/或 RTC 计数器。
- 等待 RTC 控制寄存器中的 SECF 位被设置，再更改 RTC 闹钟和/或 RTC 计数器。

11.2 实时时钟应用实例 ----- 得到当前 RTC 计数值

11.2.1 实例描述

本实例非常简单，首先初始化 RTC，再初始化 SysTick 和串口，然后每隔一秒输出当前时间。该 RTC 配置为 1 秒增长 1024，如果外部时钟精确的话，每次打印出来的计数器比前面一个大 1024.

11.2.2 硬件设计

本实例用到串口输出部分，具体硬件电路图参考串口那一章节。

11.2.3 软件设计

本实例通过 BKP 寄存器 BKP_DR1 来指示 RTC 是否初始化，在第一次初始化时，初始化成功后，会将 0xA5A5 写入该寄存器。如果读出该寄存器的值不是 0xA5A5，表示 RTC 未被初始化，进行初始化，若为 0xA5A5，则等待 RTC 寄存器同步。

该实例的主要代码如下：

文件 RTC_main.c：

```
/**
  * @brief   Main program, RTC example.
  * @param   None
  * @retval None
  */
int main(void)
{
```

```
    RTC_t rtc;
    ARC_RTC_Init();
    ARC_SysTick_Init();
    ARC_COM_Init();
    USART_Cmd(USART1, ENABLE);
    while (1)
    {
        ARC_RTC_gettime(&rtc);
        ARC_SysTick_Delay(1000);
        printf("%d-%d-%d %d:%d:%d\n", rtc.year, rtc.month, rtc.mday,
rtc.hour, rtc.min, rtc.sec);
    }
}
```

文件 ARC_RTC.c：

```
#define FIRSTYEAR     2000            // start year
#define FIRSTDAY      6               // 0 = Sunday
static const uint8_t DaysInMonth[] = { 31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30,
31 };
/**
  * @brief   populates time-struct based on counter-value.
  * @param   sec, the time to be converted to RTC struct.
  * @param   *t, pointer to the RTC struct.
  * @retval None
  */
static void counter_to_struct( uint32_t sec, RTC_t *t )
{
    uint16_t day;
    uint8_t year;
    uint16_t dayofyear;
    uint8_t leap400;
    uint8_t month;

    t->sec = sec % 60;
    sec /= 60;
    t->min = sec % 60;
    sec /= 60;
    t->hour = sec % 24;
    day = (uint16_t)(sec / 24);

    t->wday = (day + FIRSTDAY) % 7;        // weekday

    year = FIRSTYEAR % 100;                    // 0..99
    leap400 = 4 - ((FIRSTYEAR - 1) / 100 & 3);   // 4, 3, 2, 1
```

```
    for(;;)
    {
        dayofyear = 365;
        if( (year & 3) == 0 )
        {
            dayofyear = 366;                    // leap year
            if( year == 0 || year == 100 || year == 200 ) // 100 year exception
            {
                if( --leap400 ) // 400 year exception
                {
                    dayofyear = 365;
                }
            }
        }
        if( day < dayofyear )
        {
            break;
        }
        day -= dayofyear;
        year++;                             // 00..136 / 99..235
    }
    t->year = year + FIRSTYEAR / 100 * 100; // + century

    if( dayofyear & 1 && day > 58 )      // no leap year and after 28.2.
    {
        day++;              // skip 29.2.
    }

    for( month = 1; day >= DaysInMonth[month-1]; month++ )
    {
        day -= DaysInMonth[month-1];
    }

    t->month = month;                   // 1..12
    t->mday = day + 1;                  // 1..31
}

/*****************************************************************************
* Function Name    : struct_to_counter
* Description      : calculates second-counter from populated time-struct
* Input            : Pointer to time-struct
* Output           : none
* Return           : counter-value (unit seconds, 0 -> 1.1.2000 00:00:00),
```

```
*   Based on code from "LalaDumm" found in the mikrocontroller.net forum.
**********************************************************************************/

/**
   * @brief    calculates second-counter from populated time-struct.
   * @param    *t, pointer to the RTC struct.
   * @retval the second that converted from the RTC struct.
   */
static uint32_t struct_to_counter( const RTC_t *t )
{
      uint8_t i;
      uint32_t result = 0;
      uint16_t idx, year;

      year = t->year;

      /* Calculate days of years before */
      result = (uint32_t)year * 365;
      if (t->year >= 1)
      {
            result += (year + 3) / 4;
            result -= (year - 1) / 100;
            result += (year - 1) / 400;
      }

      /* Start with 2000 a.d. */
      result -= 730485UL;

      /* Make month an array index */
      idx = t->month - 1;

      /* Loop thru each month, adding the days */
      for (i = 0; i < idx; i++)
      {
            result += DaysInMonth[i];
      }

      /* Leap year? adjust February */
      if (year % 400 == 0 || (year%4 == 0 && year%100 !=0))
      {
            ;
      }
      else
      {
```

```
            if (t->month > 1)
            {
                result--;
            }
        }

        /* Add remaining days */
        result += t->mday;

        /* Convert to seconds, add all the other stuff */
        result = (result-1) * 86400L + (uint32_t)t->hour * 3600 + (uint32_t)t->min *
60 + t->sec;

        return result;
}

/**
   * @brief    Adjusts time.
   * @param    time_set, the time to be set.
   * @retval None
   */
void ARC_RTC_SetCounter(uint32_t time_set)
{
   /* Wait until last write operation on RTC registers has finished */
   RTC_WaitForLastTask();
   /* Change the current time */
   RTC_SetCounter(time_set);
   /* Wait until last write operation on RTC registers has finished */
   RTC_WaitForLastTask();
}

/**
   * @brief    sets HW-RTC with values from time-struct.
   * @param    RTC struct
   * @retval None.
   */
void ARC_RTC_settime (const RTC_t *rtc)
{
        uint32_t cnt;
        RTC_t ts;

        cnt = struct_to_counter( rtc ); // non-DST counter-value
        counter_to_struct( cnt, &ts );    // normalize struct (for weekday)
        PWR_BackupAccessCmd(ENABLE);
```

```
    ARC_RTC_SetCounter( cnt );
    PWR_BackupAccessCmd(DISABLE);
}

/**
  * @brief   get the current time.
  * @param   *rtc rtc struct of current time
  * @retval None
  */
void ARC_RTC_gettime (RTC_t *rtc)
{
    uint32_t t;

    while ( ( t = RTC_GetCounter() ) != RTC_GetCounter() )
    {
    }
    counter_to_struct( t, rtc ); // get non DST time
}


/**
  * @brief   Configures the RTC, 1 s counter will increas 1024.
  * @param   None
  * @retval None
  */
void ARC_RTC_Configuration(void)
{
    /* Enable PWR and BKP clocks */
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_PWR |
RCC_APB1Periph_BKP, ENABLE);

    /* Allow access to BKP Domain */
    PWR_BackupAccessCmd(ENABLE);

    /* Reset Backup Domain */
    BKP_DeInit();

    /* Enable LSE */
    RCC_LSEConfig(RCC_LSE_ON);
    /* Wait till LSE is ready */
    while (RCC_GetFlagStatus(RCC_FLAG_LSERDY) == RESET)
    {}

    /* Select LSE as RTC Clock Source */
```

```
    RCC_RTCCLKConfig(RCC_RTCCLKSource_LSE);

    /* Enable RTC Clock */
    RCC_RTCCLKCmd(ENABLE);

    /* Wait for RTC registers synchronization */
    RTC_WaitForSynchro();

    /* Wait until last write operation on RTC registers has finished */
    RTC_WaitForLastTask();

    /* Set RTC prescaler: set RTC period to 1sec */
    RTC_SetPrescaler(32767); /* RTC period = RTCCLK/RTC_PR = (32.768
KHz)/(31+1) */

    /* Wait until last write operation on RTC registers has finished */
    RTC_WaitForLastTask();
}

/**
  * @brief  Initialize RTC, 1 / 1024 s interval.
  * @param  None
  * @retval None
  */
void ARC_RTC_Init()
{
    if (BKP_ReadBackupRegister(BKP_DR1) != 0xA5A5)
    {
        /* Backup data register value is not correct or not yet programmed
(when
        the first time the program is executed) */

        /* RTC Configuration */
        ARC_RTC_Configuration();

        ARC_RTC_SetCounter(0x0);

        BKP_WriteBackupRegister(BKP_DR1, 0xA5A5);
    }
    else
    {
        /* Wait for RTC registers synchronization */
        RTC_WaitForSynchro();
    }
```

```
    /* Clear reset flags */
    RCC_ClearFlag();
}
```