控制组STM32培训3 —— Timer

一、 Systick定时器

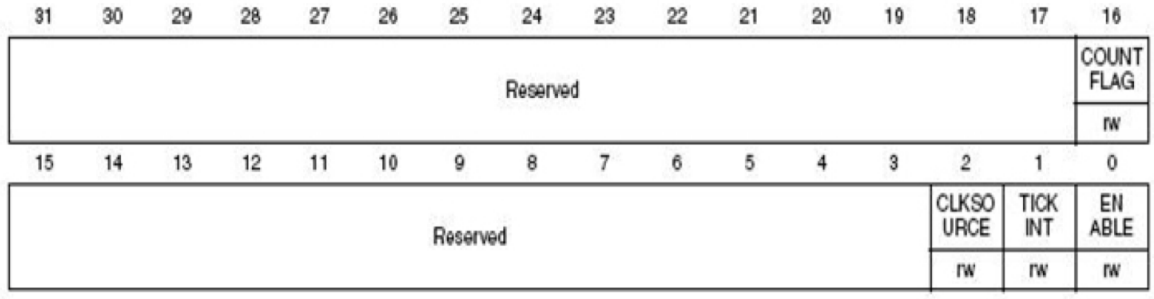
1. 简介

`Cortex-M3`处理器内部包含了一个简单的定时器。因为所有的CM3芯片都带有这个定时器，软件在不同CM3器件间的移植工作得以化简。该定时器的时钟源可以是内部时钟，或者是外部时钟。不过，STCLK的具体来源则由芯片设计者决定，因此不同产品之间的时钟频率可能会大不相同，你需要查找芯片的器件手册来决定选择什么作为时钟源。`Cortex-M3`的内核中包含一个 SysTick 时钟。SysTick 为一个 24 位递减计数器，SysTick 设定初值并使能后，每经过 1 个系统时钟周期，计数值就减 1。计数到 0 时，SysTick 计数器自动重装初值并继续计数，同时内部的 COUNTFLAG 标志会置位，触发中断 (如果中断使能情况下)。

2. Systick中寄存器

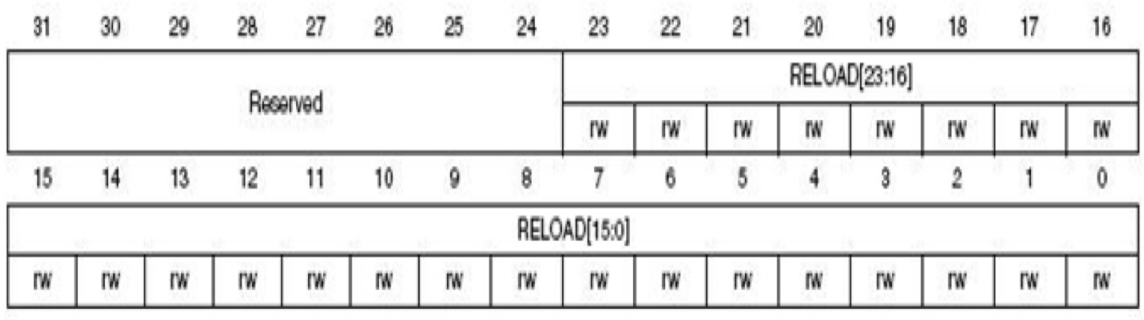
名称	地址	类型
STK_CTRL	0xE000E010	控制寄存器
STK_LOAD	0xE000E014	重载寄存器
STK_VAL	0xE000E018	当前值寄存器
STK_CALRB	0xE000E01C	校准值寄存器

2.1 STK_CTRL控制寄存器



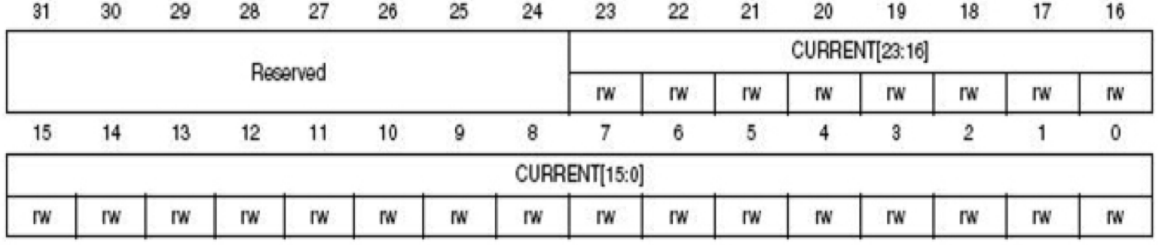
- 第0位：ENABLE，Systick 使能位
  `0`:关闭Systick功能； `1`:开启Systick功能
- 第1位：TICKINT，Systick 中断使能位
  `0`:关闭Systick中断； `1`:开启Systick中断
- 第2位：CLKSOURCE，Systick时钟源选择
  `0`:使用HCLK/8 作为Systick时钟；
  `1`:使用HCLK作为Systick时钟
- 第16位：COUNTFLAG，Systick计数比较标志，如果在上次读取本寄存器后，SysTick 已经数到了0，则该位为1。如果读取该位，该位将自动清零

2.2 STK_LOAD重载寄存器



Systick是一个递减的定时器，当定时器递减至0时，重载寄存器中的值就会被重装载，继续开始递减。STK_LOAD 重载寄存器是个24位的寄存器最大计数0xFFFFFF。

2.3 STK_VAL当前值寄存器



也是个24位的寄存器，读取时返回当前倒计数的值，写它则使之清零，同时还会清除在SysTick 控制及状态寄存器中的COUNTFLAG 标志。

3. 使用Systick实现延时功能

- Systick初始化

```
u32 fac_us;  // 延时1ns需要的滴答次数
u16 fac_ms;  // 延时1ms需要的滴答次数
void delay_init()
{
    /* 选择外部时钟  HCLK/8 */
    SysTick_CLKSourceConfig(SysTick_CLKSource_HCLK_Div8);
    /* 外部时钟为系统时钟的1/8  (8M晶振, CPU频率72M)  */
    fac_us = SystemCoreClock / 8000000;
    fac_ms = (u16)fac_us*1000;
}
```

- 延时函数

```
void delay_us(u32 nus)
{
    u32 temp;
    SysTick->LOAD = nus*fac_us; // 加载计数
    SysTick->VAL = 0x00; // 清空计数器
    SysTick->CTRL |= SysTick_CTRL_ENABLE_Msk ; // 开始倒数
    do
    {
        temp = SysTick->CTRL;
    }while((temp & 0x01) &&
           !(temp & (1 << 16)));          // 等待时间到达
    SysTick->CTRL &= ~SysTick_CTRL_ENABLE_Msk;  // 关闭计数
    SysTick->VAL = 0X00;  // 清空计数器
}
```

```
void delay_ms(u16 nms)
{
    u32 temp;
    SysTick->LOAD=(u32)nms*fac_ms;
    SysTick->VAL =0x00;
    SysTick->CTRL|=SysTick_CTRL_ENABLE_Msk ;
    do
    {
        temp=SysTick->CTRL;
    }while((temp&0x01)&&!(temp&(1<<16)));
    SysTick->CTRL&=~SysTick_CTRL_ENABLE_Msk;
    SysTick->VAL =0X00;
}
```

二、 STM32定时器

STM32中一共有11个定时器，其中2个高级控制定时器，4个通用定时器和2个基本定时器，以及2个看门狗定时器和1个系统嘀嗒定时器。其中系统嘀嗒定时器是前文中所描述的SysTick。

定时器	计数器分辨率	计数器类型	预分频系数	产生DMA请求	捕获/比较通道	互补输出
TIM1	16位	向上、向下、向上/向下	1-65536之间的任意数	可以	4	有
TIM2 TIM3 TIM4 TIM5	16位	向上、向下、向上/向下	1-65536之间的任意数	可以	4	没有
TIM6 TIM7	16位	向上	1-65536之间的任意数	可以	0	没有

1. 通用定时器主要功能

- 16位向上、向下、向上/ 向下自动装载计数器
- 16位可编程(可以实时修改)预分频器，计数器时钟频率的分频系数为1～65536之间的任意数值
- 4个独立通道：
  - 输入捕获
  - 输出比较
  - PWM生成(边缘或中间对齐模式)
  - 单脉冲模式输出

2. 时钟来源

计数器时钟可以由下列时钟源提供

- 内部时钟（CK_INT）
- 外部时钟模式1: 外部输入脚（TIx）
- 外部时钟模式2: 外部触发输入（ETR）
- 内部触发输入（ITRx）: 使用一个定时器作为另一个定时器的预分频器，如可以配置一个定时器 `TIM1` 作为另一个定时器 `TIM2` 的预分频器

3. 计数器模式

TIM2-TIM5可以由向上计数、向下计数、向上向下双向计数。向上计数模式中，计数器从0计数到自动加载值(TIMx_ARR计数器内容)，然后重新从0开始计数并且产生一个计数器溢出事件。在向下模式中，计数器从自动装入的值(TIMx_ARR)开始向下计数到0，然后从自动装入的值重新开始，并产生一个计数器向下溢出事件。而中央对齐模式（向上/向下计数）是计数器从0开始计数到自动装入的值-1，产生一个计数器溢出事件，然后向下计数到1并且产生一个计数器溢出事件；然后再从0开始重新计数。

4. 使用通用定时器实现延时功能

- TIM初始化

```
#define SYSTICKPERIOD 1e-6
#define SYSTICKFREQUENCY (1/SYSTICKPERIOD)
void delay_init()
{
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);
    TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;
    /* 寄存器重载值 */
    TIM_TimeBaseStructure.TIM_Period = 999;
    /* 预分频: 计数器+1时间为：预分频/CPU主频 */
    TIM_TimeBaseStructure.TIM_Prescaler =
                SystemCoreClock / SYSTICKFREQUENCY - 1;
    /* 向上计数 */
    TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
    TIM_TimeBaseInit(TIM2, &TIM_TimeBaseStructure);
    /* 使能TIMx 在 ARR 上的预装载寄存器 */
    TIM_ARRPreloadConfig(TIM2, ENABLE);
    /* 设置更新请求源只在计数器上溢或下溢时产生中断 */
    TIM_UpdateRequestConfig(TIM2, TIM_UpdateSource_Global);
    TIM_ClearFlag(TIM2, TIM_FLAG_Update);
}
```

- 延时函数

```
void delay_ms(u16 nms)
{
    /* 清零计数器 */
    TIM2->CNT = 0;
    TIM_Cmd(TIM2, ENABLE);

    for (int n = 0; n < nms; n++) {
        /* 等待一个延时单位结束 */
        while (TIM_GetFlagStatus(TIM2, TIM_FLAG_Update) != SET)
            TIM_ClearFlag(TIM2, TIM_FLAG_Update);
    }
    TIM_Cmd(TIM2, DISABLE);
}
```