第五讲 争分夺秒

常语人曰：“大禹圣者，乃惜寸阴，至于众人，当惜分阴。”人生当争分夺秒。

本讲学习目标：

1、了解定时器时钟体系。

2、配置 TIMx，RTC。

A：STM32 的 TIMx

在第二讲中曾提到 STM32 的时钟体系结构，当然只是简单的介绍了 RCC（复位时钟控制器）和滴答时钟的使用，而在这一讲中我们开始学习真正的时钟——通用定时器（TIMx）。当然，在这里不可能详述所有 TIM 的各个关节，但是希望能从这讲中学会简单使用 TIMx 产生定时。

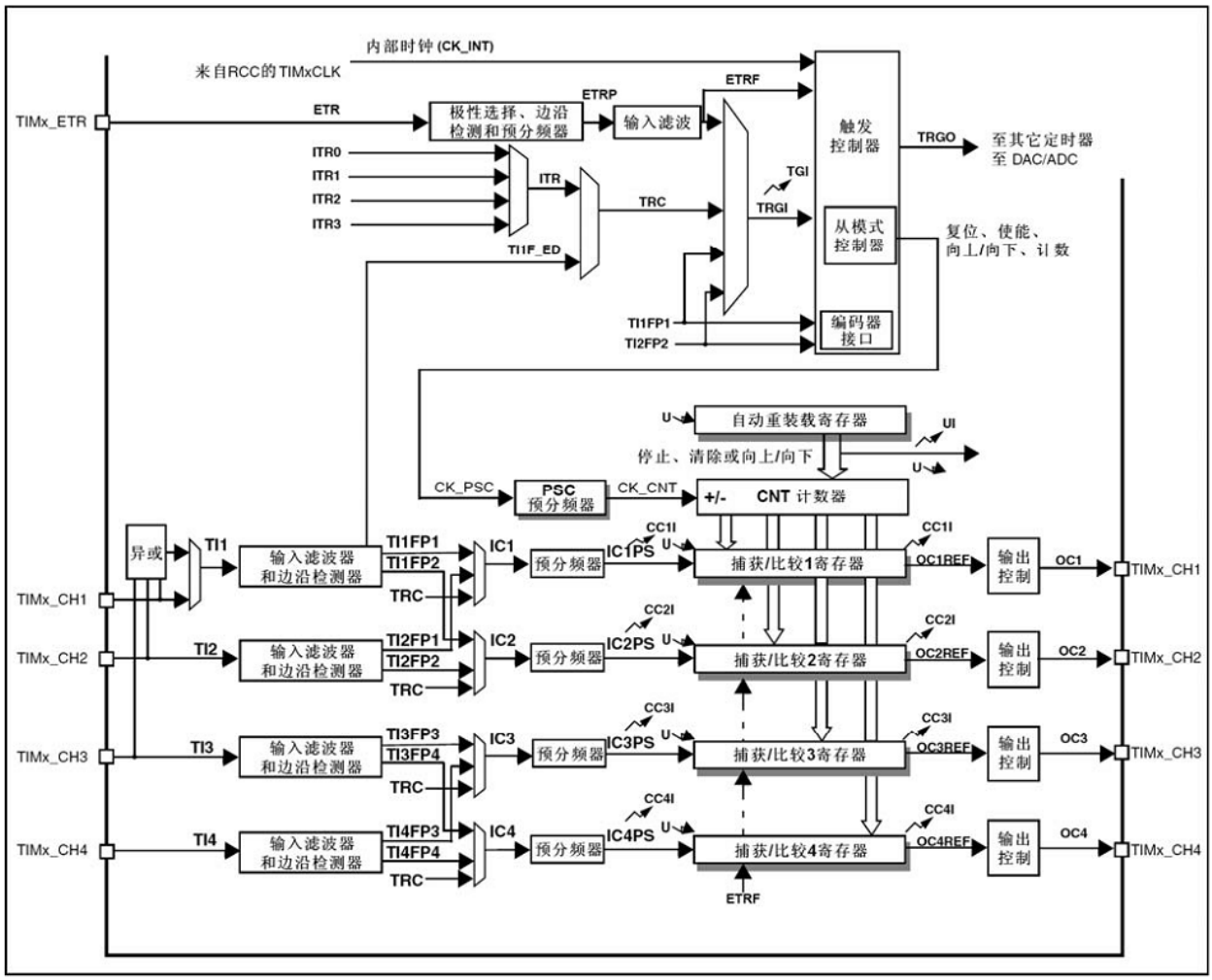
STM32 中将定时器分为了两种：通用定时器(TIMx) 以及基本定时器(TIM6 和 TIM7)和高级控制定时器(TIM1 和 TIM8) ，今天我们将要学习的是 TIMx ，也就是 STM32 的通用定时器。

这一讲的目的是利用 TIM2，建立一个秒中断。并对比而后要学习的 RTC 实时时钟。

第一件最重要的是，还是利用程序开关“stm32f10x_conf.h”开放 #include "stm32f10x_tim.h" 、#include "misc.h"。Tiim.h 可谓庞大，这与复杂多变的 STM32 时钟系统有关，更与这强大的 TIM 有关。

在 TIMx 的简介中我们可以了解到：通用定时器是一个通过可编程预分频器驱动的 16 位自动装载计数器构成的。他具有测量输入信号的脉冲长度（输入捕获）或者产生输出波形（输出比较）和 PWM 的功能。

以下是 TIMx 的框图：



注：
Reg 根据控制位的设定，在U事件时传送预加载寄存器的内容至工作寄存器
事件
中断和DMA输出

对于这张图的学习希望能从以下几个结构体开始：

```
TIM_TimeBaseInit(TIM2, &TIM_TimeBaseStructure);

TIM_OC1Init(TIM2, &TIM_OCInitStructure);
```

在开启 RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIMx, ENABLE);后，我们开始由基本时钟设置函数中的 TIM_TimeBaseStructure 开始解释图中的一些关节：

```
typedef struct
{
  uint16_t TIM_Prescaler;
  uint16_t TIM_CounterMode;
  uint16_t TIM_Period;
  uint16_t TIM_ClockDivision;
  uint8_t TIM_RepetitionCounter;
} TIM_TimeBaseInitTypeDef;
```

TIM_Prescaler ：设置了用来作为 TIMx 时钟频率除数的预分频值。它的取值必须在 0x0000 和 0xFFFF 之间。（16 位）

什么是 TIM_Prescaler

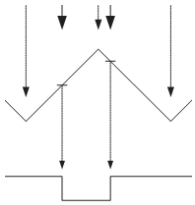
预分频器可以将计数器的时钟频率按 1 到 65536 之间的任意值分频。它是基于一个(在 TIMx_PSC 寄存器中的)16 位寄存器控制的 16 位计数器。这个控制寄存器带有缓冲器，它能够在工作时被改变。新的预分频器参数在下一次更新事件到来时被采用。

注意：预分频,就是分频系数,STM32 中,虽然 TIMx 是属于低速总线的,这条总线最高只能 36M 的速度,但芯片内部还有一个*2 的倍频器 ，用于把这个低速的 36M 倍频成 72M ， 3.0 的库中开始已经默认实现了这一步.所以我们使用的 TIMx ， 速度依旧是 72M。如果将 TIM_Prescaler 赋值 7200-1 则预分频后的效果是 0.0001S 一个周期。

如果之后设定了捕获比较值，那么除了总的计数值外，还会在一次完整的累加中，比较捕获比较值，进行动作，非常方便。

TIM_CounterMode ：选择了计数器模式。（向上、向下、中央 1、2、3）

向上向下不必赘述，而中央模式虽本讲未有使用希望能够了解：



中央 1、2、3：（笔者注：STM32 中的图不是很明显，如果没记错，ATMEL AVR 的 PWM 章节的图会相对好看点，至少有看得懂的锯齿累加的图）

CMS[1:0]：选择中央对齐模式 (Center-aligned mode selection)

00：边沿对齐模式。计数器依据方向位(DIR)向上或向下计数。
01：中央对齐模式1。计数器交替地向上和向下计数。
配置为输出的通道(TIMx_CCMRx寄存器中CCxS=00)
的输出比较中断标志位，只在计数器向下计数时被设置。
10：中央对齐模式2。计数器交替地向上和向下计数。
配置为输出的通道(TIMx_CCMRx寄存器中CCxS=00)
的输出比较中断标志位，只在计数器向上计数时被设置。
11：中央对齐模式3。计数器交替地向上和向下计数。
配置为输出的通道(TIMx_CCMRx寄存器中CCxS=00)
的输出比较中断标志位，在计数器向上和向下计数时均被设置。

注：在计数器开启时(CEN=1)，不允许从边沿对齐模式转换到中央对齐模式。

注：STM32F10x 常见应用解析中第 28 页开始讲述定时输出与 PWM，此部分写的较为完整，相比参考手册更容易理解。相关的累加图，可以对比 AVR 单片机中 PWM 模式的几张三角波、锯齿波的图（虽然容易被误解，但是不知道怎么形容好了的。）

TIM_Period ：设置了在下一个更新事件装入活动的自动重装载寄存器周期的值。它的取值必须在 0x0000 和 0xFFFF 之间。（说白了就是个基数，什么基数呢，要累加的次数罢了，一个计数值而已。）

TIM_ClockDivision ： 设置了时钟分割。

TIM_RepetitionCounter ：就是设置比较寄存器的更新速率，也就是说多少个周期才更新比较寄存器的值。（在旧版本的库中是没有的。）

```
typedef struct
{
  uint16_t TIM_OCMode;
  uint16_t TIM_OutputState;
  uint16_t TIM_OutputNState;
  uint16_t TIM_Pulse;
  uint16_t TIM_OCPolarity;
  uint16_t TIM_OCNPolarity;
  uint16_t TIM_OCIdleState;
  uint16_t TIM_OCNIdleState;
} TIM_OCInitTypeDef;
```

TIM_OCMode ：选择定时器模式。（这个很有名堂!! 要好好讲~）

TIM_Pulse ： 设置了装入捕获比较寄存器的脉冲值。（也就是说脉冲多少个的时候要动作了的。16 位 0000-FFFF）

TIM_OCPolarity ：输出极性 注：本讲暂时只需这些配置内容

TIM_ARRPreloadConfig ————使能预装载

TIM_ClearITPendingBit ————预先清除所有中断位

TIM_ITConfig ————配置中断

TIM_Cmd ————允许 TIMx 开始计数

以上这些只是对 TIMx 的配置，仅仅只是个开始，在之前所讲之中凡是中断，必须配置以及改写另外两个部分——NVIC 以及 IT。

如果仅仅利用 TIM 中断定时那么 NVIC 的配置不需要做太多事情：

```
void NVIC_Configuration(void)
{
   NVIC_InitTypeDef NVIC_InitStructure;
   NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1);
   NVIC_InitStructure.NVIC_IRQChannel = TIM3_IRQn;
   NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
   NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
   NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
   NVIC_Init(&NVIC_InitStructure);
}
```

而 IT 内具体要做这些事：

```
void TIMx_IRQHandler(void)
{
	if (TIM_GetITStatus(TIMx, TIM_IT_CC1) != RESET)
	{
		/*清空标志位*/
		TIM_ClearITPendingBit(TIMx, TIM_IT_CC1);
		/*干你想干的事情*/
	}
	else if (TIM_GetITStatus(TIMx, TIM_IT_Update) != RESET)
	{
		TIM_ClearITPendingBit(TIMx, TIM_IT_Update);
		/*干你想干的事情*/
	}
}
```

注：进入中断后查询捕获。

以下给出 TIMx 完整配置：（0.5 秒，1 秒时各在中断中查询并动作一次）

```
1  void TIM_Configuration(void)
2  {
3    TIM_TimeBaseInitTypeDef  TIM_TimeBaseStructure;
4    TIM_OCInitTypeDef  TIM_OCInitStructure;
5
6    u16 CCR1_Val = 5000;
7
8    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);
9
10   TIM_TimeBaseStructure.TIM_Period =10000;
11   TIM_TimeBaseStructure.TIM_Prescaler = 7200-1;
12   TIM_TimeBaseStructure.TIM_ClockDivision = 0x0;
13   TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
14   TIM_TimeBaseInit(TIM2, &TIM_TimeBaseStructure);
15
16   TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_Inactive;
17   TIM_OCInitStructure.TIM_Pulse = CCR1_Val;
18   TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;
19   TIM_OC1Init(TIM2, &TIM_OCInitStructure);
20   TIM_OC1PreloadConfig(TIM2, TIM_OCPreload_Disable);
21
22   TIM_ARRPreloadConfig(TIM2, ENABLE);
23   TIM_ClearITPendingBit(TIM2, TIM_IT_CC1 | TIM_IT_Update);
24   TIM_ITConfig(TIM2, TIM_IT_CC1 | TIM_IT_Update, ENABLE);
25
26   TIM_Cmd(TIM2, ENABLE);
27 }
```

B：STM32-RTC 简单设置与使用

实时时钟是啥？不就是个秒中断…（当然啦，也没这么简单的，笔者曾经基于实时时钟做过一些随机算法，还是不错的）废话不多说马上进入 RTC 的课程，我特意选取了电子白菜的超简单 RTC 仅仅做到的是 1S 1 个动作，也省去了别的废话直接上程序：

```
void RTC_Configuration(void)
{
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_PWR | RCC_APB1Periph_BKP, ENABLE);
    PWR_BackupAccessCmd(ENABLE);
    if(BKP_ReadBackupRegister(BKP_DR1) != 0xA5A5)          //如果读备份区不适 0xA5A5，说明未初始化 RTC
    {
        BKP_DeInit();                                      //重定义 BKP 备份区
        RCC_LSEConfig(RCC_LSE_ON);                         //等待低速晶振
        while(RCC_GetFlagStatus(RCC_FLAG_LSERDY) == RESET);//等待外部晶振震荡 需要等待比较长的时间
        RCC_RTCCLKConfig(RCC_RTCCLKSource_LSE);            //使用外部晶振 32768
        RCC_RTCCLKCmd(ENABLE);                             //允许 RTC
        RTC_WaitForSynchro();                              //等待 RTC 寄存器同步
        RTC_WaitForLastTask();
        RTC_ITConfig(RTC_IT_SEC, ENABLE);                  //允许 RTC 的秒中断(还有闹钟中断和溢出中断可设置)
        RTC_WaitForLastTask();                             //32768 晶振预分频值是 32767,不过一般来说晶振都不那么准
        RTC_SetPrescaler(32776);                           //如果需要校准晶振,可修改此分频值
        RTC_WaitForLastTask();
        BKP_WriteBackupRegister(BKP_DR1, 0xA5A5);          //写入 RTC 后备寄存器 1 0xa5a5
        RTC_Blank=1;                                       //这个标志代表 RTC 是没有预设的(或者说是没有上纽扣电池)
    }
    else                                                   //如果 RTC 已经设置是断过主电后再次进入的则
    {
        RTC_WaitForSynchro();                              //等待 RTC 与 APB 同步
        RTC_WaitForLastTask();
    RTC_ITConfig(RTC_IT_SEC, ENABLE);                      //使能秒中断
    RTC_WaitForLastTask();
  }
    RCC_ClearFlag();                                       //清除标志
}
```

以上这段是 RTC 的配置，当中判断了是否掉电，而下面一部分则是在 IT 中搞定秒中断，至于 NVIC 部分： NVIC_InitStructure.NVIC_IRQChannel = RTC_IRQn; 仅这些需要提示

```
extern volatile bool Sec;  //1S 标志
void RTC_IRQHandler(void)
{
        if(RTC_GetITStatus(RTC_IT_SEC) != RESET) //RTC 发生了秒中断（也有可能是溢出或者闹钟中断)
        {       RTC_ClearITPendingBit(RTC_IT_SEC);
                Sec=TRUE;

                //以免 RTC 计数溢出,这里限制了 RTC 计数值的大小.0x0001517f 实际就是一天的秒数
                if(RTC_GetCounter() >= 0x0001517f)
                {
                        RTC_SetCounter(0x0);
                }
        }
}
```

以上这段是 IT 中秒中断，以 Sec 作为标志。

对比这讲中的两种 1s 的方式，我们还可以复习下在第二讲中提到的另一种方式——滴答时钟：

```
void RCC_Configuration(void)
{
        SystemInit();
        RCC_GetClocksFreq(&RCC_ClockFreq);
        //SYSTICK分频--1ms的系统时钟中断
        if (SysTick_Config(SystemFrequency / 1000))
        {
                /* Capture error */
                while (1);
        }
}
```

```
volatile u16 Timer1;
void SysTickDelay(u16 dly_ms)
{
        Timer1=dly_ms;
        while(Timer1);
}

void SysTick_Handler(void)
{
        if(Timer1)Timer1--;
}
```

应用举例：SysTickDelay(500);

C：课后练习

第五讲旨在学习 TIMx 作为定时内容，将 1s 的时间加以分析。文中提到了 3 种可以产生一秒的方式，希望课后都可以试一下。小字部分请将 PDF 放大后查阅。谢谢。

D：听课笔记