【理论】STM32定时器时间计算公式 +【实践】TIM中断1s计时一次

前言：定时器 TIM 的详细知识点见我的博文：11.TIM定时中断-CSDN博客

STM32 定时器时间计算公式

$$Tout = \frac{((ARR+1) \times (PSC+1))}{Tclk}$$

公式解释：

ARR（TIM_Period）：自动重装载值，是定时器溢出前的计数值

PSC（TIM_Prescaler）：预分频值，是用来降低定时器时钟频率的参数

Tclk：定时器的输入时钟频率（单位Mhz），通常为系统时钟频率或者定时器外部时钟频率

Tout：定时器溢出时间（单位us）。一定要注意这个单位是us

公式由来：

1.定时器的时钟频率是Tclk，TIM_Prescaler即为PSC的值。时钟频率被分频了PSC+1，那么此时定时器的最终**频率为** $\frac{Tclk}{PSC+1}$ CSDN @阿齐Archie，故可知定时器计数值加1所需的**时间为** $\frac{PSC+1}{Tclk}$ CSDN @阿齐Archie

注：时间等于频率的倒数

2.自动重装载值即TIM_Period即ARR，定时器从0计数到ARR时清零。由第一步已经计算出了被分频了PSC+1的最终定时器的时钟频率为 $\frac{Tclk}{PSC+1}$ CSDN @阿齐Archie，这是计数一次的频率，则计数到ARR的时间为 为（ARR + 1） / $\frac{Tclk}{PSC+1}$ CSDN @阿齐Archie （时间等于频率的倒数），故定时器溢出时间（单位us）为Tout=（（ARR+1）*（PSC+1））/ Tclk。

理论联系实际，来加深理解，接下来使用STM32CubeMx + Keil来实现TIM中断实现1s计时一次。

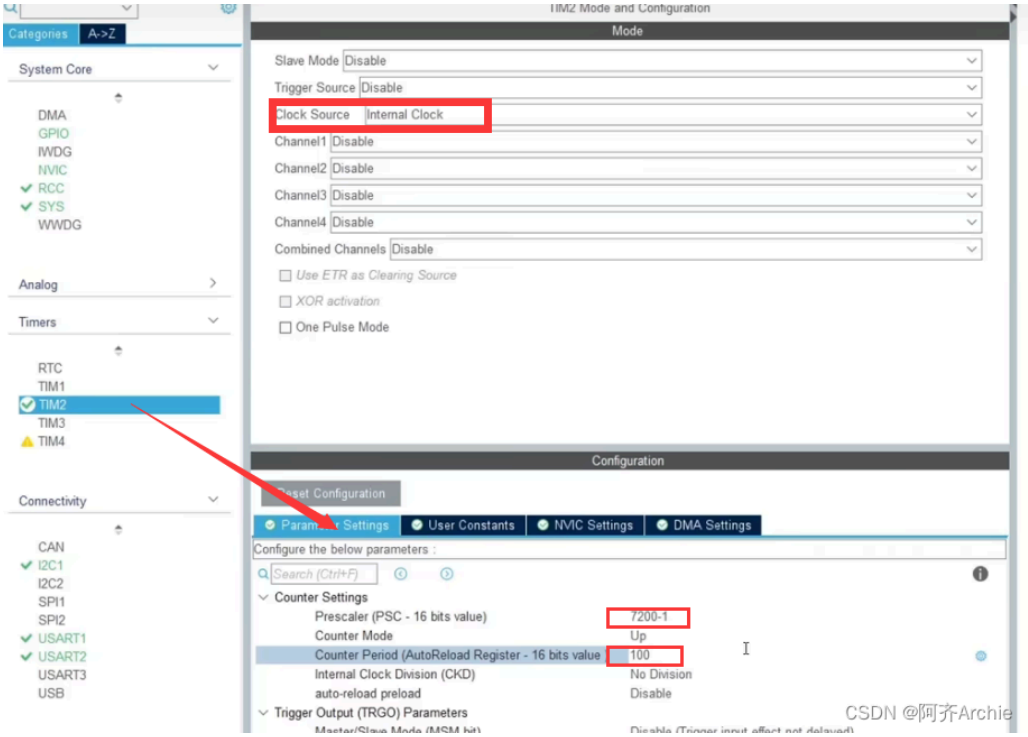
TIM中断实现1s计时一次

前言：使用的是STM32f103c8t6 ，系统主频72Mhz

目标：实现TIM中断实现1s计时一次

主要过程：配置定时器溢出时间为10ms（即定时器计数一次10ms，也就是10ms的定时器中断 ），当计次100次时是1s（1000ms），进而通过置标志位来实现1s的其它操作。

1.在STM32CubeMx中选择TIM2，设置Period（ARR）为7200，设置Prescaler（PSC）为100，根据公式计算得定时器溢出时间即定时器的中断时间（单位us）为 $\frac{7200\times100}{72}$，最后结果为10 000 us，即10ms。



对应的代码以及具体配置如下所示（HAL库版本），这段代码是一个使用TIM2定时器进行初始化配置的函数。

具体配置如下：

> 设置TIM2的时钟源配置为默认值。
> 设置TIM2的主配置为默认值。
> 对htim2即TIM_HandleTypeDef类型的结构体变量进行初始化配置：设置htim2的实例为TIM2。
> 设置htim2的预分频器　为7200-1，这将把输入时钟频率除以7200来得到TIM2的时钟频率。
> 设置htim2的计数模式为向上计数模式TIM_COUNTERMODE_UP。
> 设置htim2的计数器周期为100，这意味着当计数器达到100时，将发生定时器事件（溢出或中断）。
> 设置htim2的时钟分频因子为TIM_CLOCKDIVISION_DIV1即无时钟分频。
> 禁用htim2的自动重装载预装载功能TIM_AUTORELOAD_PRELOAD_DISABLE。这意味着在更新事件时，直接将新的周期值加载到计数器。

```
void MX_TIM2_Init(void)
{
  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig = {0};

  htim2.Instance = TIM2;
  htim2.Init.Prescaler = 7200-1;
  htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim2.Init.Period = 100;
  htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
}
```

2.写定时器2中断服务函数，10ms一次中断。这段代码是在定时器2的周期到达时触发的回调函数。在每次定时器2的周期到达时，回调函数`HAL_TIM_PeriodElapsedCallback()`会被调用。代码以及具体流程如下。

具体代码流程如下：

> 首先判断触发回调函数的定时器实例是否是htim2。如果是htim2实例，即定时器2的周期到达，进入下一步。
> `index_10ms`变量自增1，表示经过了10毫秒。
> 如果`index_10ms`变量的值能够被100整除（即经过了1秒），则将`index_led`变量设置为1。

这段代码的作用是，每隔10毫秒触发一次定时器2的中断服务函数。通过`index_10ms`变量来计数，当计数到100时（经过1秒），将`index_led`变量置为1。

在实际应用中，可以根据`index_led`变量的值来控制相关的LED灯或者执行其他操作，实现定时任务的触发和事件响应。

```
static uint16_t index_10ms = 0;
uint16_t index_led = 0;

/**
  * @brief          定时器2中断服务函数,10ms一次中断
  * @param[in]      htim:定时器
  * @retval         none
  */
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
```

```
    if (htim->Instance == htim2.Instance)
    {
        index_10ms++;
    if(index_10ms%100==0)
    {
      index_led=1;
    }
    }
}
```

3.利用定时器中断来写你自己定义的功能函数。我写的功能函数是实现1s打印一次hello，word。

这段代码其中的逻辑是通过检测外部定义的`index_led`变量的值来执行特定的操作。代码以及具体流程如下。

具体代码流程如下：

> - 当`index_led`变量的值为1时，执行以下操作：
> - 打印输出"hello,world"字符串。
> - 将`index_led`变量的值重新设置为0，表示已经处理过这次触发。

这段程序逻辑的作用是在每次`index_led`变量变为1时，打印输出"hello,world"字符串，并且只执行一次，直到下次`index_led`又变为1。

```
extern uint16_t index_led;
uint8_t led_status =0;
/**
  * @brief          自定义功能函数
  * @param[in]      none
  * @retval         none
  */
void user(void)
{
  if(index_led==1)
  {
    printf("hello,world\r\n");
    index_led=0;
  }
}
```