第15章、SysTick系统滴答定时器

一、概述

1. SysTick是一个**24位定时器**，属于**Cortex-M4内核**中的一个外设，类似NVIC。

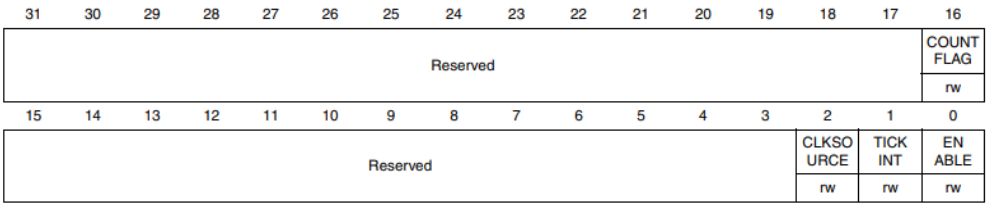
```
SysTick定时器一次最多可以计数2^24(24bit)个时钟脉冲，这个脉冲计数值保存在当前计数值寄存器STK_VAL(Systick current value register)中，只能向下计数，也就是倒计数。每接收到一个时钟脉冲(CPU主频)，STK_VAL的值就会向下减1，当减到0时，硬件会自动将重装载寄存器STK_LOAD(可以设定，跟STK_VAL初始值相等)中保存的数值加载到STK_VAL，使其重新计数。并且，系统滴答定时器就产生一次中断，以此循环往复，只要不把它在SysTick控制及状态寄存器中的使能位清除，就永不停息。
```

2. 一个周期定时器，用于提供时间基准，多为操作系统所使用，常用于对时间要求严格的情况，意义是很重要的。
3. **注意：想要更改滴答定时器的中断周期，只能通过SysTick_LOAD来更改，不能通过SysTick_VAL更改。**

二、相关寄存器

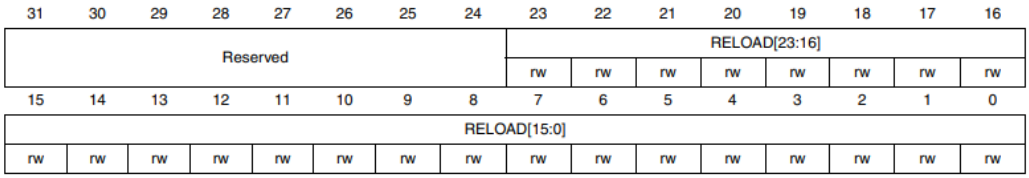
寄存器名称	寄存器描述
STK_CTRL	SysTick 控制及状态寄存器
STK_LOAD	SysTick 重装载数值寄存器
STK_VAL	SysTick 当前数值寄存器
STK_CALIB	SysTick 校准数值寄存器

1. STK_CTRL **控制及状态寄存器**



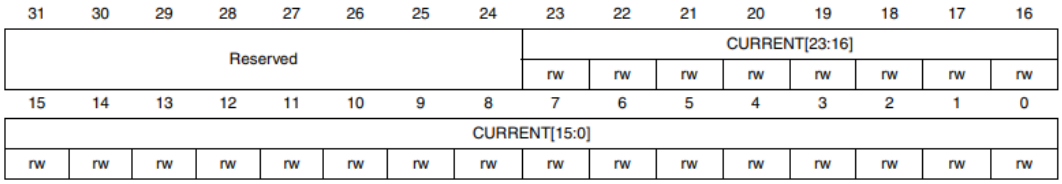
位段	名称	类型	复位值	描述
16	COUNTFLAG	R/W	0	如果在上次读取本寄存器后，SysTick已经数到了0，则该位为1、如果读取该位，该位将自动清零
2	CLKSOURCE	R/W	0	0= AHB / 8 ， 1=处理器时钟(AHB)
1	TICKINT	R/W	0	1=SysTick倒数到0时产生SysTick异常请求 ， 0=数到0时无动作
0	ENABLE	R/W	0	使能位，值1启动计数

2. `STK_LOAD` **重装载数值寄存器**



位段	名称	类型	复位值	描述
$23:0$	RELOAD	R/W	0	值-1，比如需要100个计数，那么把它设置为99。

3. `STK_VAL` **当前数值寄存器**



位段	名称	类型	复位值	描述
$23:0$	CURRENT	R/W	0	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

4. `STK_CALIB` **校准数值寄存器**

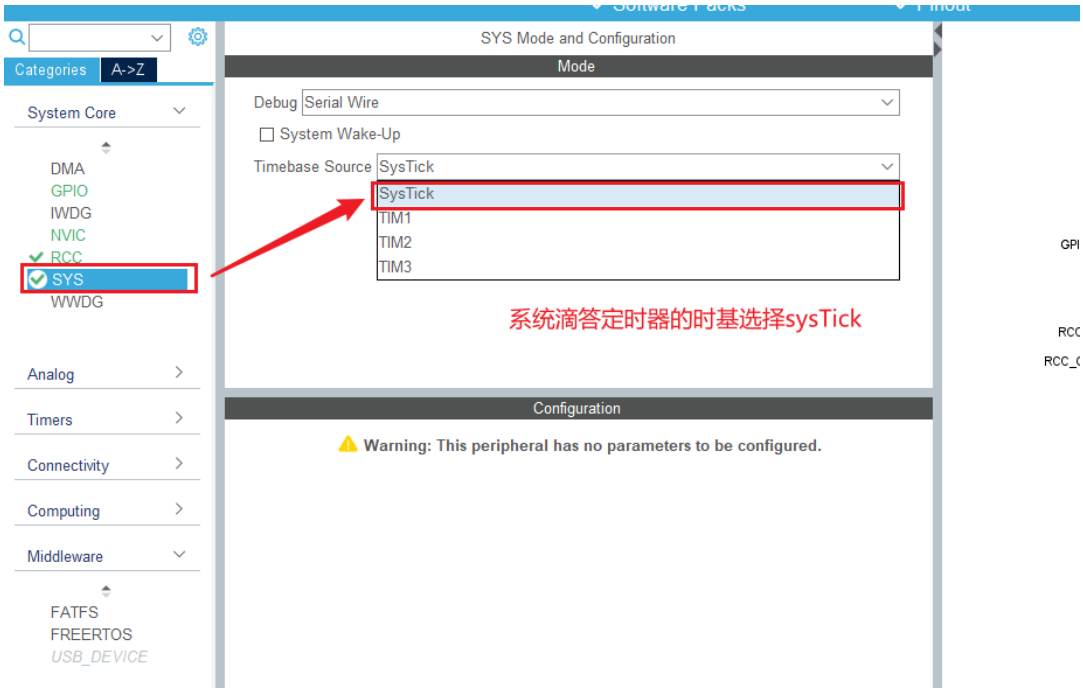
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
NO REF	SKEW	Reserved						TENMS[23:16]							
r	r							r	r	r	r	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TENMS[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

位段	名称	类型	复位值	描述
31	NOREF	R	-	1：无可用参考时钟；0：有可用参考时钟(AHB/8)。该位由芯片硬件设定。
30	SKEW	R	-	指示TENMS值是否准确；1:1毫秒不精确定时的校准值未知，因为TENMS值未知。
23：0	TENMS	R	0	当SysTick计数器在AHB/8上作为外部时钟运行时，指示校准值。 该值由芯片厂家给出。

三、Cubemx配置

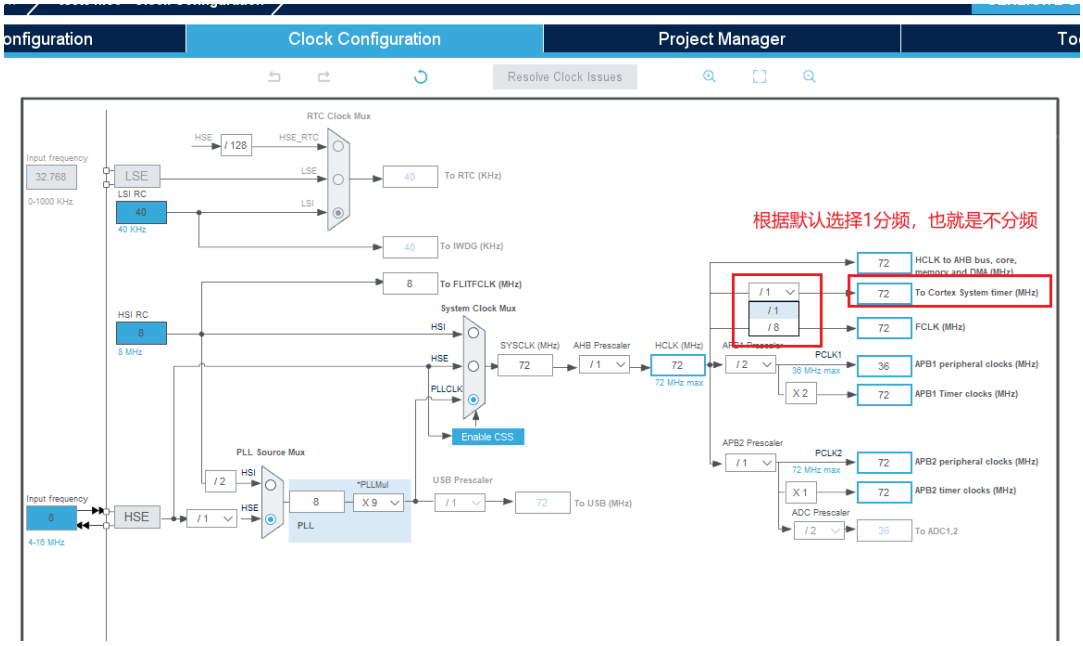
因为是内核的系统定时器，所以，配置中只有两个地方需要知道。

1. 系统滴答定时器时基选择



2. 时钟树设置

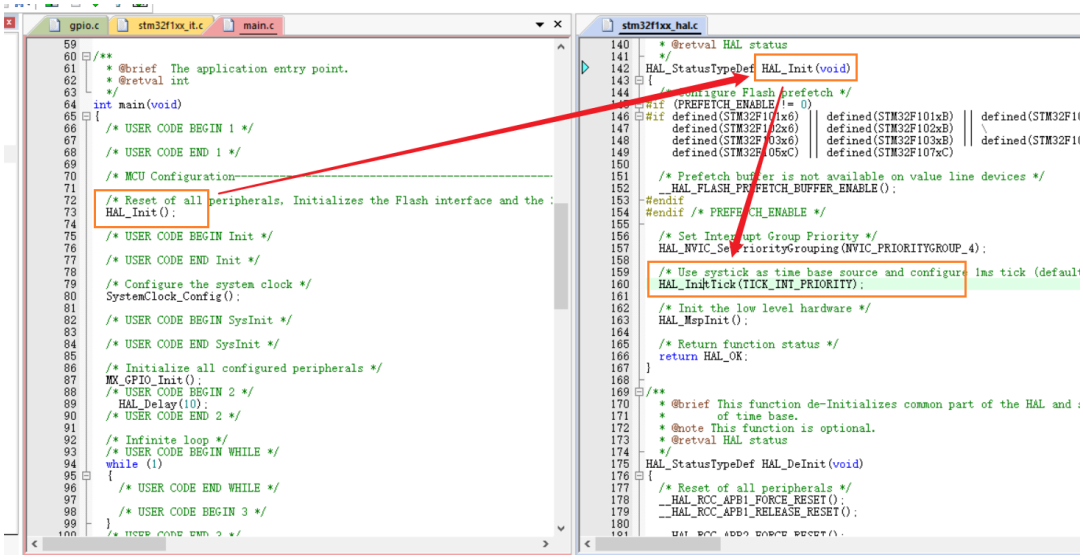
根据数据手册我们可以查到，系统滴答定时器的频率来源于主频AHB上

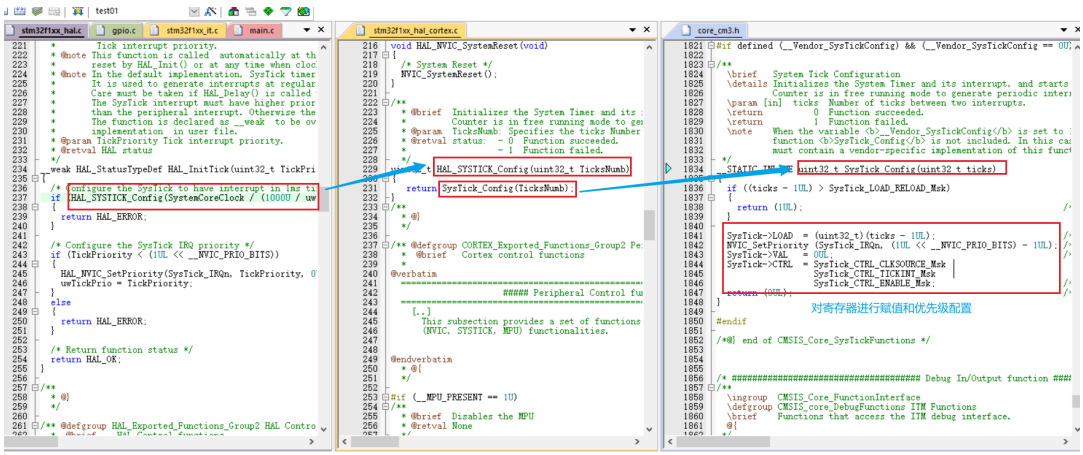


四、代码

在系统定时器中，我们只需要知道用到几个接口函数即可。

1. 初始化过程





初始化的系统滴答定时器函数`HAL_InitTick()`中包含了`HAL_SYSTICK_Config()`初始化系统计时器及其中断和`HAL_NVIC_SetPriority()`系统滴答定时器中断优先级配置。并且要想改变滴答定时器的定时中断，那么我我们就需要在`void SystemClock_Config(void);`函数的最后加入这三个函数。

```
// HAL_RCC_GetHCLKFreq()/1000    1ms中断一次
// HAL_RCC_GetHCLKFreq()/100000  10us中断一次
// HAL_RCC_GetHCLKFreq()/1000000 1us中断一次
HAL_SYSTICK_Config(HAL_RCC_GetHCLKFreq()/1000000);// 配置并启动系统滴答定时器
/* 系统滴答定时器时钟源 */
HAL_SYSTICK_CLKSourceConfig(SYSTICK_CLKSOURCE_HCLK);
/* 系统滴答定时器中断优先级配置 */
HAL_NVIC_SetPriority(SysTick_IRQn, 0, 0);
```

2. `HAL_SYSTICK_Config()`初始化系统计时器及其中断，并启动系统计时器。

```
/**
  * @brief  初始化系统计时器及其中断，并启动系统计时器。
  *         计数器处于自由运行模式，以产生周期性中断。
  * @param  TicksNumb: 指定两次中断之间的滴答数。
  * @retval status:  - 0  Function succeeded.
  *                  - 1  Function failed.
  */
uint32_t HAL_SYSTICK_Config(uint32_t TicksNumb)
{
   return SysTick_Config(TicksNumb);
}
/*
 * 因为这个定时器是内核所有，所有更改值的时候，也是调用内核函数更改
 */
// 用例：
// HAL_RCC_GetHCLKFreq()/1000    1ms中断一次
// HAL_RCC_GetHCLKFreq()/100000  10us中断一次
// HAL_RCC_GetHCLKFreq()/1000000 1us中断一次
HAL_SYSTICK_Config(HAL_RCC_GetHCLKFreq()/1000); // 配置并启动系统滴答定时器
```

3. `HAL_NVIC_SetPriority()`系统滴答定时器中断优先级配置

```
/**
  * @brief  设置中断的优先级。
  * @param  IRQn: 外部中断号。
  *         此参数可以是IRQn_类型枚举的枚举数（有关完整的STM32设备IRQ通道列表，请参阅相应的CMSIS设备文件（stm32fxxxxx.h））
  * @param  PreemptPriority: IRQn通道的抢占优先级。
```

```
  *         此参数可以是介于0和15之间的值。优先级值越低，表示优先级越高
  * @param  SubPriority: IRQ通道的子优先级
  *         此参数可以是介于0和15之间的值。优先级值越低，表示优先级越高。
  * @retval None
  */
void HAL_NVIC_SetPriority(IRQn_Type IRQn, uint32_t PreemptPriority, uint32_t SubPriority)
{
  uint32_t prioritygroup = 0x00U;
  /* Check the parameters */
  assert_param(IS_NVIC_SUB_PRIORITY(SubPriority));
  assert_param(IS_NVIC_PREEMPTION_PRIORITY(PreemptPriority));

  prioritygroup = NVIC_GetPriorityGrouping();

  NVIC_SetPriority(IRQn, NVIC_EncodePriority(prioritygroup, PreemptPriority, SubPriority));
}
// 用例：
/* 系统滴答定时器中断优先级配置 */
HAL_NVIC_SetPriority(SysTick_IRQn, 0, 0);
```

4. 默认情况用`Cubemx`配置的工程中是不会调用系统滴答定时器的回调函数的，这里需要自行更改和调用。

```
// 在stm32fxxx_it.c中
/**
* @brief 此函数用于处理系统计时。
*/
void SysTick_Handler(void)
{
  /* USER CODE BEGIN SysTick_IRQn 0 */

  /* USER CODE END SysTick_IRQn 0 */
  HAL_IncTick();
  HAL_SYSTICK_IRQHandler(); // 默认是不加的，加入该函数，该函数会调用回调函数
  /* USER CODE BEGIN SysTick_IRQn 1 */

  /* USER CODE END SysTick_IRQn 1 */
}
// 在stm32fxxx_hal_cortex.c中
/**
```

```
  * @brief  此函数处理SYSTICK中断请求。
  * @retval None
  */
void HAL_SYSTICK_IRQHandler(void)
{
  HAL_SYSTICK_Callback();
}
/**
  * @brief  系统回调函数，这个函数是一个弱定义，可以在其他文件中重写
  * @retval None
  */
__weak void HAL_SYSTICK_Callback(void)
{
  /* NOTE : 不应修改此函数名，当需要回调时,回调可以在用户文件中实现。
*/
}
```

5. 通常情况下我们用的最多的就是延时函数

```
/**
  * @brief 此函数根据变量incremented提供最小延迟（以毫秒为单位,看看情
况）。
  * @note 在默认实现中，SysTick timer是时基的来源。
  *       它用于在uwTick递增的固定时间间隔生成中断。
  * @note 此函数被声明为弱定义，如果用户文件中有其他实现，将被覆盖。
  * @param Delay指定延迟时间长度(默认情况是毫秒,实际根据情况确定)。
  * @retval None
  */
__weak void HAL_Delay(uint32_t Delay)
{
  uint32_t tickstart = HAL_GetTick();
  uint32_t wait = Delay;

  /* Add a freq to guarantee minimum wait */
  if (wait < HAL_MAX_DELAY)
  {
    wait += (uint32_t)(uwTickFreq);
  }

  while ((HAL_GetTick() - tickstart) < wait)
  {
  }
}
```