STM32Cube高效开发教程（基础篇）

第9章 基础定时器

王维波
中国石油大学（华东）控制科学与工程学院

STM32Cube高效开发教程（基础篇）

作者：王维波，鄢志丹，王钊

人民邮电出版社

2021年9月出版

如果有读者需要本书课件的PPT版本用于备课，可以给作者发邮件免费获取，并可加入专门的教学和技术交流QQ群

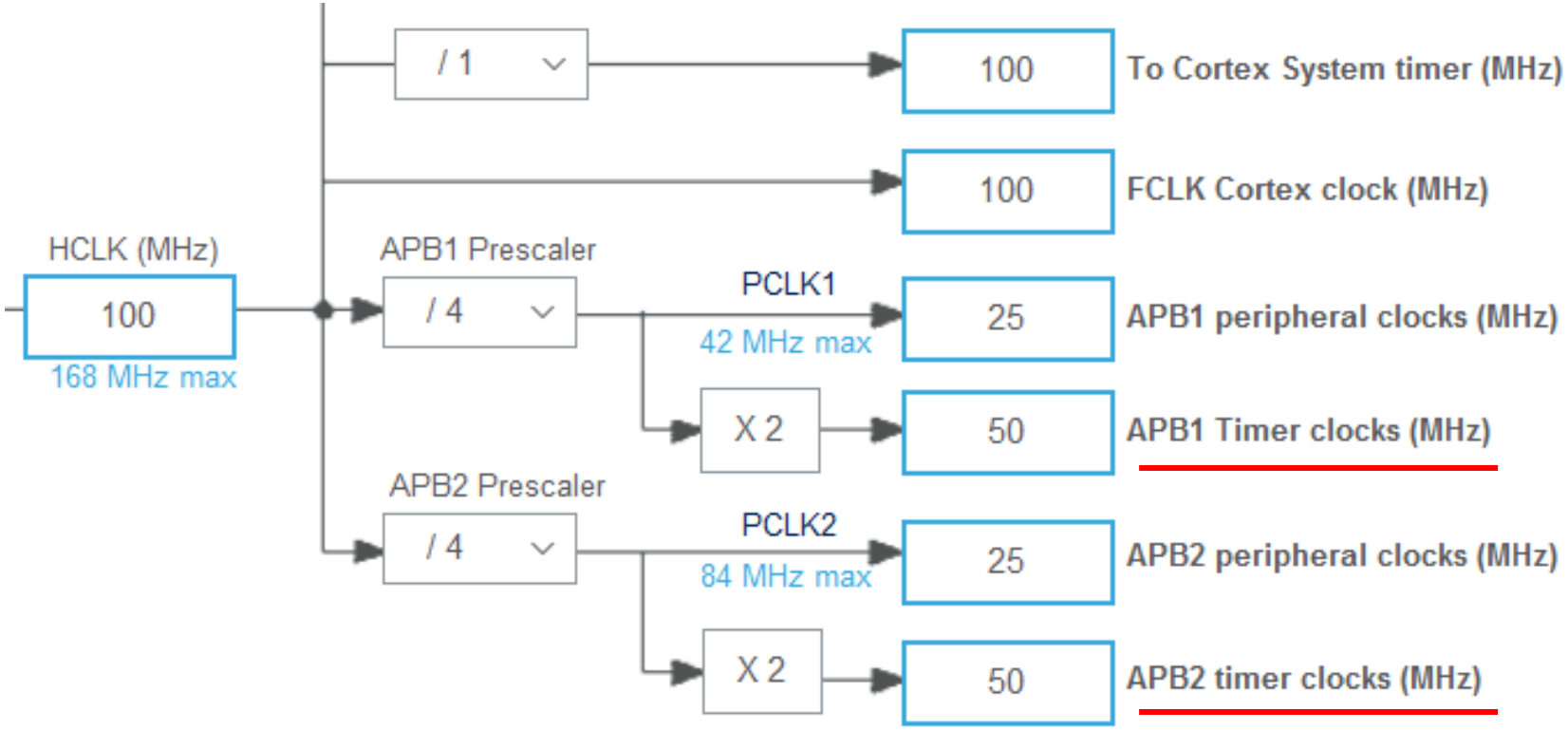
邮箱：wangwb@upc. edu. cn

第9章 基础定时器

STM32F407有2个高级控制定时器（Advanced-control timer），8个通用定时器（General-purpose timer），2个基础定时器（Basic timer）

类型	定时器	计数器长度	计数类型	DMA 请求生成	捕获/比较通道数	挂载总线
基础	TIM6，TIM7	16位	递增	有	0	APB1
通用	TIM2，TIM5	32位	递增、递减、递增/递减	有	4	APB1
	TIM3，TIM4	16位	递增、递减、递增/递减	有	4	APB1
	TIM9	16位	递增	无	2	APB2
	TIM12	16位	递增	无	2	APB1
	TIM10，TIM11	16位	递增	无	1	APB2
	TIM13，TIM14	16位	递增	无	1	APB1
高级控制	TIM1，TIM8	16位	递增、递减、递增/递减	有	4	APB2

时钟树上APB1和APB2总线上的定时器时钟信号来源



第9章 基础定时器

9.1 定时器概述

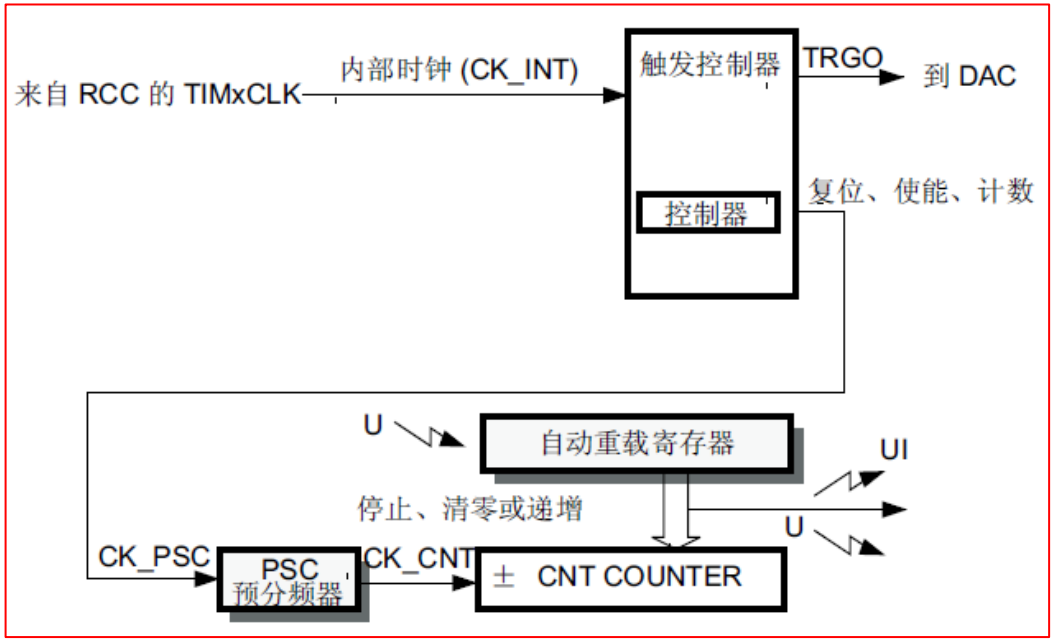
9.2 基础定时器内部结构和功能

9.3 基础定时器HAL驱动程序

9.4 基础定时器使用示例

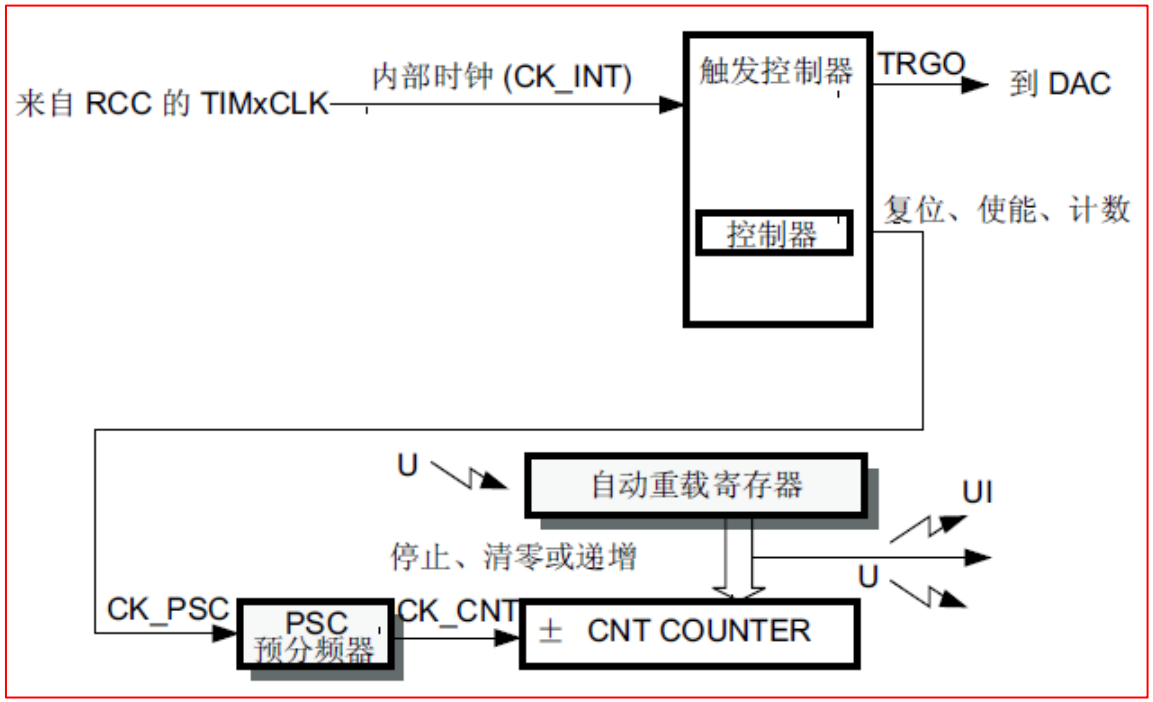
TIM6和TIM7是基础定时器，它们的功能相同

- 只能使用内部时钟信号CK_INT，最高频率84MHz
- 16位自动重载寄存器（auto-reload register）
- 16位可编程预分频器，分频范围1至65536
- 可以用于触发DAC的同步电路
- 只有一种事件引起中断，即计数器上溢时产生的更新事件



它有3个16位寄存器，这三个寄存器的值都可以读写。其中，预分频寄存器和自动重载寄存器有影子寄存器用于底层工作。

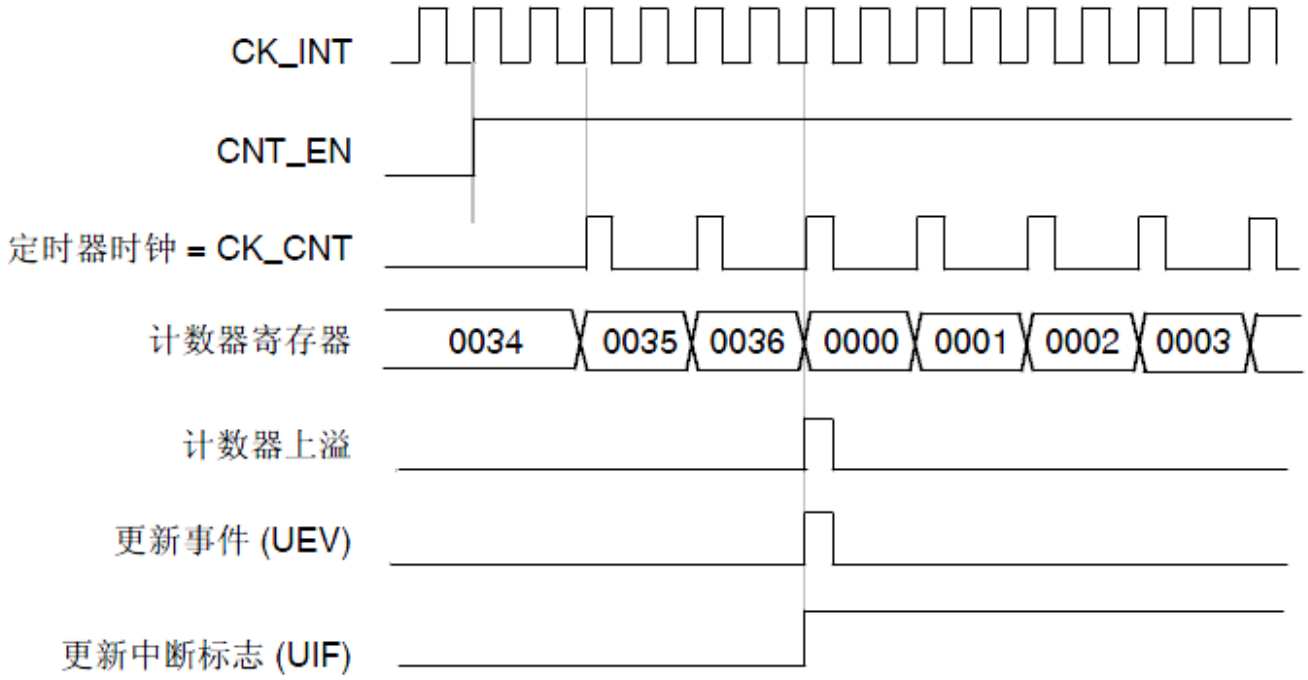
- 预分频寄存器（TIMx_PSC）
- 计数寄存器（TIMx_CNT），计数溢出时产生更新事件（Update Event，UEV）中断



- 自动重载寄存器（TIMx_ARR），存储的值用于与计数器的值进行比较

假设CK_INT频率为50MHz，预分频系数为50000，则计数器时钟频率为

$$f_{CK\_CNT} = \frac{f_{CK\_PSC}}{50000} = 1000Hz$$



若自重载寄存器值为1000，则定时周期为1000毫秒，计数溢出时产生UEV事件

9.3 基础定时器HAL驱动程序

9.3.1 基础定时器相关函数

9.3.2 其他通用操作函数

9.3.3 中断处理

9.3.1 基础定时器相关函数

分组	函数名	功能描述
初始化	HAL_TIM_Base_Init()	定时器初始化，设置各种参数和连续定时模式
	HAL_TIM_Base_MspInit()	弱函数，在HAL_TIM_Base_Init()里被调用，需重新实现
	HAL_TIM_OnePulse_Init()	将定时器配置为单次定时模式，需要先执行HAL_TIM_Base_Init()
启动和停止	HAL_TIM_Base_Start()	以轮询工作方式启动定时器，不会产生中断
	HAL_TIM_Base_Stop()	停止轮询工作方式的定时器
	HAL_TIM_Base_Start_IT()	以中断工作方式启动定时器，发生UEV事件时产生中断
	HAL_TIM_Base_Stop_IT()	停止中断工作方式定时器的
	HAL_TIM_Base_Start_DMA()	以DMA工作方式启动定时器
	HAL_TIM_Base_Stop_DMA()	停止DMA工作方式的定时器
获取状态	HAL_TIM_Base_GetState()	获取基础定时器的当前状态

1. 定时器初始化

```
HAL_StatusTypeDef HAL_TIM_Base_Init(TIM_HandleTypeDef *htim);
```

参数htim是定时器对象指针，结构体TIM_HandleTypeDef

```
typedef struct
{
  TIM_TypeDef                 *Instance;     //定时器基址
  TIM_Base_InitTypeDef        Init;          // 定时器参数
  HAL_TIM_ActiveChannel       Channel;       //当前通道
  DMA_HandleTypeDef           *hdma[7];      //DMA处理相关数组
  HAL_LockTypeDef             Lock;          //是否锁定
  __IO HAL_TIM_StateTypeDef   State;         // 定时器工作状态
} TIM_HandleTypeDef;
```

其中，Instance是用定时器基址指定具体的定时器，Init是定时器的各种参数，是一个结构体类型TIM_Base_InitTypeDef

结构体TIM_Base_InitTypeDef，保存定时器参数

```
typedef struct
{
  uint32_t Prescaler;            // 预分频系数
  uint32_t CounterMode;       // 计数模式，递增、递减、递增/递减
  uint32_t Period;               // 计数周期
  uint32_t ClockDivision;       // 内部时钟分频,基本定时器无此参数
  uint32_t RepetitionCounter;  // 重复计数器值，用于PWM模式
  uint32_t AutoReloadPreload;  // 是否使能TIMx_ARR寄存器的缓冲功能
} TIM_Base_InitTypeDef;
```

定时器初始化示意代码如下：

```
 TIM_HandleTypeDef  htim6;
  htim6.Instance = TIM6;
  htim6.Init.Prescaler = 49999;
//   其他属性设置
HAL_TIM_Base_Init(&htim6);
```

2. 启动定时器

- 轮询方式：HAL_TIM_Base_Start(htim)
- 中断方式：HAL_TIM_Base_Start_IT(htim)
- DMA方式：HAL_TIM_Base_Start_DMA(htim)

参数htim是定时器对象指针，结构体TIM_HandleTypeDef

3. 获取定时器运行状态

获取定时器的运行状态

```
HAL_TIM_StateTypeDef  HAL_TIM_Base_GetState(TIM_HandleTypeDef  *htim);
```

函数返回数据是枚举类型HAL_TIM_StateTypeDef

```
typedef enum
{
  HAL_TIM_STATE_RESET     = 0x00U,    /* 定时器还未被初始化，或被禁用了  */
  HAL_TIM_STATE_READY     = 0x01U,    /* 定时器已经初始化，可以使用了    */
  HAL_TIM_STATE_BUSY      = 0x02U,    /* 一个内部处理过程正在执行        */
  HAL_TIM_STATE_TIMEOUT   = 0x03U,    /* 定时到期(Timeout)状态           */
  HAL_TIM_STATE_ERROR     = 0x04U     /* 发生错误，Reception过程正在运行 */
} HAL_TIM_StateTypeDef;
```

9.3.2 其他通用操作函数

宏函数直接操作寄存器，主要用于在定时器运行时直接读取或修改某些寄存器的值

函数名	功能描述
__HAL_TIM_ENABLE()	启用某个定时器，就是将定时器控制寄存器TIMx_CR1的CEN位置1
__HAL_TIM_DISABLE()	禁用某个定时器
__HAL_TIM_GET_COUNTER()	在运行时读取定时器的当前计数值，就是读取TIMx_CNT寄存器的值
__HAL_TIM_SET_COUNTER()	在运行时设置定时器的计数值，就是设置TIMx_CNT寄存器的值
__HAL_TIM_GET_AUTORELOAD()	在运行时读取自重载寄存器TIMx_ARR的值
__HAL_TIM_SET_AUTORELOAD()	在运行时设置自重载寄存器TIMx_ARR的值，并改变定时的周期
__HAL_TIM_SET_PRESCALER()	在运行时设置与分频系数，就是设置分频寄存器TIMx_PSC的值

这些函数都需要一个定时器对象指针作为参数，如启用定时器的函数定义如下：

```
#define __HAL_TIM_ENABLE(__HANDLE__)
                ((__HANDLE__)->Instance->CR1|=(TIM_CR1_CEN))
```

函数的功能就是将定时器的TIMx_CR1寄存器的CEN位置1。这个函数的使用示意代码如下：

```
 TIM_HandleTypeDef  htim6;
__HAL_TIM_ENABLE(&htim6);
```

9.3.3 中断处理

每个定时器有一个中断号，基础定时器只有一个UEV中断事件源，但是通用定时器和高级控制器有多个中断事件源

函数名	功能描述
__HAL_TIM_GET_FLAG()	获取某个事件是否触发的标志，就是读取状态寄存器TIMx_SR中相应的中断事件位是否置1
__HAL_TIM_GET_IT_SOURCE()	判断是否是某个事件产生的中断，返回值为SET或RESET
__HAL_TIM_CLEAR_IT()	清除某个事件的中断标志，就是将状态寄存器TIMx_SR中相应的中断事件位是否置0
__HAL_TIM_ENABLE_IT()	启用某个事件的中断，就是将中断使能寄存器TIMx_DIER中相应事件位置1
__HAL_TIM_DISABLE_IT()	禁用某个事件的中断，就是将中断使能寄存器TIMx_DIER中相应事件位置0
HAL_TIM_IRQHandler()	定时器ISR函数里的通用处理函数
HAL_TIM_PeriodElapsedCallback()	弱函数，UEV事件中断的回调函数

1. 中断事件类型

表示中断事件类型的宏定义

```
#define TIM_IT_UPDATE    TIM_DIER_UIE      //更新中断，UEV
#define TIM_IT_CC1       TIM_DIER_CC1IE    //捕获/比较1中断
#define TIM_IT_CC2       TIM_DIER_CC2IE    //捕获/比较2中断
#define TIM_IT_CC3       TIM_DIER_CC3IE    //捕获/比较3中断
#define TIM_IT_CC4       TIM_DIER_CC4IE    //捕获/比较4中断
……
```

使能某个中断事件源，其定义如下：

```
#define __HAL_TIM_ENABLE_IT(__HANDLE__, __INTERRUPT__)
    ((__HANDLE__)->Instance->DIER |= (__INTERRUPT__))
```

2. 定时器中断处理流程

所有定时器的ISR函数都是调用函数HAL_TIM_IRQHandler()

```
void TIM6_DAC_IRQHandler(void)
{
   HAL_TIM_IRQHandler(&htim6);
}
```

这个函数内部判断中断事件类型，调用对应的回调函数

```
/* TIM Update event */
 if (__HAL_TIM_GET_FLAG(htim, TIM_FLAG_UPDATE) != RESET)   //判断
     事件标志
 {
   if (__HAL_TIM_GET_IT_SOURCE(htim, TIM_IT_UPDATE) != RESET)
     //判断中断源
   {
     __HAL_TIM_CLEAR_IT(htim, TIM_IT_UPDATE);   //清除中断标志
     HAL_TIM_PeriodElapsedCallback(htim);
   }
 }
```

第9章 基础定时器

9.1 定时器概述

9.2 基础定时器内部结构和功能

9.3 基础定时器HAL驱动程序

9.4 基础定时器使用示例

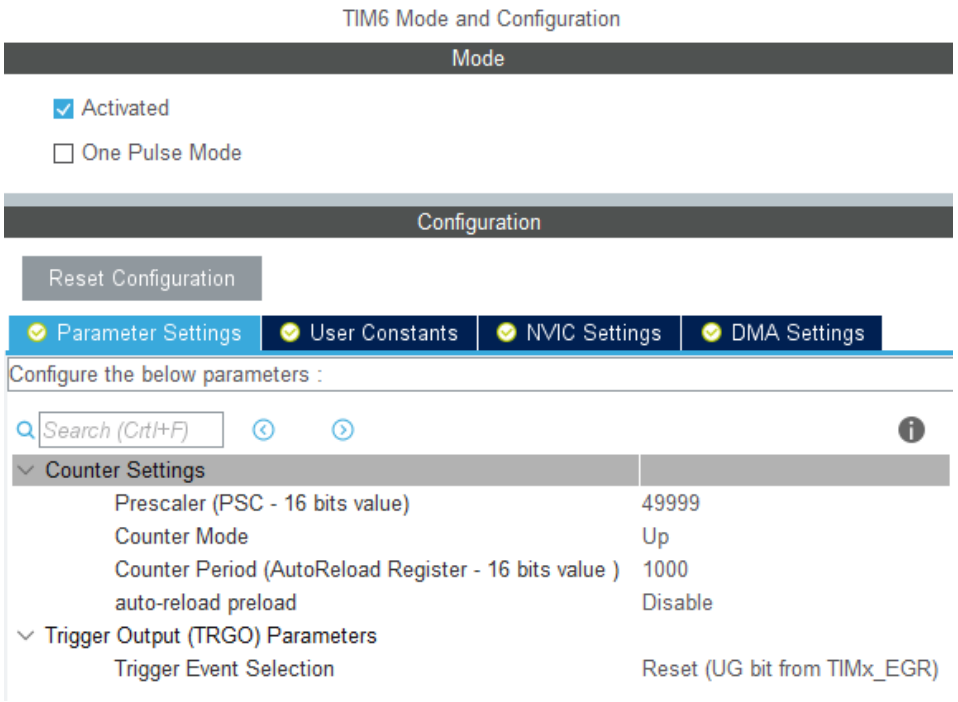
9.4.1 实例功能和STM32CubeMX项目配置

使用LED1和LED2测试基础定时器TIM6和TIM7的功能

- TIM6设置为连续定时模式，定时周期500ms，以中断方式启动TIM6，在UEV事件中断回调函数里使里使LED1输出翻转。
- TIM7设置为单次定时模式，定时周期2000ms，按下KeyRight键之后使LED2点亮，并以中断方式启动TIM7，在UEV事件中断回调函数里使里使LED2输出翻转。

定时器TIM6的模式和配置

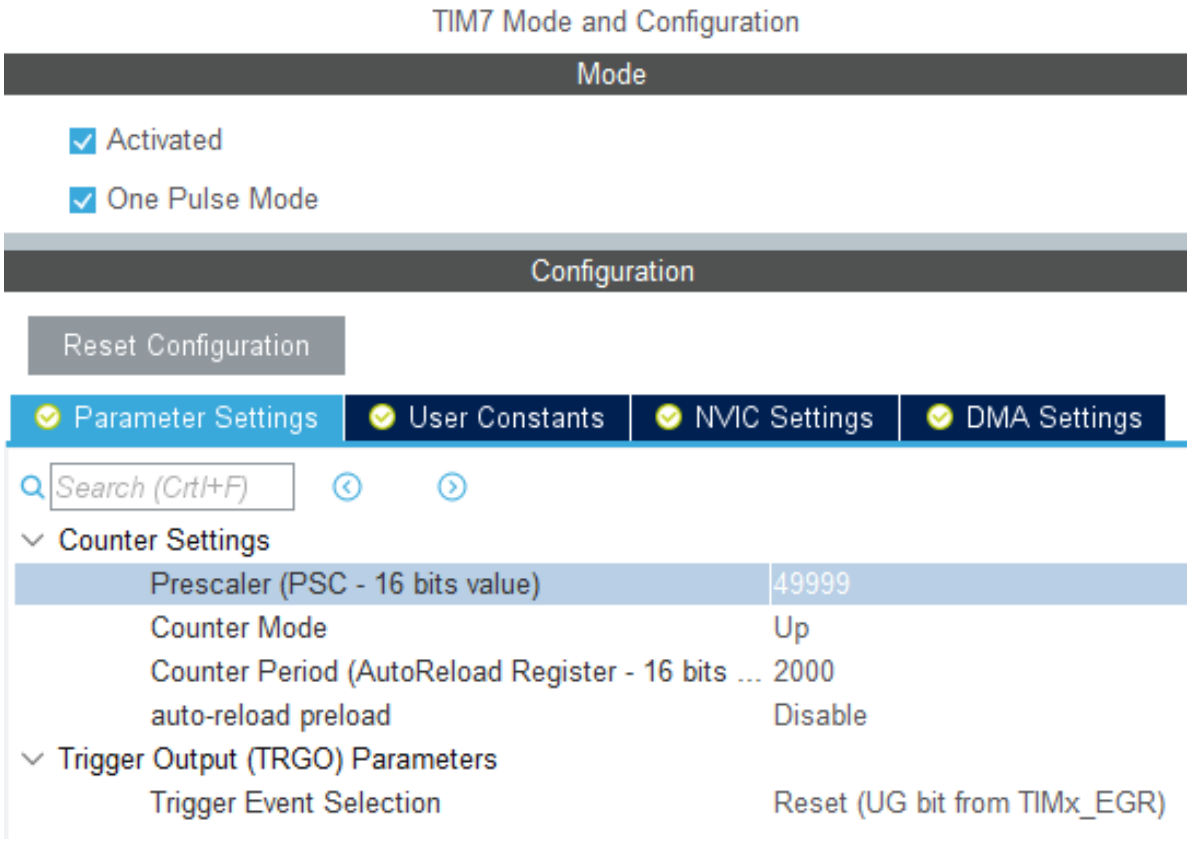
- Prescaler，预分频系数
- Counter Mode，计数模式
- Counter Period，计数周期
- auto-reload preload，是否启用TIMx_ARR寄存器的缓冲功能
- Trigger Event Selection，触发事件选择，设置为1就是用软件方式触发了一次溢出



定时器TIM7的模式和配置

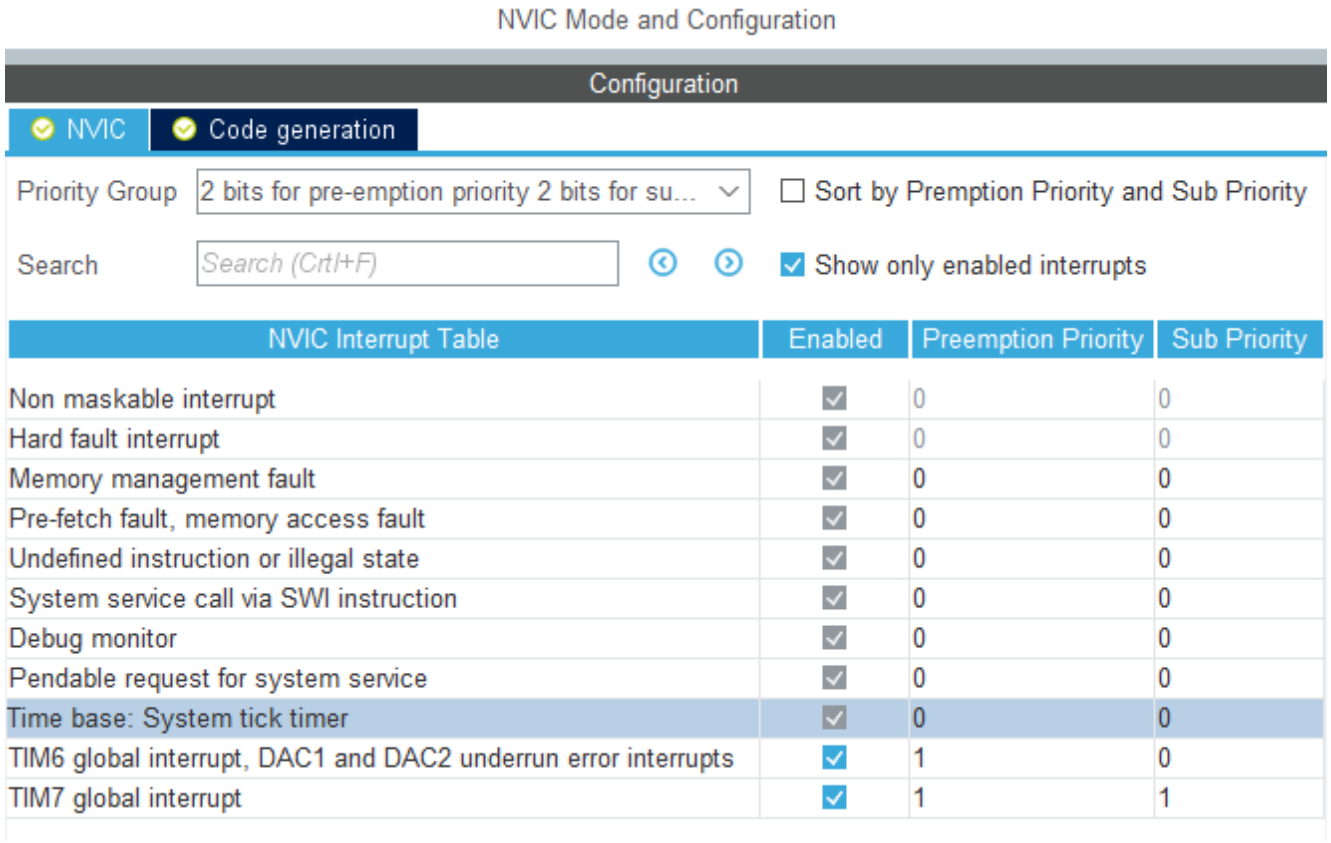
One Pulse Mode，单脉冲模式（输出比较的特殊模式）

用于基础定时器时，就是单次定时模式



NVIC设置

两个定时器的抢占优先级都设置为1，其实设置为0也是没有问题的，因为本实例程序里不会用到延时函数HAL_Delay()



9.4.2 项目初始化代码分析

1. 主程序

完整代码看源程序main.c

```
LED1_OFF();//熄灭 LED1
LED2_OFF();//熄灭 LED2
HAL_TIM_Base_Start_IT(&htim6); //以中断方式启动TIM6

while (1)
{
    KEYS  curKey=ScanPressedKey(KEY_WAIT_ALWAYS);
    if (curKey==KEY_RIGHT)
    {
        LED2_ON();//点亮LED2
        HAL_TIM_Base_Start_IT(&htim7); //以中断方式启动TIM7
        HAL_Delay(300);//消除按键后抖动的影响
    }
}
```

- 以中断方式启动TIM6
- 在while循环里检测KeyRight，以中断方式启动TIM7

2. 定时器初始化

文件tim.h中定义了定时器对象和初始化函数

```
extern TIM_HandleTypeDef htim6;    //TIM6的外设对象变量
extern TIM_HandleTypeDef htim7; ;   //TIM7的外设对象变量


void MX_TIM6_Init(void);          //TIM6初始化函数
void MX_TIM7_Init(void);          //TIM7初始化函数
```

完整代码看源程序tim.c

- TIM6定时周期500ms，连续定时
- TIM7定时周期2000ms，单脉冲模式

函数HAL_TIM_Base_Init(&htim6) 中还调用了一个Msp函数，重新实现的这个函数设置定时器的中断

```
void HAL_TIM_Base_MspInit(TIM_HandleTypeDef* tim_baseHandle)
{
  if(tim_baseHandle->Instance==TIM6)
  {
    __HAL_RCC_TIM6_CLK_ENABLE();    // TIM6 时钟使能
    /* TIM6 中断初始化 */
    HAL_NVIC_SetPriority(TIM6_DAC_IRQn, 1, 0);   //设置中断优先级
    HAL_NVIC_EnableIRQ(TIM6_DAC_IRQn);        //使能中断
  }
}
```

3. 定时器中断处理

定时器TIM6和TIM7的ISR函数的代码框架

```
void TIM6_DAC_IRQHandler(void)
{
    HAL_TIM_IRQHandler(&htim6);
}

void TIM7_IRQHandler(void)
{
    HAL_TIM_IRQHandler(&htim7);
}
```

9.4.3 编写用户功能代码

在文件tim.c中重新实现UEV事件中断回调函数

```
/* USER CODE BEGIN 1 */
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
  if (htim->Instance == TIM6)
    HAL_GPIO_TogglePin(LED1_GPIO_Port, LED1_Pin);
  else if(htim->Instance == TIM7)
    HAL_GPIO_TogglePin(LED2_GPIO_Port, LED2_Pin);
}
/* USER CODE END 1 */
```

示例运行效果

运行时的LCD提示信息