嵌入式系统原理及实验

顾　震

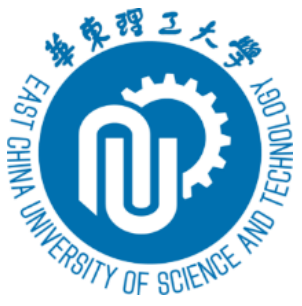
信息科学与工程学院自动化系

华东理工大学

Email: guzhen@ecust.edu.cn

课程大纲

1. 嵌入式系统导论
2. Cortex-M3微处理器
3. STM32最小系统及开发环境
4. 嵌入式C语言
5. 通用输入输出GPIO模块
6. 中断
7. 定时器原理与应用
8. USART通信原理及实现
9. DMA控制器
10. SPI与I2C通信原理及实现
11. 模数转换原理及实现
12. 人工智能辅助的嵌入式项目开发
13. 嵌入式应用前沿



7. 定时器原理与应用

本章知识与能力要求

- 了解和掌握定时器的基本功能；
- 理解和掌握STM32定时器的内部结构、工作模式和主要特性；
- 掌握STM32通用定时器延时的工作原理；
- 熟悉STM32 HAL库中有关定时器的库函数；
- 掌握SysTick定时器定时功能；
- 掌握基于HAL库实现定时器精确延时功能。

7. 定时器原理与应用

- 计时应用于日常生活中的各方面，嵌入式系统中采用定时器外设来实现各种需要计时的应用。
- 计时的本质是利用**稳定重复的周期性事件**作为“尺子”，用 周期重复的**次数**度量时间的长短。

7. 定时器原理与应用

7.1 STM32定时器模块

7.2 PWM

7.3 SysTick定时器

7.4 看门狗

7.5 定时器模块的HAL库接口函数及应用

7.1 STM32定时器模块

定时器的主要功能

■ 定时

在需要精确控制时间的场合，通过微控制器内部时钟脉冲计数实现定时。

■ 计数

脉冲计数，使用微控制器的外部时钟（PCLK）来计数，可对固定周期的脉冲信号计数。

■ 输入捕获

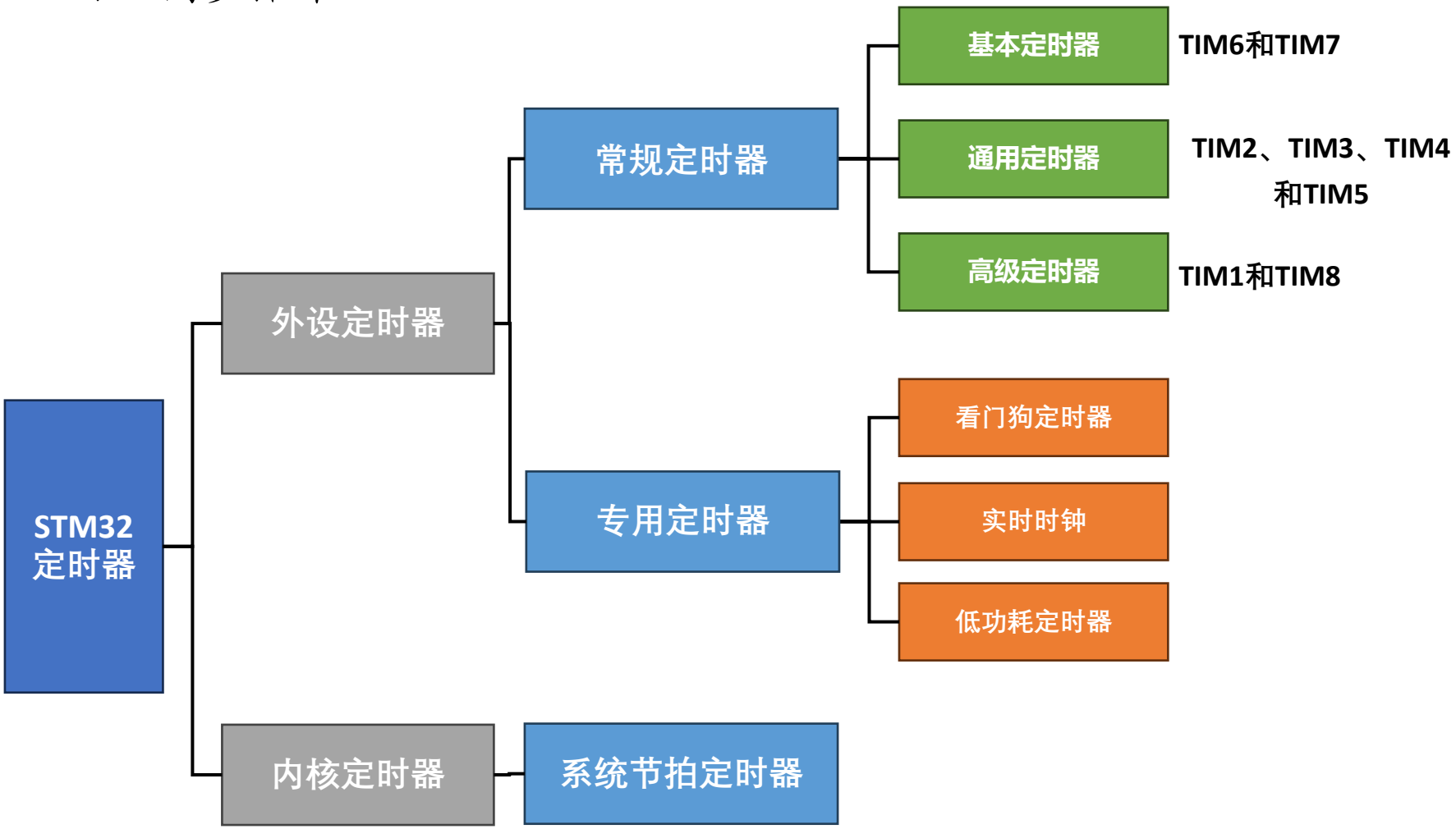
对输入信号进行捕获，实现对脉冲的频率测量，可用于对外部输入信号脉冲宽度的测量，比如测量电机转速。

■ 输出比较

将计数器计数值和设定值进行比较，根据比较结果输出不同电平，用于控制输出波形，比如直流电机的调速。

7.1 STM32定时器模块

- STM32定时器种类多，功能强大，这些定时器完全独立、互不干扰，可以同步操作。

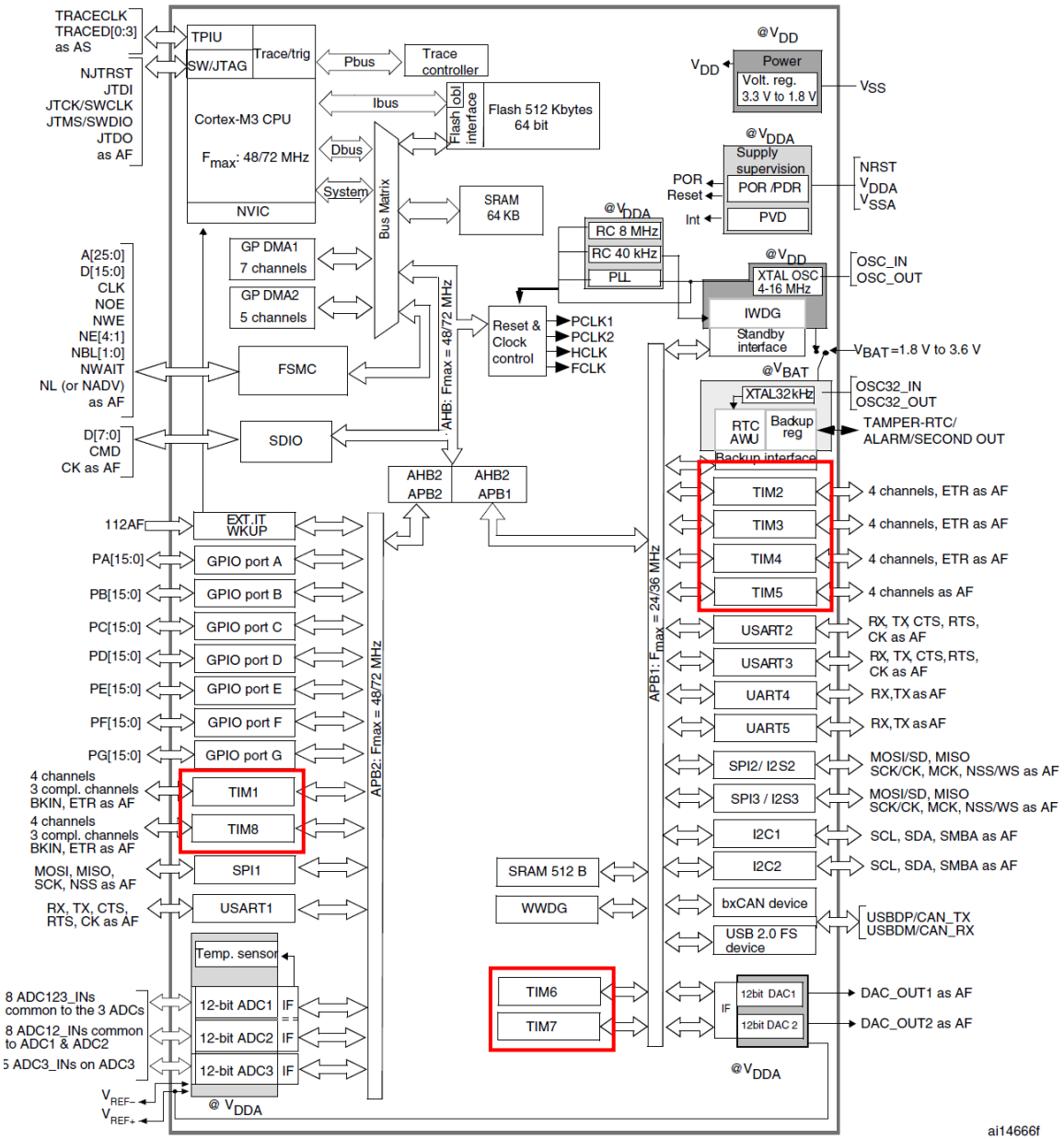


7.1 STM32定时器模块

STM32定时器分类比较表

定时器	基本定时器 (TIM6、TIM7)	通用定时器 TIMx(x=2~5)	高级定时器 (TIM1、TIM8)
计数器类型	16位，向上	16位， 向上、 向下、 向上/向下	16位， 向上、 向下、 向上/向下

7.1 STM32定时器模块



7.1 STM32定时器模块

1 7.1.1 通用定时器

2 7.1.2 基本定时器

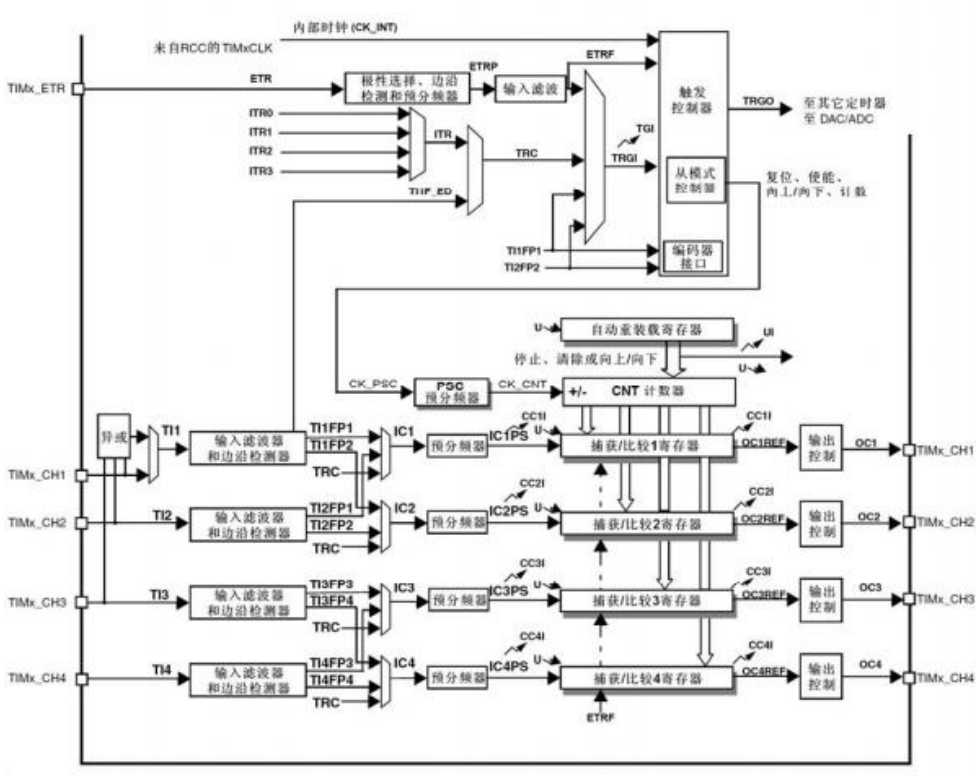
3 7.1.3 高级定时器

7.1.1 通用定时器

TIM2、TIM3、TIM4、TIM5为STM32的4个独立的**16位通用定时器**

主要功能：

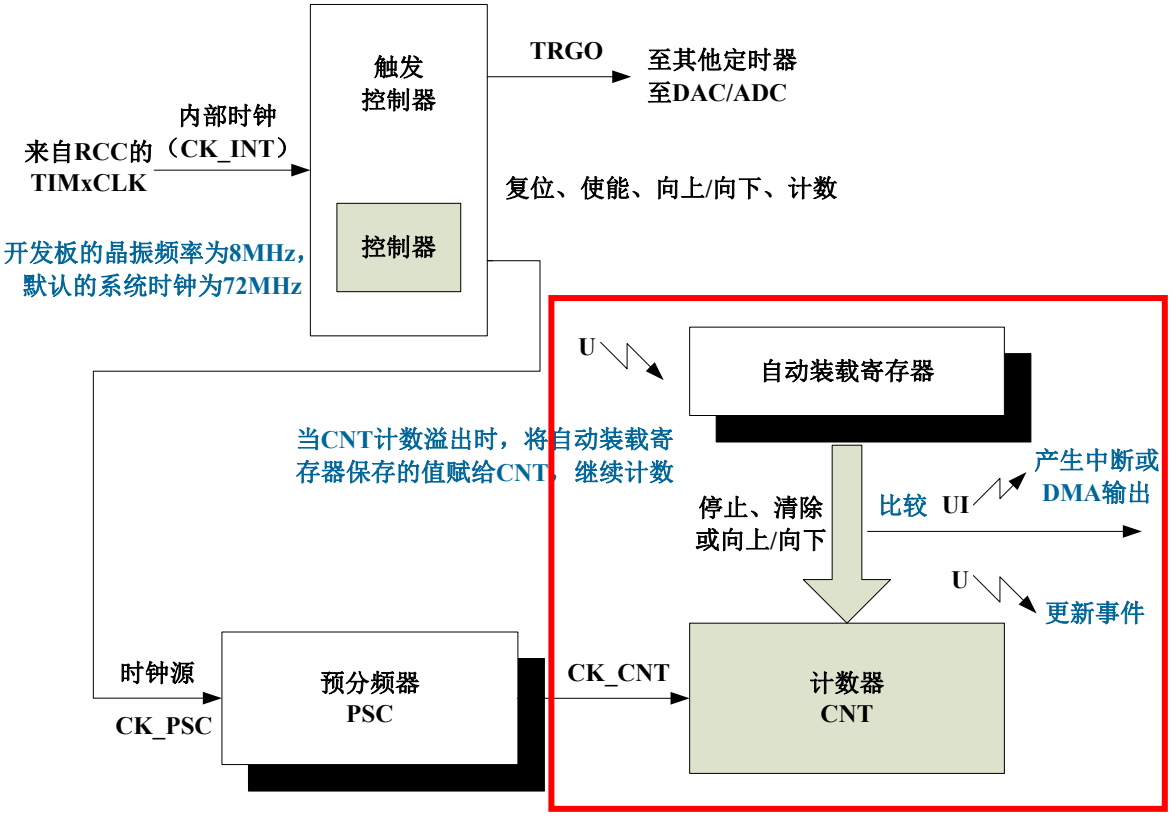
1. 定时
2. 测量输入信号的脉冲长度（输入捕获）
3. 输出所需波形（输出比较、产生PWM、单脉冲输出等）



通用定时器内部结构框图

7.1.1 通用定时器

- STM32通用定时器TIMx（x=2，3，4，5）主要由时钟源、时钟单元、捕获和比较通道等构成，核心是可编程预分频驱动的**16位自动装载计数器**。

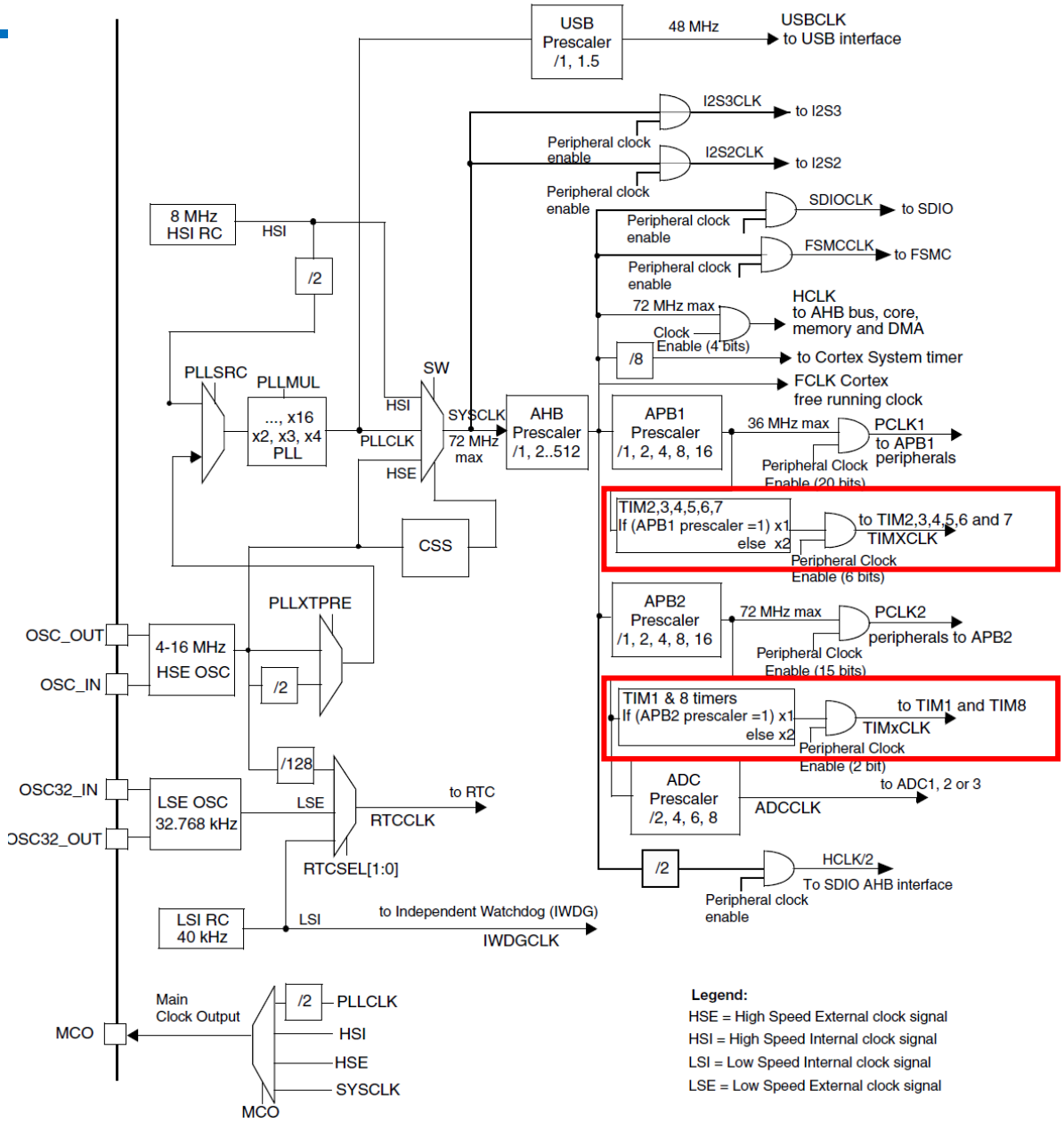


通用定时器内部简化图

7.1.1 通用定时器

1 时钟源

- 定时器TIM2~TIM7挂接在APB1上
- 定时器TIM1和TIM8挂接在APB2上



7.1.1 通用定时器

2 预分频器PSC

可以以1～65536之间的任意数值对时钟源CK_PSC的时钟频率进行分频，输出CK_CNT脉冲供计数器CNT进行计数。

$$f_{CK\_CNT} = f_{CK\_PSC} / (PSC+ 1)$$

- 预分频器内部本质是一个分频计数器，当输入M个时钟脉冲时，分频器输出1个脉冲（即分频系数为M）。
- 若PSC设置为P，分频计数器会从0计数到P（共P+1个输入脉冲），然后输出1个脉冲，因此实际分频系数是P+1

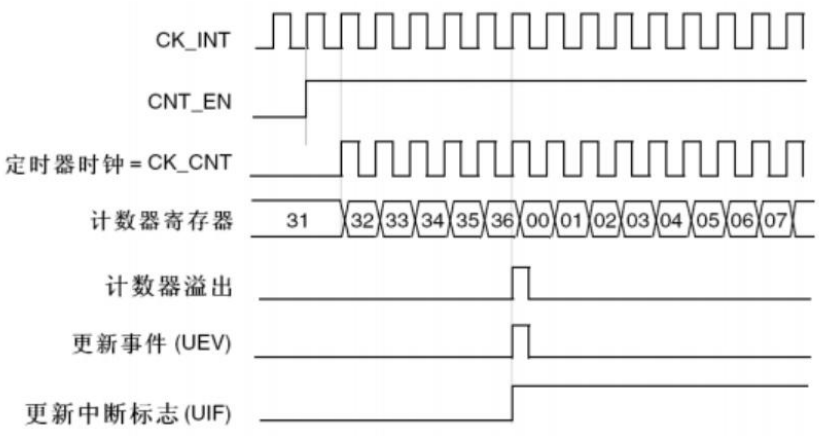
例： 输入时钟源频率为72MHz，要得到2MHz计数脉冲，PSC的取值

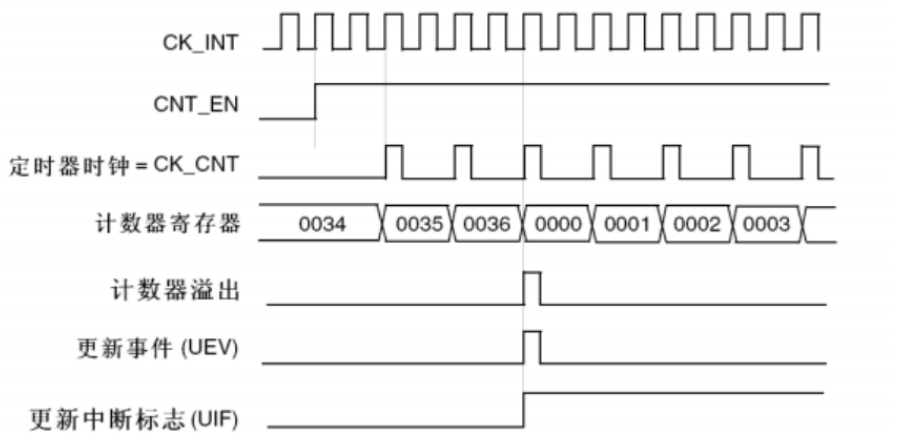
PSC = 72 / 2 – 1 = 35

7.1.1 通用定时器

3 计数器CNT

- TIMxCNT是一个16位的寄存器，计数范围为1～65535，可以向上计数、向下计数或向下向上双向计数。
- 要得到想要的计数值，需要对输入时钟频率进行分频。
- 当计数值达到设定值时，便产生溢出事件，溢出时产生中断或DMA请求，然后再由自动装载寄存器进行重新加载或更新。
- 计数器溢出中断属于软件中断，执行相应的定时器中断服务程序。





7.1.1 通用定时器

自动装载寄存器ARR

- **自动装载寄存器**通过存储一个预设的目标值，定义计数器的计数终点，进而控制定时器的更新频率。其取值与计数器CNT相同。
- 当计数器（CNT）的值达到 ARR 的值时，会触发更新，此时计数器通常会复位（如向上计数时回到 0，向下计数时回到 ARR），开始新一轮计数。

定时器的定时时间主要取决于定时周期和预分频系数，计算公式为：

定时时间=（ARR+1）×（预分频系数PSC+1）/输入时钟频率

或　　T=（TIM_Period +1）*（TIM_Prescaler +1）/TIMxCLK

例如：使用通用定时器定时1s。假设系统时钟为72MHz，通用定时器时钟TIMxCLK为72MHz，设置如下：

预分频系数PSC=36000-1；　ARR=2000-1；

定时时间=2000×36000/72000000=1s。

7.1.1 通用定时器

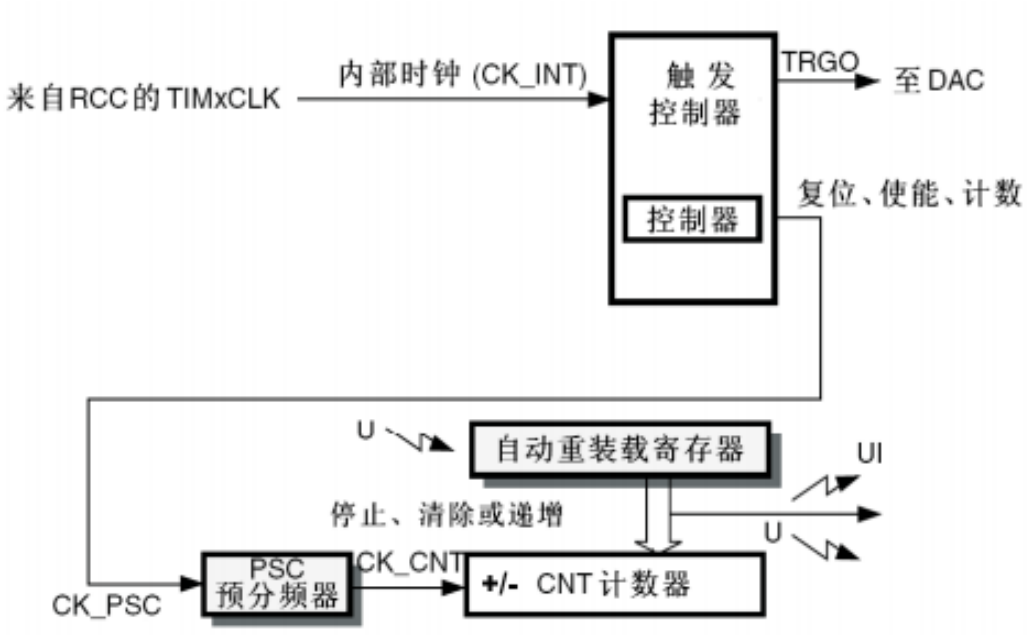
练习：系统时钟为72MHz，需要分别使用2个IO引脚控制2个LED小灯分别以1 ms和1s的时间间隔进行变化，只使用一个定时器，应该如何配置？

参考答案

1. 将定时器中断设置为1 ms，即（PSC+1）*（ARR+1）/72000000 = 0.001s，用于1 ms小灯的控制；如PSC = 71，ARR=999

2. 在定时器中断中加入变量进行中断次数计数，并进行判断，变量值达到999后，触发1s小灯的变化。

7.1.2 基本定时器



STM32有2个基本定时器TIM6和TIM7，可用作：

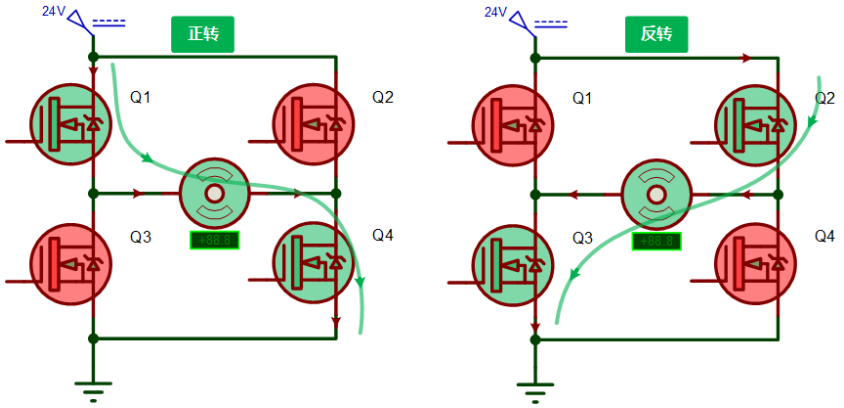
- **通用的16位计数器：**
- **产生DAC触发信号**

基本定时器的计数模式只有向上计数模式。

7.1.3 高级定时器

TIM1和TIM8是STM32的2个16位的高级定时器

1. 兼容通用定时器的所有基础功能



2. 互补 PWM 输出与死区控制

- **互补 PWM 输出**：每个高级定时器通道可同时输出**主 PWM**和**互补 PWM,**分别驱动桥式电路的上桥臂和下桥臂。
- **死区控制（Dead Time）**：在互补 PWM 输出时，可插入一段“死区时间”，确保上、下桥臂的功率器件不会同时导通。

3. 硬件级安全保护：刹车功能（Break）

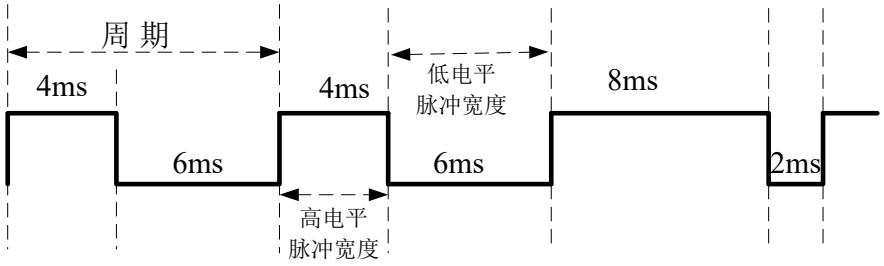
集成了**刹车保护机制**，用于在故障时快速切断功率输出。

7.2 PWM

PWM（Pulse Width Modulation，脉冲宽度调制）是一种利用脉冲宽度即占空比实现对模拟信号进行控制的技术，即是对模拟信号电平进行数字表示的方法。

广泛应用于电力电子技术中，比如PWM控制技术在逆变电路中的应用；

PWM还应用于直流电机调速，如变频空调的交直流变频调速，除实现调速外，还具有节能等特性。

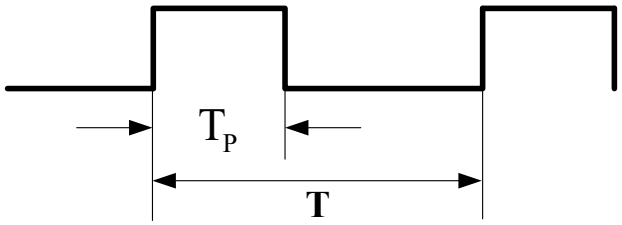


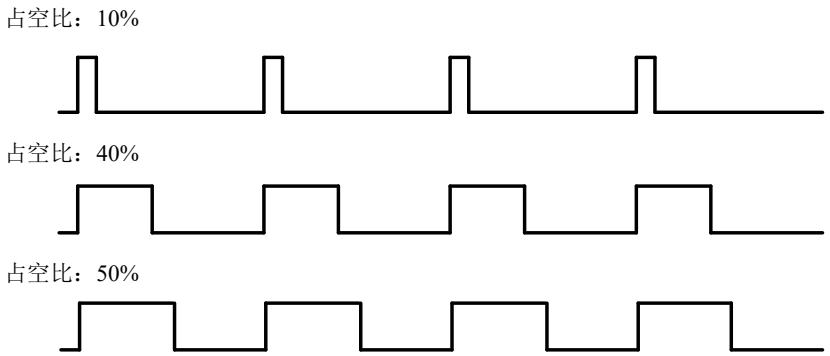
周期为10ms（频率为100Hz）的PWM波形

7.2 PWM

占空比（Duty Cycle），是指在一个周期内，高电平时间占整个信号周期的百分比，即高电平时间与周期的比值：

$$占空比=T_p/T$$





不同占空比的PWM波形

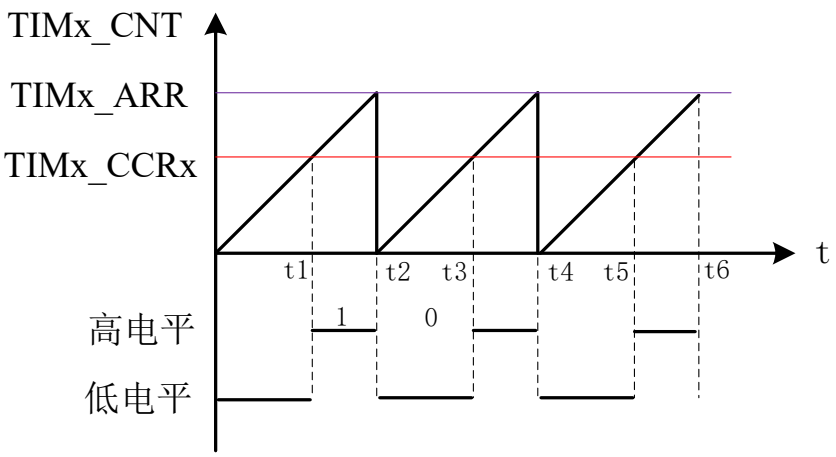
7.2 PWM

7.2.1 PWM的工作原理

7.2.1 STM32定时器PWM的工作原理

- STM32的定时器除了TIM6和TIM7，其他定时器都可以用来产生PWM输出；
- 高级定时器TIM1和TIM8可以同时产生多达7路的PWM输出；
- 通用定时器能同时产生多达4路的PWM输出；
- STM32中每个定时器有4个输入通道：TIMx_CH1~TIMx_CH4；
- 每个通道对应1个捕获/比较寄存器TIMx_CRRx，将寄存器值和计数器值相比较，通过比较结果输出高低电平，从而得到PWM信号；
- 脉冲宽度调制模式可以产生一个由TIMx_ARR寄存器确定频率、由TIMx_CCRx寄存器确定占空比的信号。

7.2.1 STM32定时器PWM的工作原理



- 在PWM的一个周期内，定时器从0开始向上计数，在0-t1时间段，定时器计数器TIMx_CNT值小于TIMx_CCRx值，输出低电平；
- 在t1-t2时间段，定时器计数器TIMx_CNT值大于TIMx_CCRx值，输出高电平；
- 当定时器计数器的值TIMx_CNT达到ARR时，定时器溢出，重新从0开始向上计数，如此循环。

7.3 SysTick定时器

- Cortex-M3内核中的SysTick定时器，是一个24位的从重载值向下递减到0的计数器，是NVIC的一部分，根植于NVIC；
- 常用于精确定时，为操作系统提供必要的时钟节拍，为RTOS的任务调度提供一个有节奏的“心跳”。

✓ 精确延时，在多任务操作系统中为系统提供时间基准（时基）；
✓ 任务切换，为每个任务分配时间片。

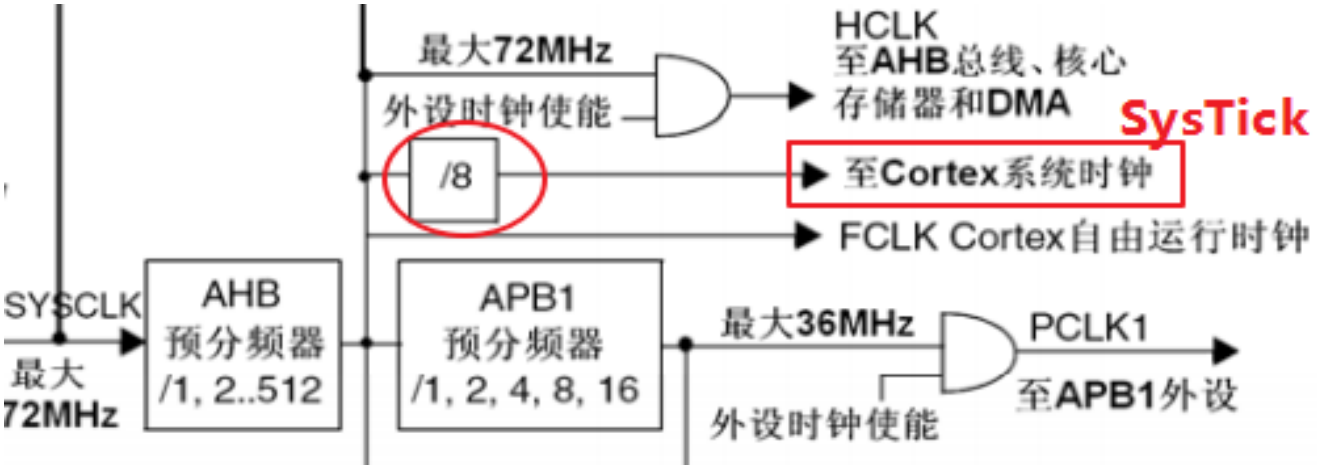
SysTick定时器一共有四个寄存器，分别是

- SysTick控制及状态寄存器CTRL
- SysTick重装载数值寄存器LOAD
- SysTick当前数值寄存器VAL
- SysTick校准数值寄存器CALIB。

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	STK_CTRL	Reserved															COUNTFLAG	Reserved													CLKSOURCE	TICK INT	ENABLE
	Reset Value																0														0	0	0
0x04	STK_LOAD	Reserved								RELOAD[23:0]																							
	Reset Value									0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x08	STK_VAL	Reserved								CURRENT[23:0]																							
	Reset Value									0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x0C	STK_CALIB	Reserved								TENMS[23:0]																							
	Reset Value									0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

7.3 SysTick定时器

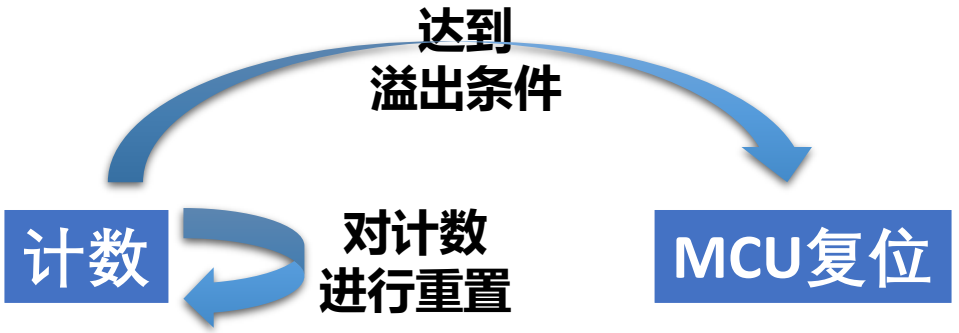
- SysTick定时器的时钟源可以是内部时钟（FCLK）或者是外部时钟，系统默认的SysTick定时器是由AHB时钟（HCLK）8分频得到的，即SysTick的频率为9MHz；
- SysTick定时器从设定的初值计数到0时，会**自动重装初值**继续计数，同时**触发中断**，因此，只需确定计数的次数就可以精确得到延迟时间。



7.4 看门狗

- 当微控制器受到外部干扰或程序中出现不可预知的逻辑故障导致**应用程序脱离正常的执行流程**时（俗称程序跑飞），在一定的时间间隔内使系统**复位**，回到初始状态；
- 看门狗设计是用来监视MCU程序运行状态的，是确保系统可靠稳定运行的一种有效措施。

WatchDog看门狗

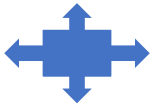


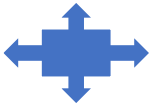
7.4 看门狗

维度	独立看门狗（IWDG）	窗口看门狗（WWDG）
时钟源	独立 LSI（40kHz）	APB1 时钟
喂狗约束	超时前任意时间	“窗口值 ~最大值” 区间
定时范围	宽（ms~s）	窄（μs~ms）
核心作用	系统级故障监控（防死循环）	程序实时性监控（防流程异常）
独立性	完全独立于系统时钟	依赖系统时钟

7.5 定时器模块的HAL库接口函数及应用

目 录


7.5.1定时器HAL库接口函数


7.5.2定时器HAL库应用实例

7.5.1 定时器HAL库接口函数

定时器模块的HAL库常用接口函数可分为七大类：

类型	函数及功能描述
基本定时器功能函数	HAL_TIM_Base_Init(TIM_HandleTypeDef *htim); 功能描述：基本定时器初始化函数
	void HAL_TIM_Base_MspInit(TIM_HandleTypeDef *htim); 功能描述：基本定时器硬件初始化配置函数，该函数被HAL库内部调用
定时器输出比较功能函数	HAL_TIM_OC_Init(TIM_HandleTypeDef *htim); 功能描述：定时器输出比较的初始化函数
	HAL_TIM_OC_Start(TIM_HandleTypeDef *htim, uint32_t Channel); 功能描述：定时器输出比较轮询方式的启动函数
	HAL_TIM_OC_Start_IT(TIM_HandleTypeDef *htim, uint32_t Channel); 功能描述：定时器输出比较中断方式的启动函数
	HAL_TIM_OC_Start_DMA(TIM_HandleTypeDef *htim, uint32_t Channel, uint32_t *pData, uint16_t Length); 功能描述：定时器输出比较DMA方式的启动函数
定时器PWM函数	HAL_TIM_PWM_Init(TIM_HandleTypeDef *htim); 功能描述：定时器PWM初始化函数
	HAL_TIM_PWM_Start(TIM_HandleTypeDef *htim, uint32_t Channel); 功能描述：启动定时器PWM轮询方式的函数
	HAL_TIM_PWM_Start_IT(TIM_HandleTypeDef *htim, uint32_t Channel); 功能描述：启动定时器PWM中断方式的函数
	HAL_TIM_PWM_Start_DMA(TIM_HandleTypeDef *htim, uint32_t Channel, uint32_t *pData, uint16_t Length); 功能描述：启动定时器PWM的DMA方式的函数

7.5.1 定时器HAL库接口函数

定时器模块的HAL库常用接口函数可分为七大类：

类型	函数及功能描述
定时器输入捕获函数	HAL_TIM_IC_Init(TIM_HandleTypeDef *htim); 功能描述：定时器输入捕获初始化函数
	HAL_TIM_IC_Start(TIM_HandleTypeDef *htim, uint32_t Channel); 功能描述：启动定时器输入捕获轮询方式的函数
	HAL_TIM_IC_Start_IT(TIM_HandleTypeDef *htim, uint32_t Channel); 功能描述：启动定时器输入捕获中断方式的函数
	HAL_TIM_IC_Start_DMA(TIM_HandleTypeDef *htim, uint32_t Channel, uint32_t *pData, uint16_t Length); 功能描述：启动定时器输入捕获的DMA方式的函数
定时器中断及回调函数	void HAL_TIM_IRQHandler(TIM_HandleTypeDef *htim); 功能描述：定时器中断处理函数
	void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim); 功能描述：定时器定时/更新中断回调函数，用户在该函数内编写实际的中断处理程序
	void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim); 功能描述：定时器输入捕获回调函数
	void HAL_TIM_PWM_PulseFinishedCallback(TIM_HandleTypeDef *htim); 功能描述：定时器PWM结束回调函数

7.5.1 定时器HAL库接口函数

定时器模块的HAL库常用接口函数可分为七大类：

类型	函数及功能描述
定时器功能配置函数	HAL_TIM_OC_ConfigChannel(TIM_HandleTypeDef *htim, TIM_OC_InitTypeDef* sConfig, uint32_t Channel); 功能描述：定时器输出比较配置函数
	HAL_TIM_PWM_ConfigChannel(TIM_HandleTypeDef *htim, TIM_OC_InitTypeDef* sConfig, uint32_t Channel); 功能描述：定时器PWM配置函数
	HAL_TIM_IC_ConfigChannel(TIM_HandleTypeDef *htim, TIM_IC_InitTypeDef* sConfig, uint32_t Channel); 功能描述：定时器输入捕获配置函数
定时器状态函数	HAL_TIM_OC_GetState(TIM_HandleTypeDef *htim); 功能描述：获取定时器输出比较的状态函数
	HAL_TIM_PWM_GetState(TIM_HandleTypeDef *htim); 功能描述：获取定时器PWM的状态函数
	HAL_TIM_IC_GetState(TIM_HandleTypeDef *htim); 功能描述：获取定时器输入捕获的状态函数

7.5.1 定时器HAL库接口函数

1．定时器初始化配置函数

HAL_TIM_Base_Init()

```
HAL_StatusTypeDef
HAL_TIM_Base_Init(TIM_HandleTypeDef *htim)
{
  /* 检测htim句柄是否有效 */
 if(htim == NULL)
 {
  return HAL_ERROR;
 }
  /* 采用断言方式检测参数是否有效 */
 assert_param(IS_TIM_INSTANCE(htim->Instance));
 assert_param(IS_TIM_COUNTER_MODE(htim-
>Init.CounterMode));
```

```
HAL_StatusTypeDef
HAL_TIM_Base_Init(TIM_HandleTypeDef *htim)
{
  /* 检测htim句柄是否有效 */
 if(htim == NULL)
 {
  return HAL_ERROR;
 }
  /* 采用断言方式检测参数是否有效 */
 assert_param(IS_TIM_INSTANCE(htim->Instance));
 assert_param(IS_TIM_COUNTER_MODE(htim-
>Init.CounterMode));
 assert_param(IS_TIM_CLOCKDIVISION_DIV(htim-
>Init.ClockDivision));
 assert_param(IS_TIM_AUTORELOAD_PRELOAD(htim-
>Init.AutoReloadPreload));
```

7.5.1 定时器HAL库接口函数

1．定时器初始化配置函数 HAL_TIM_Base_Init()

解析：函数源码中主要涉及到两个重要的函数HAL_TIM_Base_MspInit(htim)和TIM_Base_SetConfig( )，HAL库每个外设模块都设计有单独的底层初始化回调函数。

```
if(htim->State == HAL_TIM_STATE_RESET)
 {
   /* 取消上锁*/
  htim->Lock = HAL_UNLOCKED;
   /*初始化底层硬件：GPIO、CLOCK、 NVIC */
  HAL_TIM_Base_MspInit(htim);
 }
  /*设置TIM状态 */
 htim->State= HAL_TIM_STATE_BUSY;
  //将状态设置为BUSY
  /* 配置TIM的基本参数 */
 TIM_Base_SetConfig(htim->Instance, &htim->Init);
  /* 初始化TIM 状态，TIM就绪 */
 htim->State= HAL_TIM_STATE_READY;
  //将状态设置为READY
 return HAL_OK;
}
```

7.5.1 定时器HAL库接口函数

HAL库在TIM_TypeDef的基础上封装了一个结构体TIM_HandleTypeDef（定时器句柄）

```
Typedef struct
{
 TIM_TypeDef    *Instance;  /*TIM寄存器的实例化*/
 TIM_Base_InitTypeDef    Init;
    /*配置定时器的基本参数 */
 HAL_TIM_ActiveChannel    Channel;  /*配置定时器通道 */
 DMA_HandleTypeDef       *hdma[7];  /*配置DMA */
 HAL_LockTypeDef          Lock;      /*上锁*/
 __IO HAL_TIM_StateTypeDef   State;
    /*定时器操作状态*/
} TIM_HandleTypeDef;
```

7.5.1 定时器HAL库接口函数

TIM_HandleTypeDef结构体又包含了定时器初始化结构体TIM_Base_InitTypeDef

```
typedef struct
{
  uint32_t  Prescaler; /*定时器预分频系数，其值范围在0x0000 ~ 0xFFFF之间 */
  uint32_t  CounterMode;    /*定时器计数模式，向上计数、向下计数和中心对齐模式 */
  uint32_t  Period;         /*定时器定时周期，其值范围为0x0000 ~ 0xFFFF */
  uint32_t  ClockDivision;    /* 定时器时钟分频因子*/
  uint32_t  RepetitionCounter;  /*重复计数器，仅用于高级定时器TIM1和TIM8*/
  uint32_t  AutoReloadPreload;  /*定时器自动重装载值*/
} TIM_Base_InitTypeDef;
```

7.5.1 定时器HAL库接口函数

2．启动定时器函数

HAL_TIM_Base_Start()

解析：该数只有一个输入参数，为TIM_HandleTypeDef类型的结构体指针变量，程序中主要是调用__HAL_TIM_ENABLE(htim)函数使能相应的定时器，该函数实质是一个宏定义：

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

配置定时器CR1的使能位为“1”，返回HAL_OK，表示初始化成功。

```
HAL_StatusTypeDef
HAL_TIM_Base_Start(TIM_HandleTypeDef *htim)
{
 uint32_t  tmpsmcr;
  /* 使用断言，检测参数是否正确 */
 assert_param(IS_TIM_INSTANCE(htim->Instance));
  /*设置定时器的状态 */
 htim->State = HAL_TIM_STATE_BUSY;  /* 使能外设*/
 tmpsmcr = htim->Instance->SMCR & TIM_SMCR_SMS;
 if (!IS_TIM_SLAVEMODE_TRIGGER_ENABLED(tmpsmcr))
 {   __HAL_TIM_ENABLE(htim);
 }
  /* 定时器就绪*/
 htim->State = HAL_TIM_STATE_READY;
  /* 返回HAL_OK，初始化成功*/
 return HAL_OK;
}
```

7.5.1 定时器HAL库接口函数

3．启动定时器中断模式定时函数 HAL_TIM_Base_Start_IT()

解析：该启动函数的核心代码为：__HAL_TIM_ENABLE(htim)，跳转到该函数，可以发现该函数其实是一个宏定义：

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

该宏定义的作用是将CR1的计数器使能位设置为“1”。

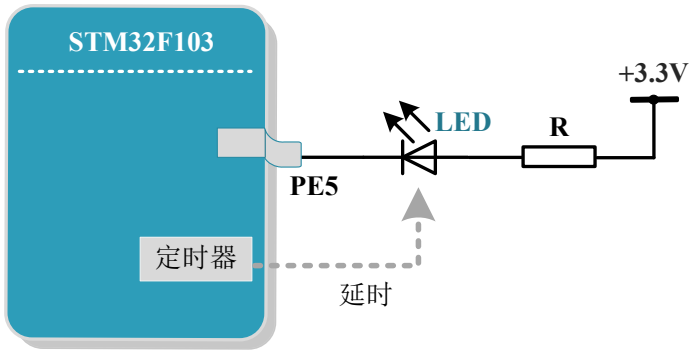
```
HAL_StatusTypeDef HAL_TIM_Base_Start_IT(TIM_HandleTypeDef *htim)
{
  uint32_t tmpsmcr;
   /* 使用断言，进行参数状态检测*/
  assert_param(IS_TIM_INSTANCE(htim->Instance));
   /*使能定时器更新中断*/
  __HAL_TIM_ENABLE_IT(htim, TIM_IT_UPDATE);
   /* 使能外设*/
  tmpsmcr = htim->Instance->SMCR & TIM_SMCR_SMS;
  if (!IS_TIM_SLAVEMODE_TRIGGER_ENABLED(tmpsmcr))
  {   __HAL_TIM_ENABLE(htim);
  }
  /* 返回HAL_OK*/
  return HAL_OK;
}
```

7.5.2 定时器HAL库应用实例

功能

利用基于HAL库采用定时器TIM3产生精确延时1s

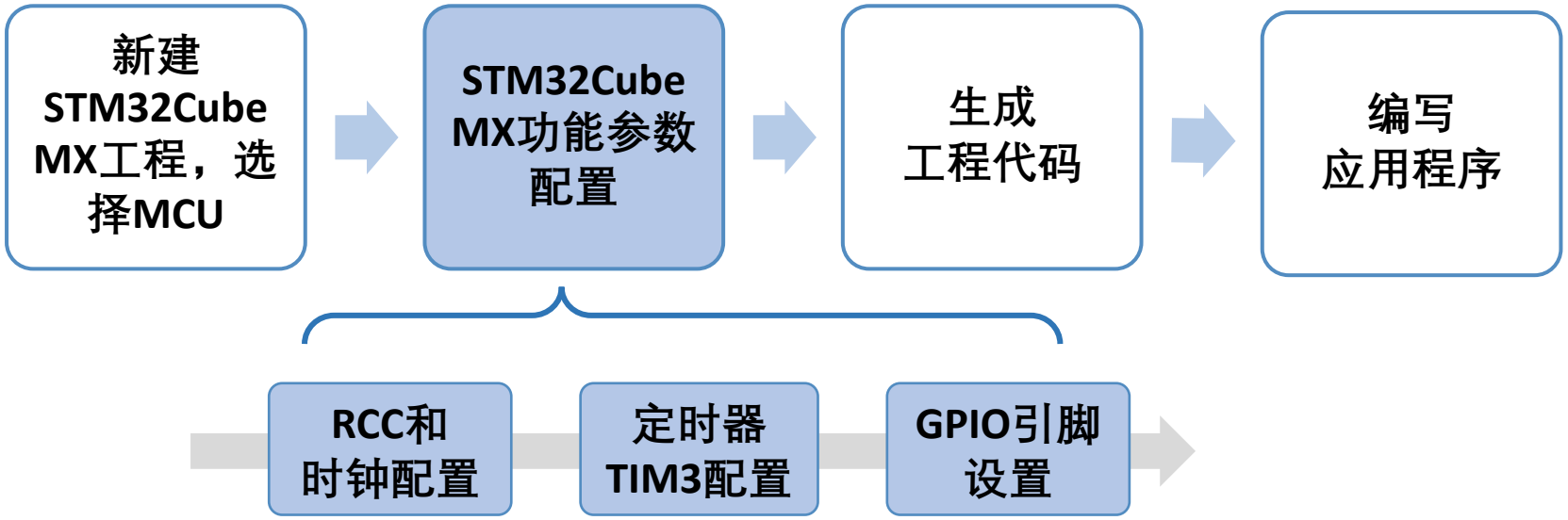
硬件设计



利用定时器TIM3产生1s的定时中断，通过LED指示灯进行状态显示，LED连接在STM32F103的PE5引脚上，低电平有效

7.5.2 定时器HAL库应用实例

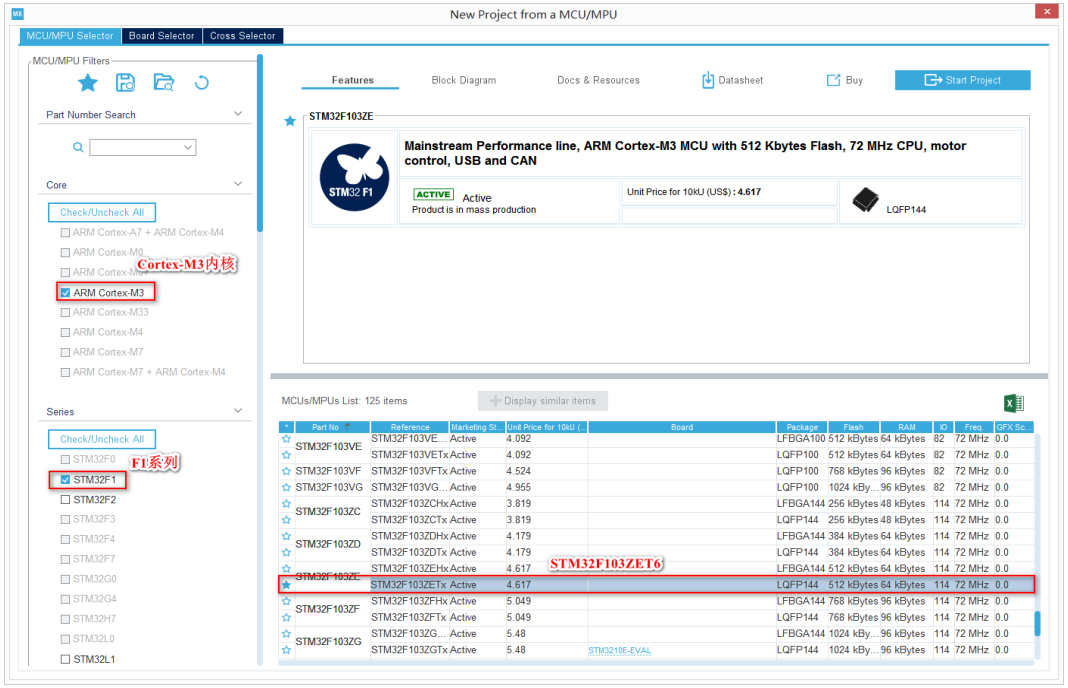
软件设计流程



7.5.2 定时器HAL库应用实例

软件设计——新建STM32CubeMX工程，选择MCU

新建一个STM32CubeMX工程，选择MCU，这里采用STM32F103ZET6芯片。

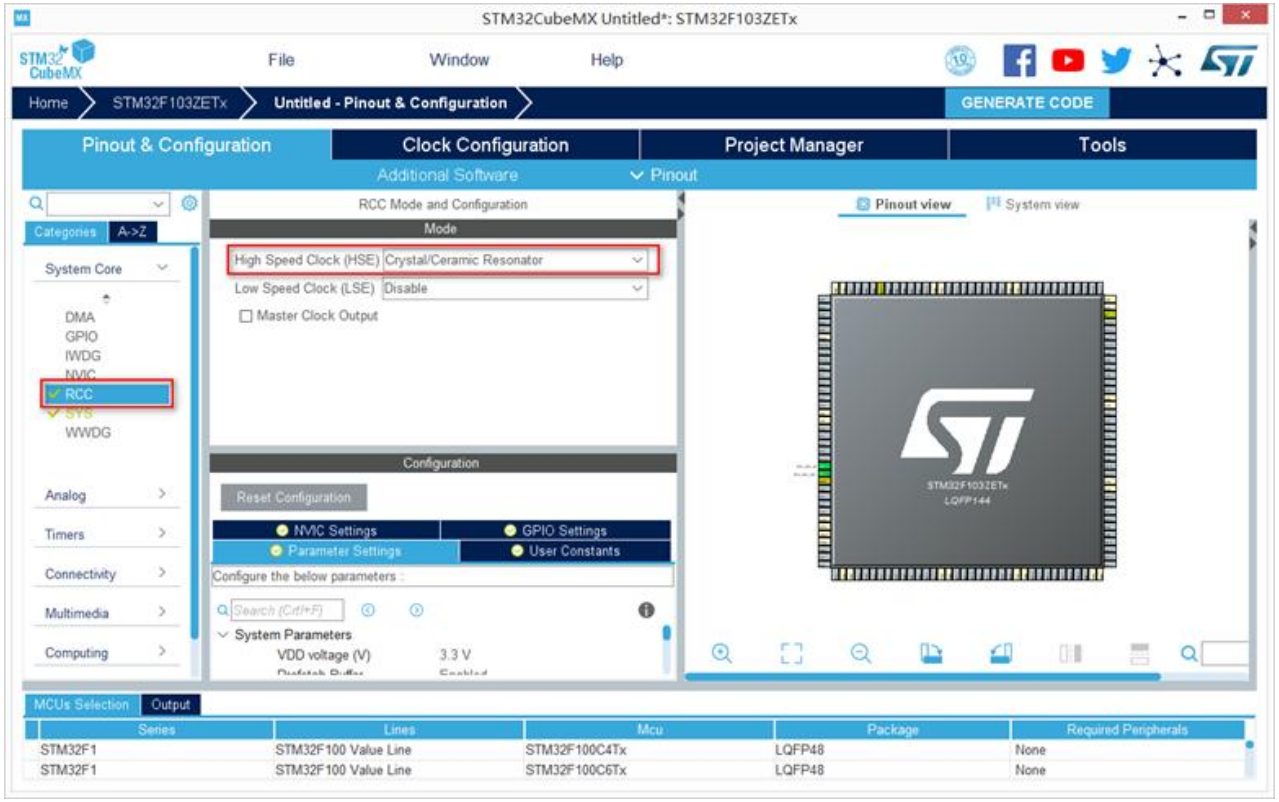


7.5.2 定时器HAL库应用实例

软件设计——STM32CubeMX功能参数配置

RCC配置

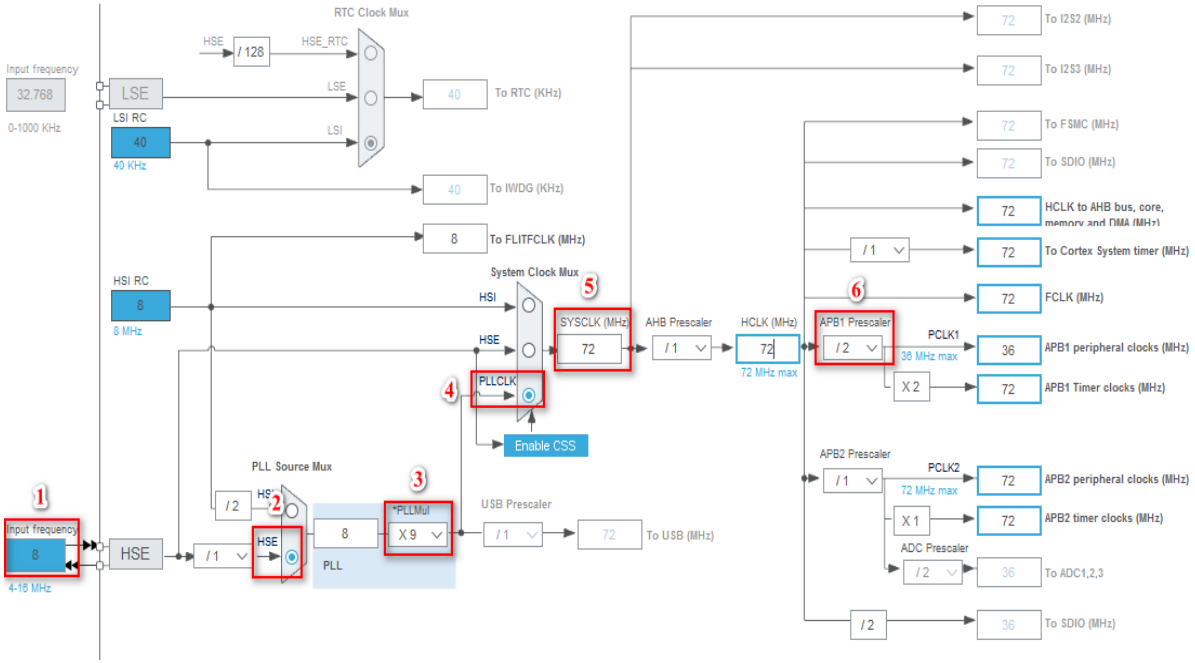
HSE选择“Crystal/Ceramic Resonator”（晶振/陶瓷谐振器），LSE选择“Disable”



7.5.2 定时器HAL库应用实例

软件设计——STM32CubeMX功能参数配置

时钟配置



配置STM32的时钟系统，系统时钟配置为72MHz，APB2为72MHz，配置APB1为36MHz。

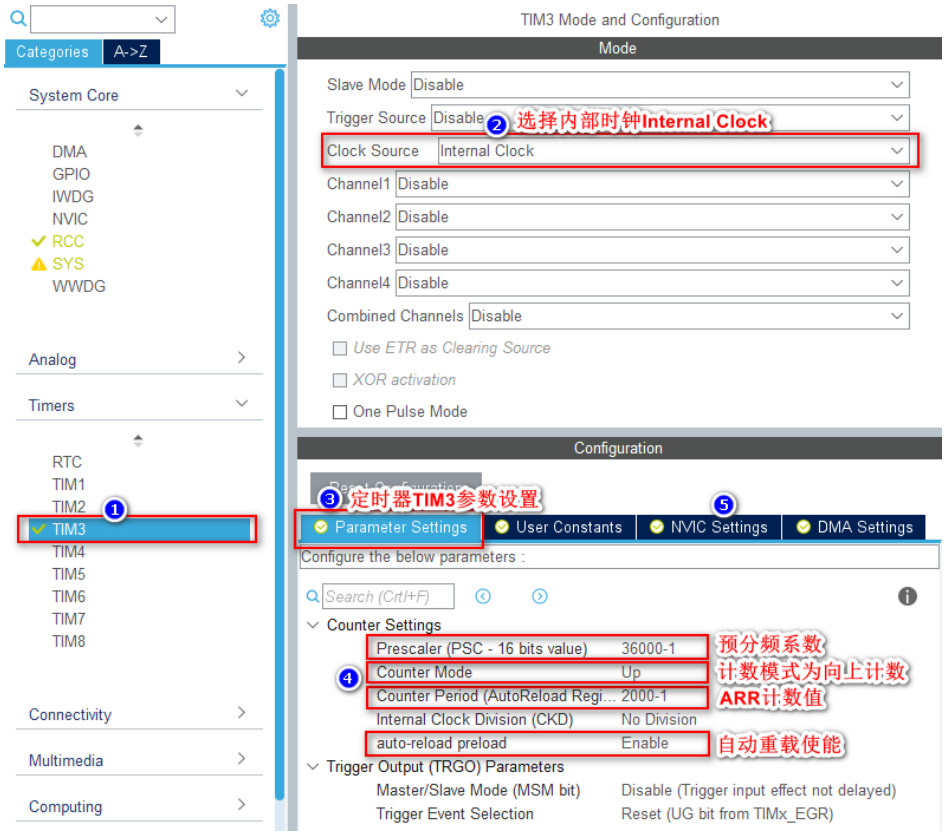
7.5.2 定时器HAL库应用实例

软件设计——STM32CubeMX功能参数配置

- “Clock Source”设置为“Internal Clock”（内部时钟）；
- 预分频系数“Prescaler”设置为“36000-1”；“Counter Mode”设置为“Up”；
- 计数值“Counter Period”设置为“2000-1”；
- “auto-reload preload”设置为“Enable”。

则定时器TIM3精确定时1s。

定时器TIM3配置

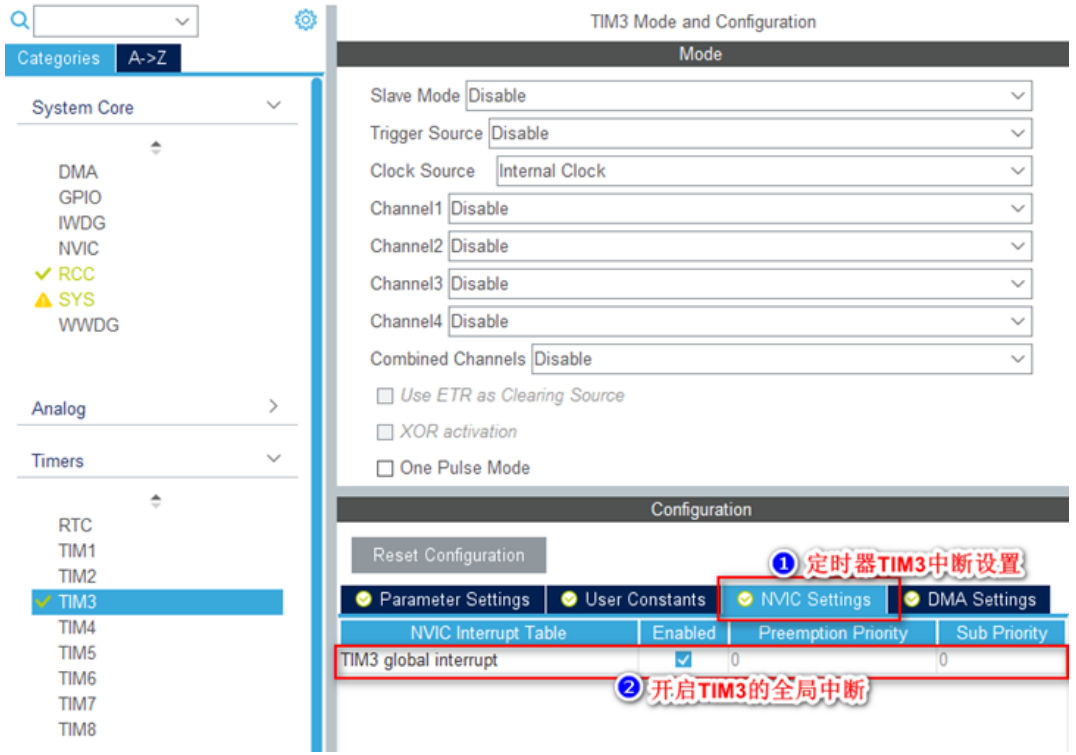


7.5.2 定时器HAL库应用实例

软件设计——STM32CubeMX功能参数配置

- 在定时器TIM3参数配置“Configuration”选项配置页的“NVIC Settings”中使能定时器TIM3的全局中断

定时器TIM3配置



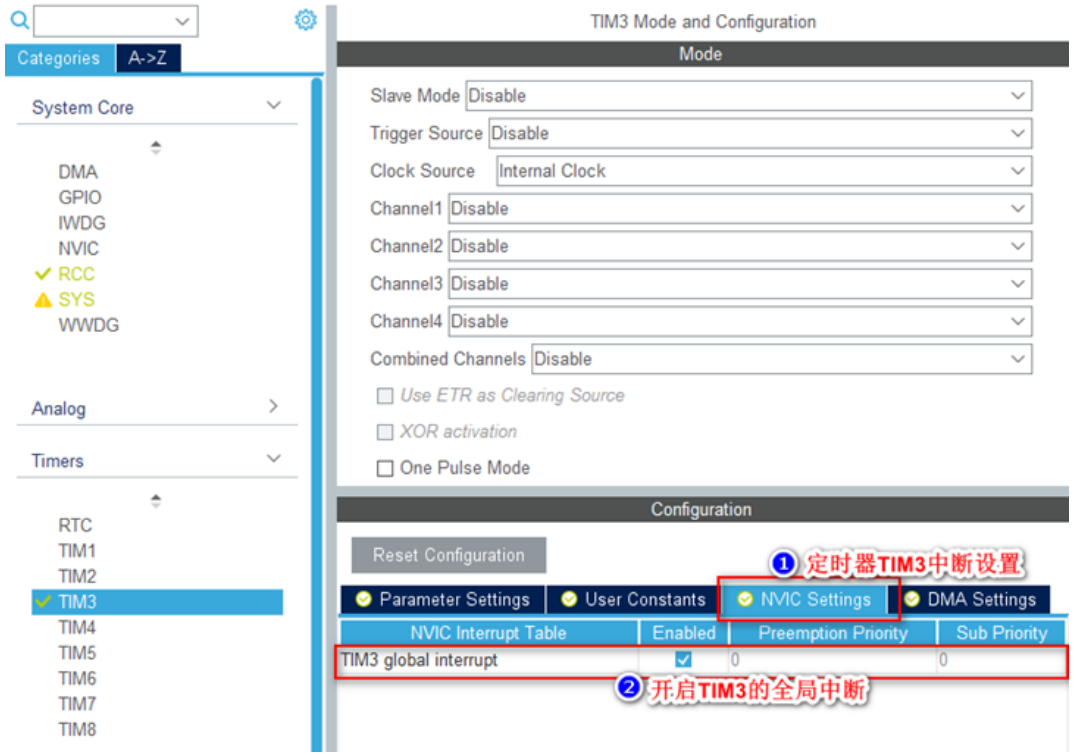
使能TIM3的全局中断

7.5.2 定时器HAL库应用实例

软件设计——STM32CubeMX功能参数配置

- 在定时器TIM3参数配置“Configuration”选项配置页的“NVIC Settings”中使能定时器TIM3的全局中断

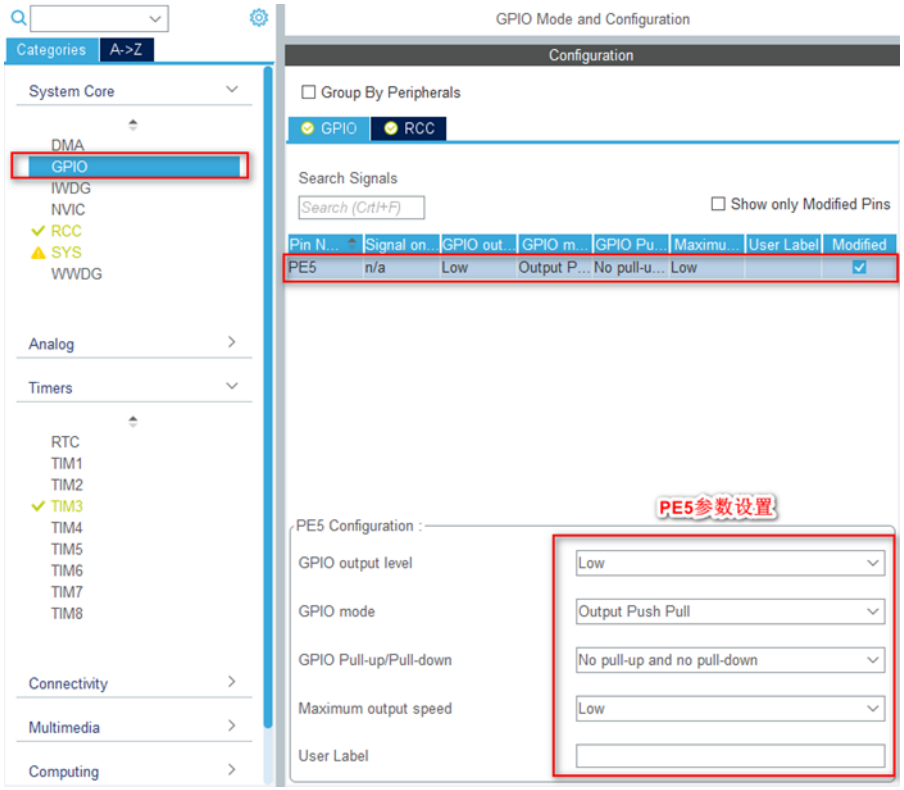
定时器TIM3配置



使能TIM3的全局中断

7.5.2 定时器HAL库应用实例

软件设计——STM32CubeMX功能参数配置



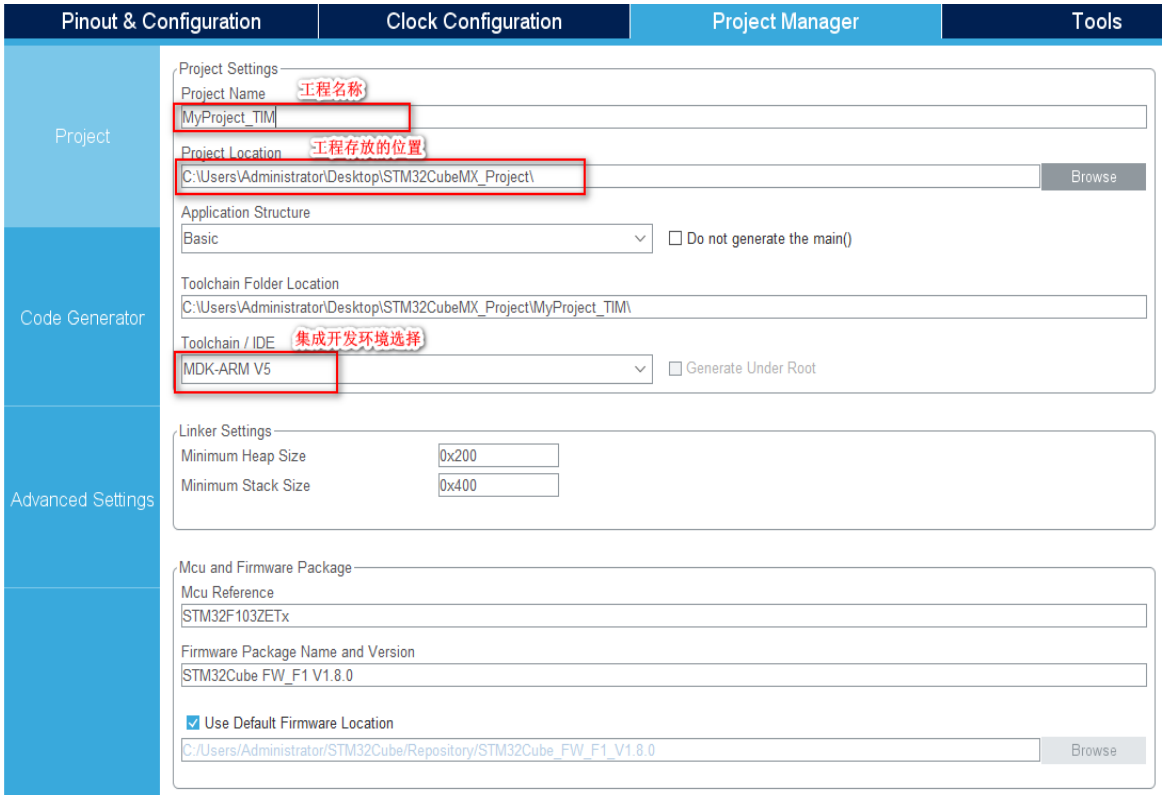
GPIO引脚设置

- 设置PE5引脚为“GPIO_Output”，用于本例的LED指示灯显示，并在“GPIO Mode and Configuration”中配置相应的参数

7.5.2 定时器HAL库应用实例

软件设计——生成工程代码

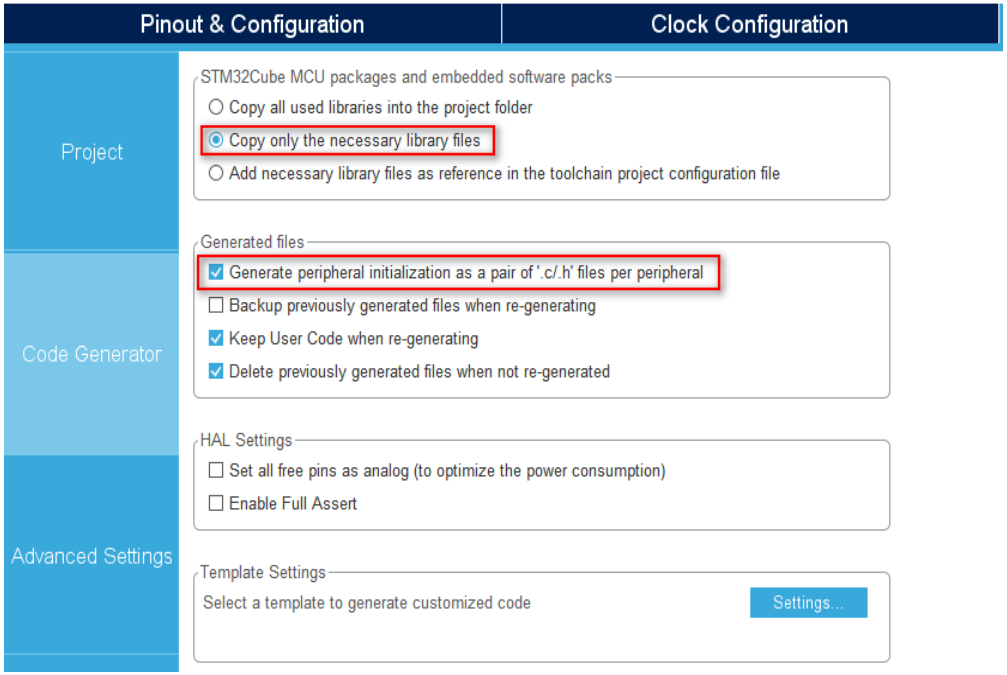
工程相关参数配置



配置工程名称、工程保存位置、选择“MDK-ARM V5”编译器等，生成工程代码

7.5.2 定时器HAL库应用实例

软件设计——生成工程代码

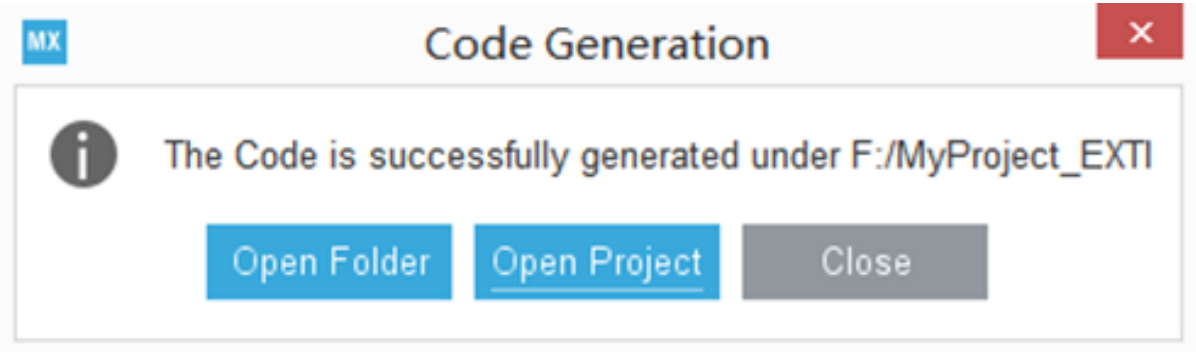


在“Code Generator”选项栏中找到“Generated files”框，勾选“Generate peripheral initialization as a pair of ’.c/.h’ files per IP”，将外设初始化的代码设置生成为独立的.c源文件和.h头文件。

7.5.2 定时器HAL库应用实例

软件设计——生成工程代码

通过STM32CubeMX的菜单栏中的“Generate Code”生成工程代码，生成代码后，会提示是否打开该工程窗口



7.5.2 定时器HAL库应用实例

软件设计——编写应用程序

在main.c文件中的/*USER CODE BEGIN2 */和/*USER CODE END2 */之间添加开启定时器TIM3中断的程序

```
MX_TIM3_Init();
/* USER CODE BEGIN 2 */
HAL_TIM_Base_Start_IT(&htim3);
//启动定时器TIM3定时中断
/* USER CODE END 2 */
```

在mian.c文件的/*USER CODE BEGIN4 */和/*USER CODE END4 */之间添加TIM3的中断回调函数

```
/* USER CODE BEGIN 4 */
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
   if(htim->Instance == htim3.Instance)
   {   HAL_GPIO_TogglePin(GPIOE,GPIO_PIN_5);
   }
}
/* USER CODE END 4 */
```