第7章 STM32通用定时器

退出

7.1 STM32F103定时器概述

STM32F103定时器相比于传统的51单片机要完善和复杂得多，它是专为工业控制应用量身定做，具有延时、频率测量、PWM输出、电机控制及编码接口等功能。

STM32F103微控制器内部集成了多个可编程定时器，可以分为基本定时器（TIM6和TIM7）、通用定时器（TIM2~TIM5）和高级定时器（TIM1、TIM8）3种类型。从功能上看，基本定时器的功能是通用定时器的子集，而通用定时器的功能又是高级定时器的一个子集，各类定时器功能描述见表。

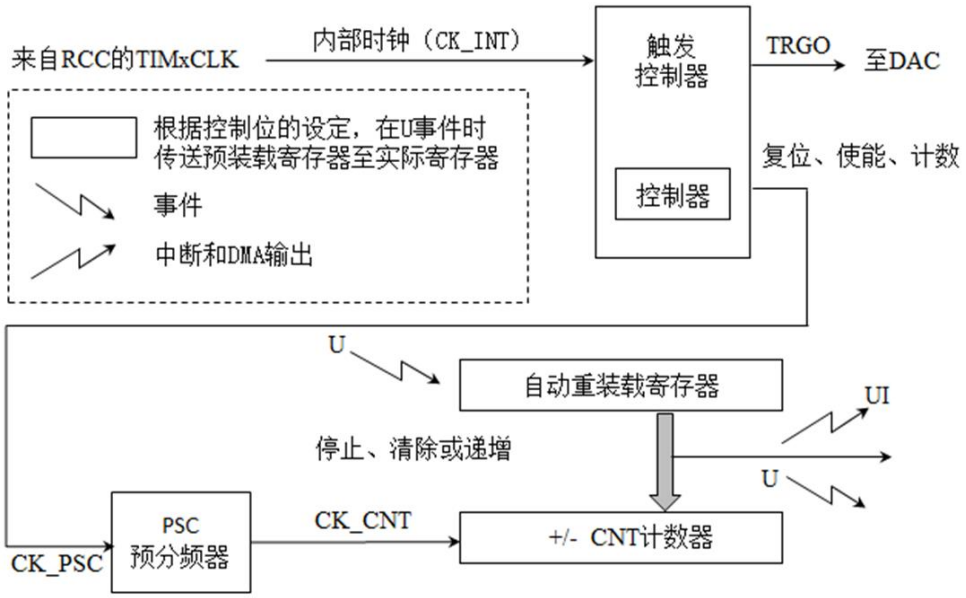
STM32F103定时器功能查询表

主要特点	基本定时器	通用定时器	高级定时器
内部时钟CK_INT来源	APB1分频器	APB1分频器	APB2分频器
预分频器的位数(分频范围)	16位 (1~65536)	16位 (1~65536)	16位 (1~65536)
计数器的位数（计数范围）	16位 (1~65536)	16位 (1~65536)	16位 (1~65536)
更新中断和DMA	√	√	√
计数方向	↑	↑、↓、↑↓	↑、↓、↑↓
外部事件计数	x	√	√
定时器触发或级联	x	√	√
4个独立捕获/比较通道	x	√	√
单脉冲输出方式	x	√	√
正交编码器输入	x	√	√
霍尔传感器输入	x	√	√
刹车信号输入	x	x	√
带死区的PWM互补输出	x	x	√

退出

7.2 基本定时器

STM32F103基本定时器 TIM6 和 TIM7 各包含一个16位自动装载计数器，由各自的可编程预分频器驱动。这2个定时器是互相独立的，不共享任何资源。



基本定时器的主要特性

TIM6和TIM7定时器的主要功能包括：

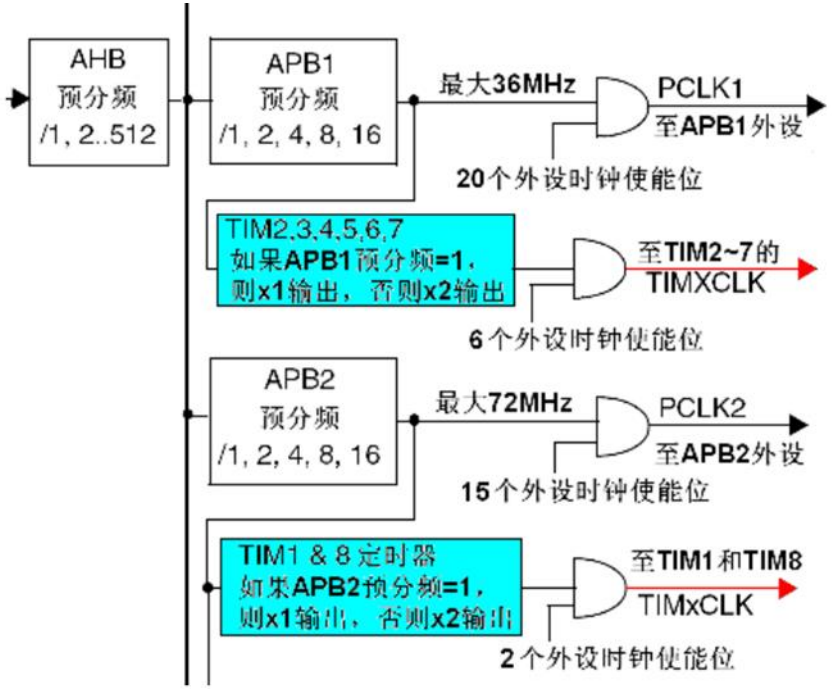
- 16位自动重装载累加计数器
- 16位可编程(可实时修改)预分频器
- 触发DAC的同步电路
- 在更新事件(计数器溢出)时产生中断/DMA请求

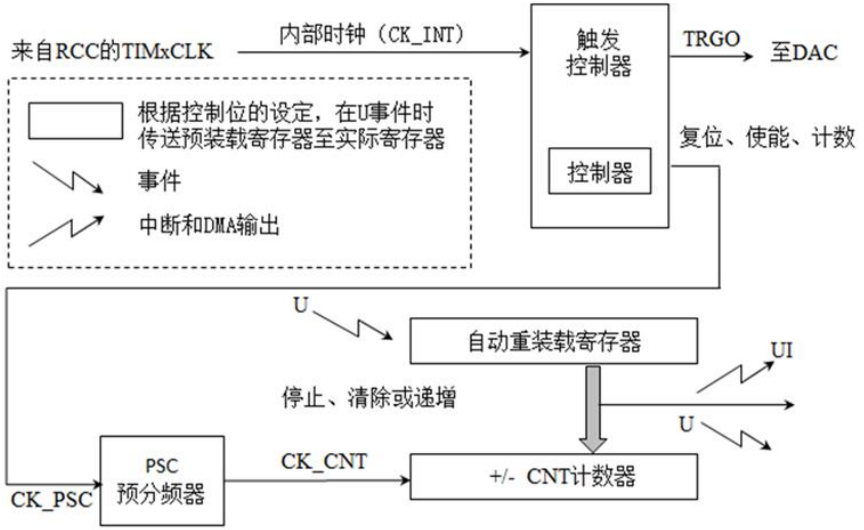
退出

基本定时器的功能

1. 时钟源

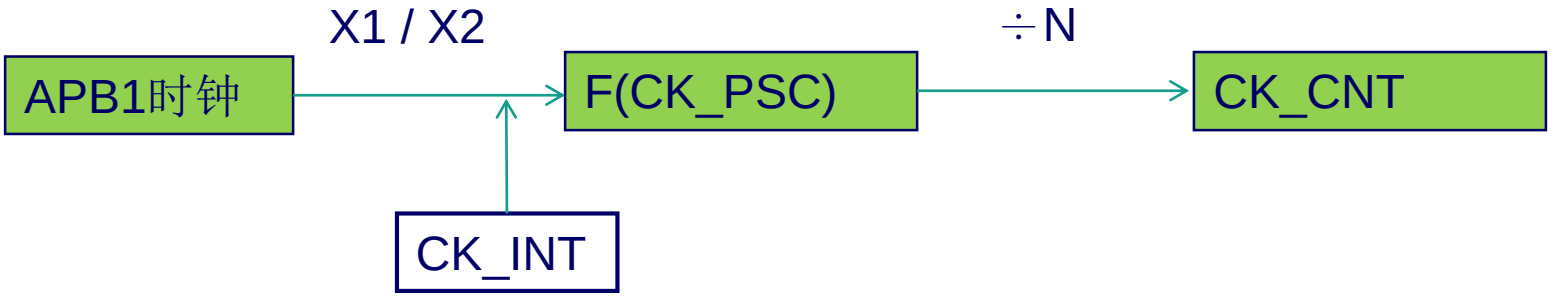
从STM32F103基本定时器内部结构图可以看出，基本定时器TIM6和TIM7只有一个时钟源，即内部时钟CK_INT。基本定时器TIM6和TIM7的TIMxCLK来源于APB1预分频器的输出。





基本定时器的功能

1. 时钟源



除非APB1的分频系数是1，否则通用定时器的时钟等于APB1时钟的2倍。

默认调用 SystemInit 函数情况下：

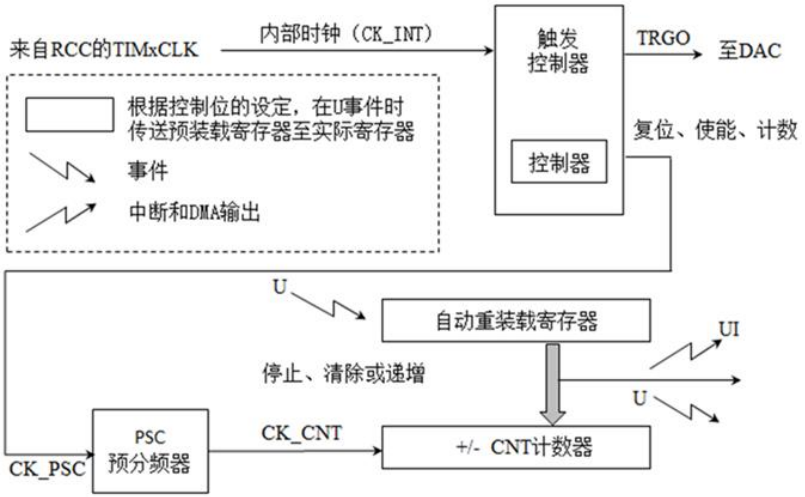
SYSCLK=72M

AHB时钟=72M

APB1时钟=36M

所以APB1的分频系数=AHB/APB1时钟=2

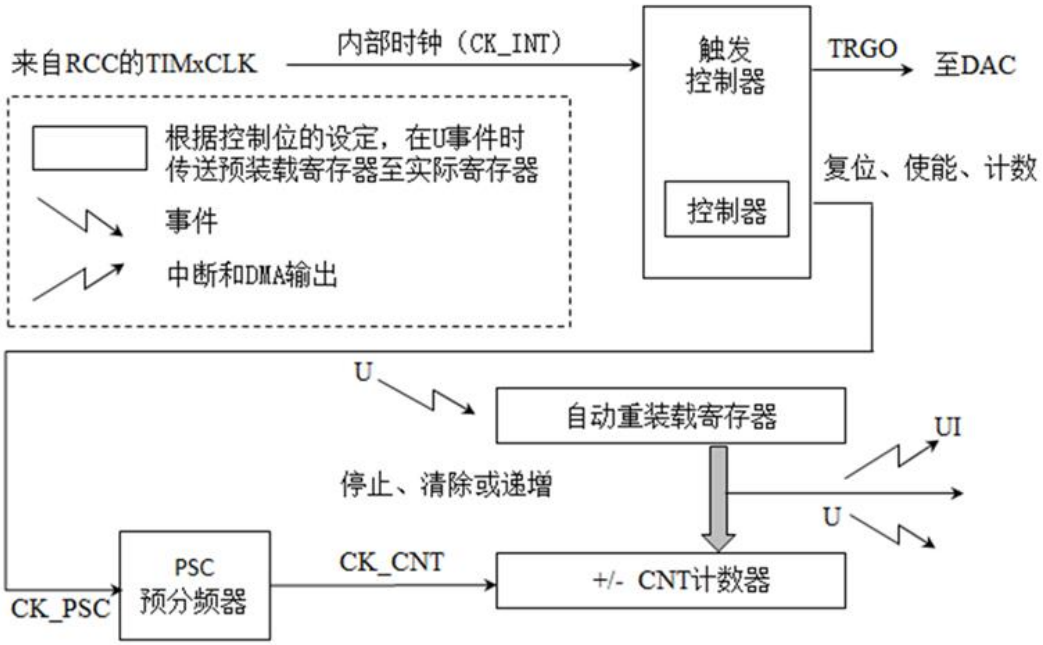
所以，通用定时器时钟CK_INT=2*36M=72M



基本定时器的功能

2. 时基单元

这个可编程定时器的主要部分是一个带有自动重装载的16位累加计数器，计数器的时钟通过一个预分频器得到。软件可以读写计数器(TIMx_CNT)、自动重装载寄存器(TIMx_ARR)和预分频寄存器(TIMx_PSC)，**即使计数器运行时也可以操作**。



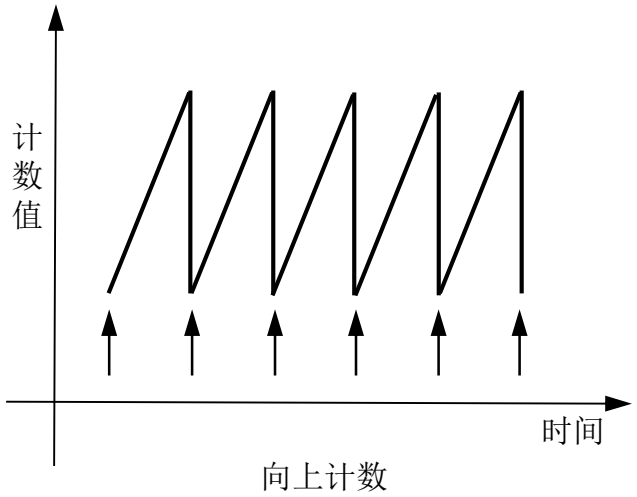
退出

3. 计数模式

STM32F103基本定时器只有向上计数工作模式，其工作过程如图所示，其中↑表示产生溢出事件。

基本定时器工作时，脉冲计数器 TIMx_CNT 从0累加计数到自动重装载数值（TIMx_ARR寄存器），然后重新从0开始计数并产生一个计数器溢出事件。由此可见，如果使用基本定时器进行延时，延时时间可以由以下公式计算：

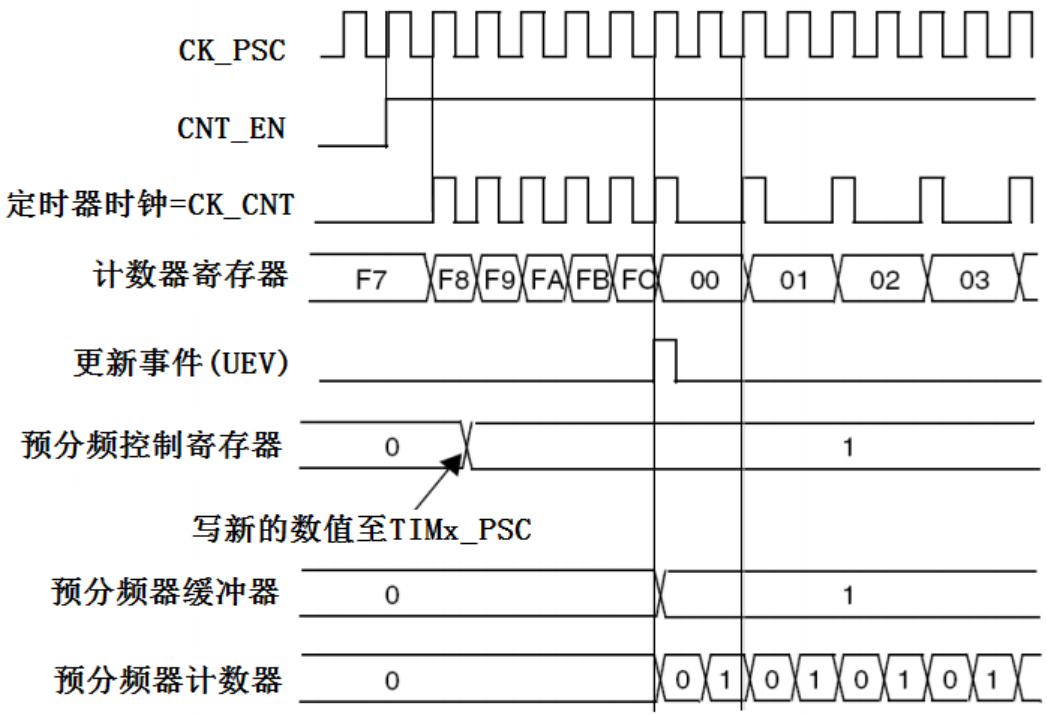
延时时间=（TIMx_ARR+1）*（TIMx_PSC+1）/ TIMxCLK

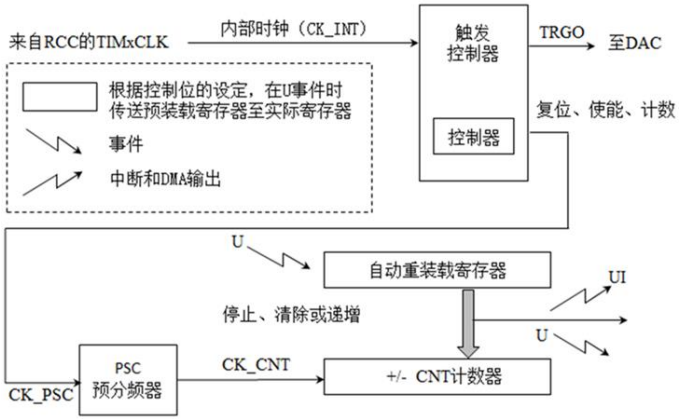


退出

4. PSC 预分频器

预分频可以以系数介于1至65536之间的任意数值对计数器时钟分频。它是通过一个16位寄存器(TIMx_PSC)的计数实现分频。



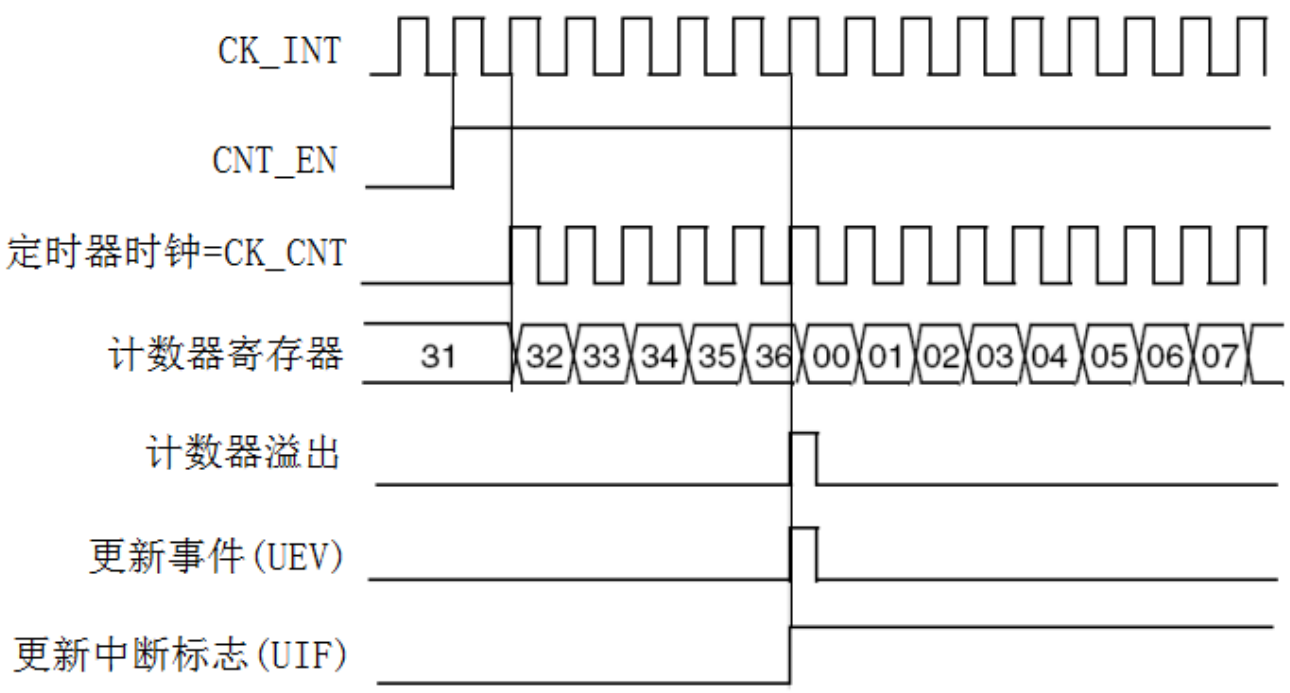


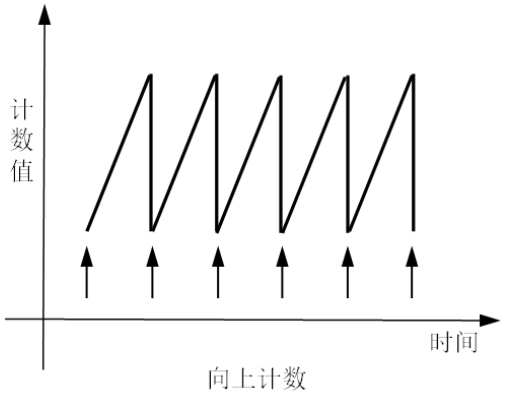
预分频系数从1变到2的计数器时序图

退出

5. 定时器定时周期设置

延时时间 =（TIMx_ARR+1）*（TIMx_PSC+1）/ TIMxCLK





计数器时序图（内部时钟分频系数为1）

退出

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
保留						CKD[1:0]		ARPE	CMS[1:0]		DIR	OPM	URS	UDIS	CEN
						rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

位7	**ARPE**：自动重装载预装载允许位 (Auto-reload preload enable) 0：TIMx_ARR寄存器没有缓冲； 1：TIMx_ARR寄存器被装入缓冲器。
位4	**DIR**：方向 (Direction) 0：计数器向上计数； 1：计数器向下计数。 注：当计数器配置为中央对齐模式或编码器模式时，该位为只读。
位0	**CEN**：使能计数器 (Counter enable) 0：禁止计数器； 1：使能计数器。 注：在软件设置了CEN位后，外部时钟、门控模式和编码器模式才能工作。触发模式可以自动地通过硬件设置CEN位。

基本定时器的寄存器

STM32F103基本定时器相关寄

操作这些外设寄存器，由于课程是

- **TIM6 和TIM7 控制寄存器 1(TIMx_CR1)**
- TIM6 和TIM7 控制寄存器 2(TIMx_CR2)
- TIM6 和TIM7 DMA/中断使能寄存器(TIMx_DIER)
- TIM6 和TIM7 状态寄存器(TIMx_SR)
- TIM6 和TIM7 事件产生寄存器(TIMx_EGR)
- **TIM6 和TIM7 计数器(TIMx_CNT)**
- **TIM6 和TIM7 预分频器(TIMx_PSC)**
- **TIM6 和TIM7 自动重装载寄存器(TIMx_ARR)**

退出

7.3 通用定时器

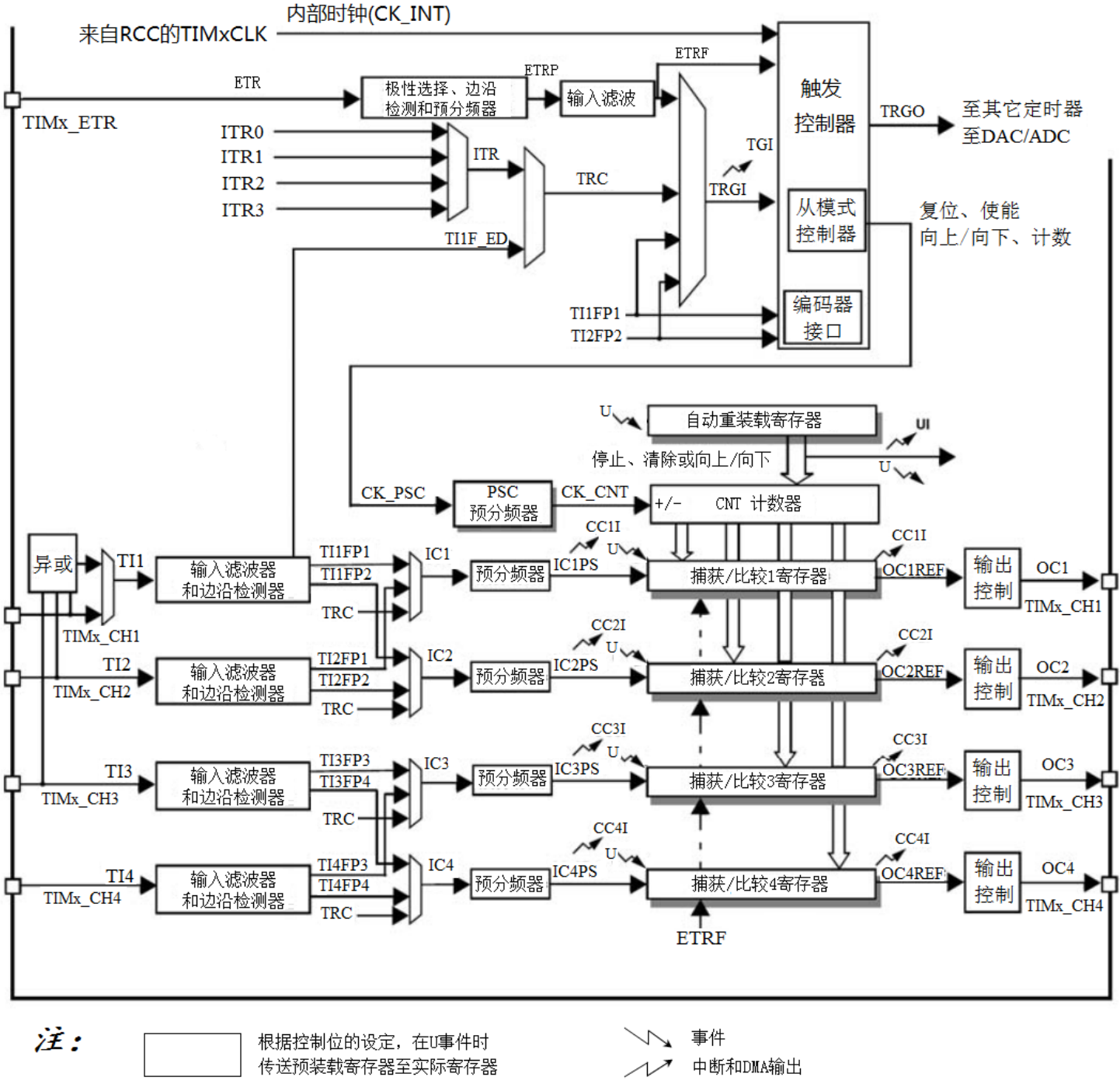
通用定时器（TIM2、TIM3、TIM4和TIM5）是一个通过可编程预分频器驱动的16位自动装载计数器构成。它适用于多种场合，包括**测量输入信号的脉冲长度（输入捕获）**或者**产生输出波形（输出比较和PWM）**。使用定时器预分频器和RCC时钟控制器预分频器，脉冲长度和波形周期可以在几个微秒到几个毫秒间调整。每个定时器都是完全独立的，没有互相共享任何资源。它们可以一起同步操作。

通用定时器主要功能

通用TIMx （TIM2、 TIM3、 TIM4和TIM5）定时器功能包括：

- 16位向上、向下、向上/向下自动装载计数器。
- 16位可编程(可以实时修改)预分频器。
- 4个独立通道：
  - ◆ 输入捕获◆ 输出比较◆ PWM生成◆ 单脉冲模式输出
- 使用外部信号控制定时器和定时器互连的同步电路。
- 如下事件发生时产生中断/DMA：
  - ◆ 更新： ◆ 触发事件◆ 输入捕获◆ 输出比较
- 支持针对定位的增量(正交)编码器和霍尔传感器电路。
- 触发输入作为外部时钟或者按周期的电流管理。

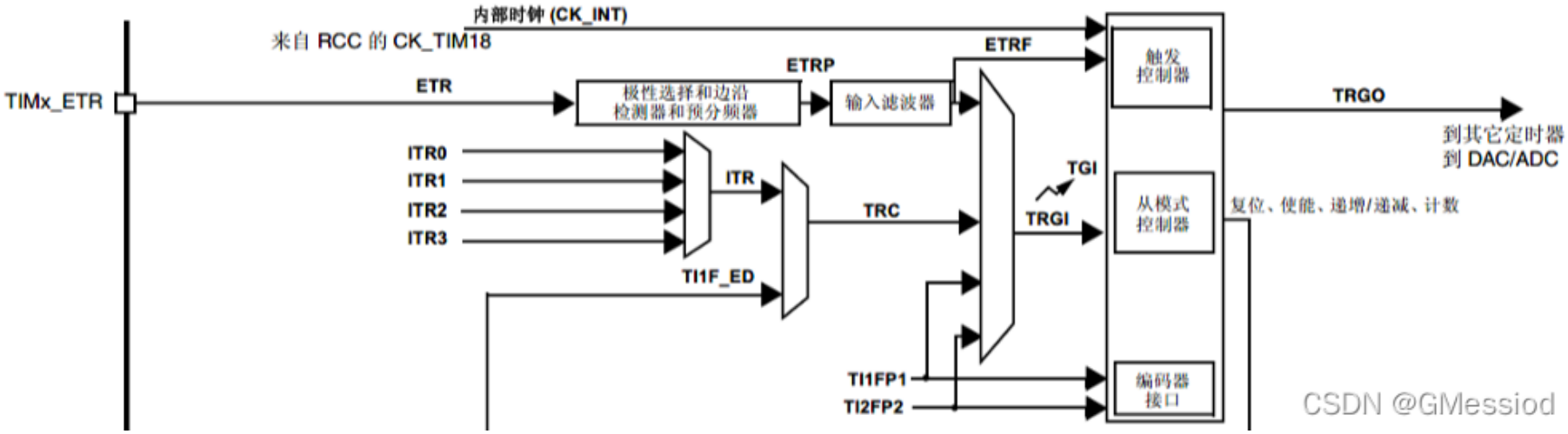
通用定时器内部结构如图所示，相比于基本定时器其内部结构要复杂多，其中最显著的地方就是增加了4个捕获/比较寄存器TIMx_CCR，这也是通用定时器之所以拥有那么多强大功能的原因。



1. 时钟选择

相比于基本定时器单一的内部时钟源，STM32F103通用定时器的16位计数器的时钟源有多种选择，可由以下时钟源提供。

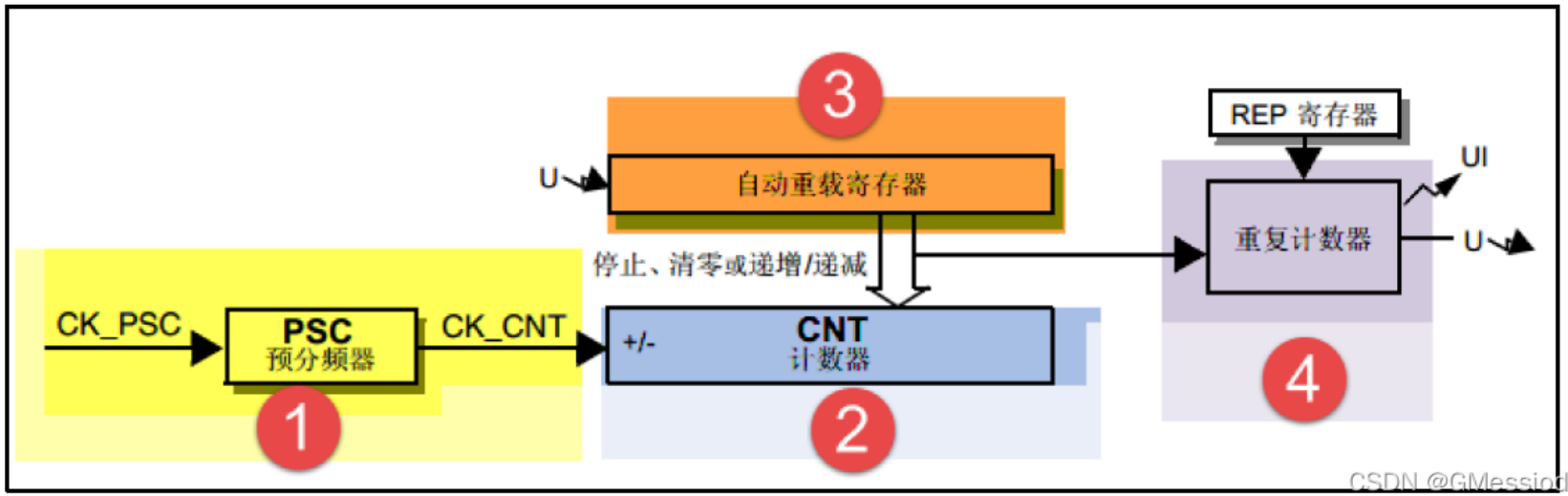
- 内部时钟(CK_INT)
- 外部时钟模式1：外部输入捕获引脚(TIx)
- 外部时钟模式2：外部触发输入(ETR)
- 内部触发输入(ITRx)：使用一个定时器作为另一个定时器的预分频器



CSDN @GMessiod

2. 时基单元

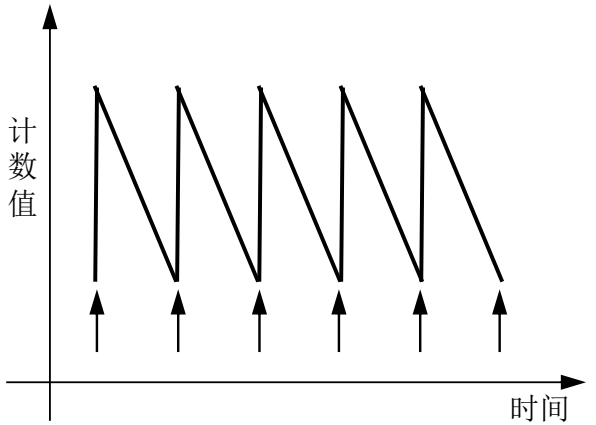
该部分主要实现了计数器的功能，时钟源信号通过预分频器产生计数脉冲信号（CK_CNT），计数器可以进行**向上计数、向下计数或者向上向下双向计数**，当计数值达到预设的自动重装载寄存器（ARR）的值时可以产生溢出事件或者中断信号。配合捕获比较寄存器（CCR）还可以实现输入捕获和输出比较功能。



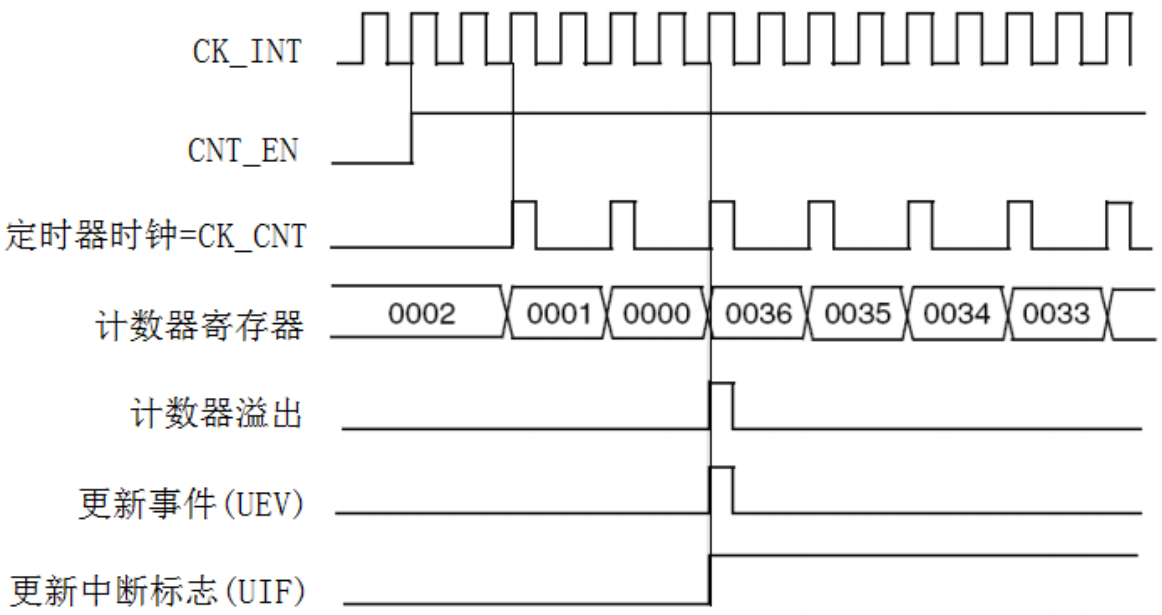
退出

3. 计数模式

工作在向下计数模式下，自动重装载寄存器值为0x36，内部预分频系数为2的计数器时序图。

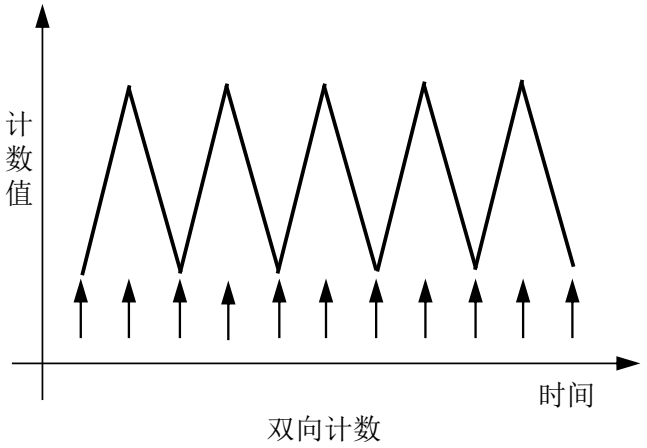


向下计数

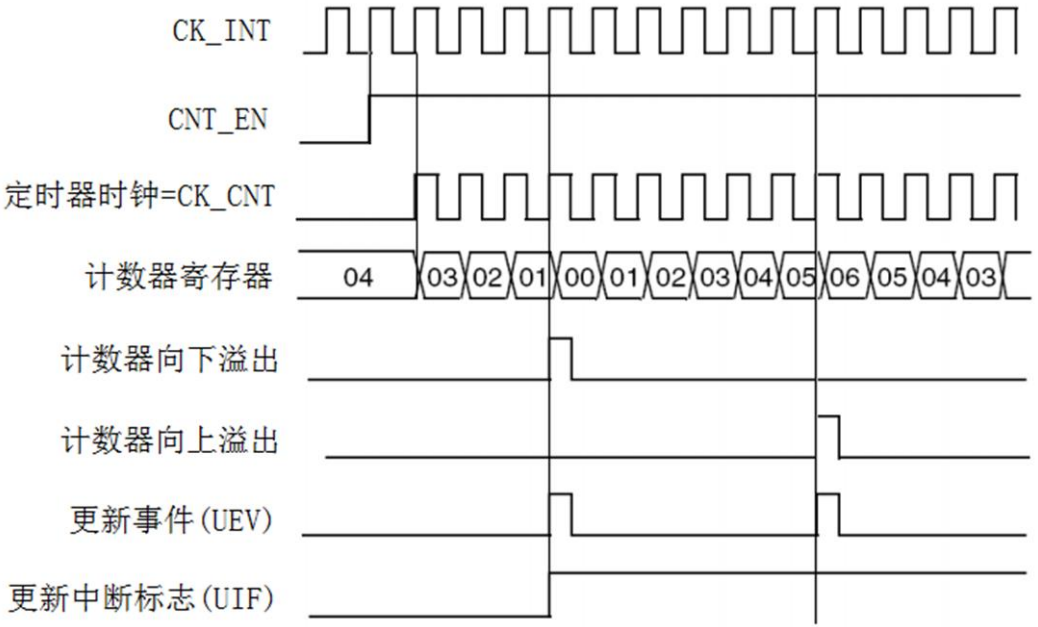


退出

向上/向下计数模式又称为中央对齐模式或双向计数模式，其工作过程如图所示，计数器从0开始计数到自动加载的值(TIMx_ARR寄存器)−1，产生一个计数器溢出事件，然后向下计数到1并且产生一个计数器下溢事件；然后再从0开始重新计数。

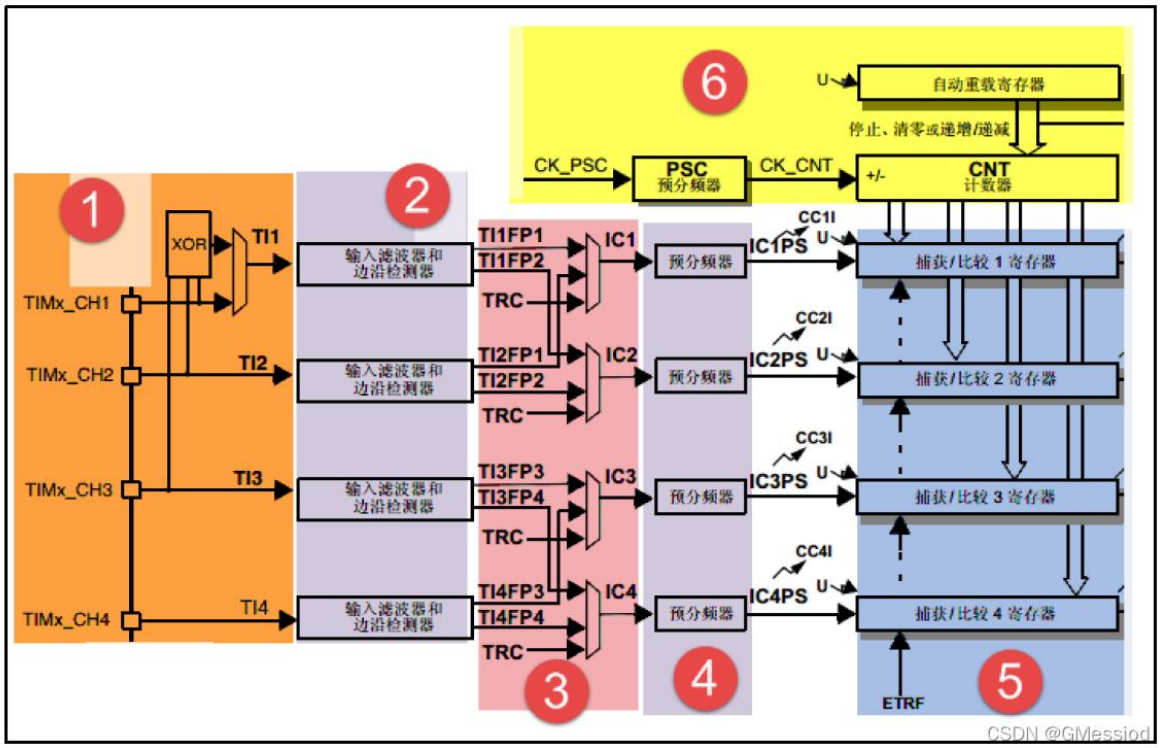


工作在向上/向下计数模式下的通用定时器，当自动重装载寄存器TIMx_ARR的值为0x06，内部预分频系数为1（预分频寄存器TIMx_PSC的值为0）的计数器时序图如图所示。



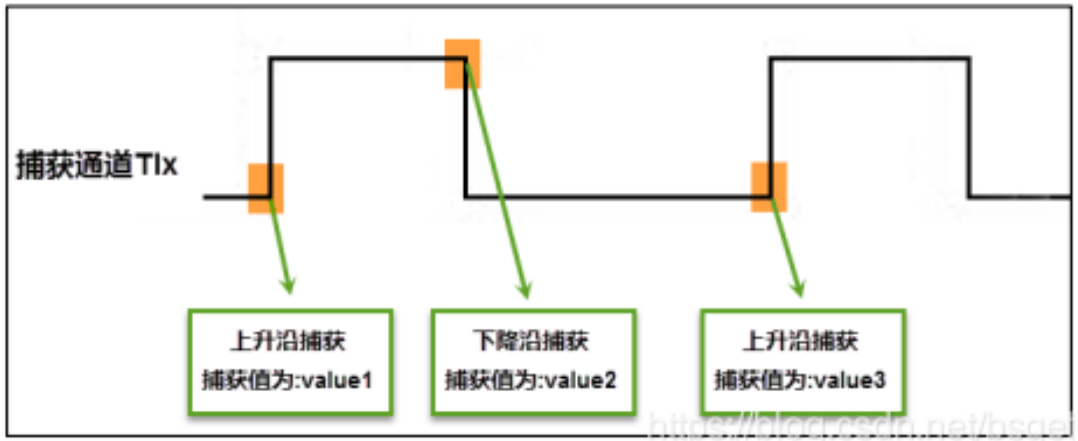
4. 输入捕获

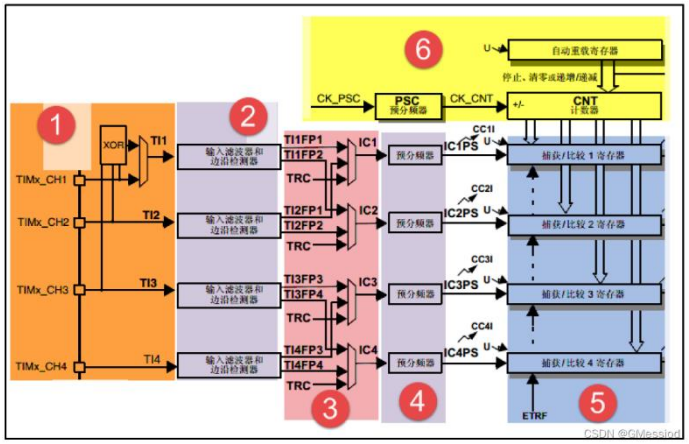
TIMx_CH1~TIMx_CH4 这四个通道，在芯片中都有对应的引脚，当脉冲从通道口进入时，经过输入滤波器（抗干扰的作用），然后经过边沿检测器检测到上升沿（下降沿），经过分频器，输入到公用部分中的捕获寄存器中，然后捕获寄存器记录此刻CNT计数器的值，当下一次下降沿（上升沿）过来时，也记录下CNT计数器的值，这样就可以计算出输入脉冲的宽度。



退出

4. 输入捕获

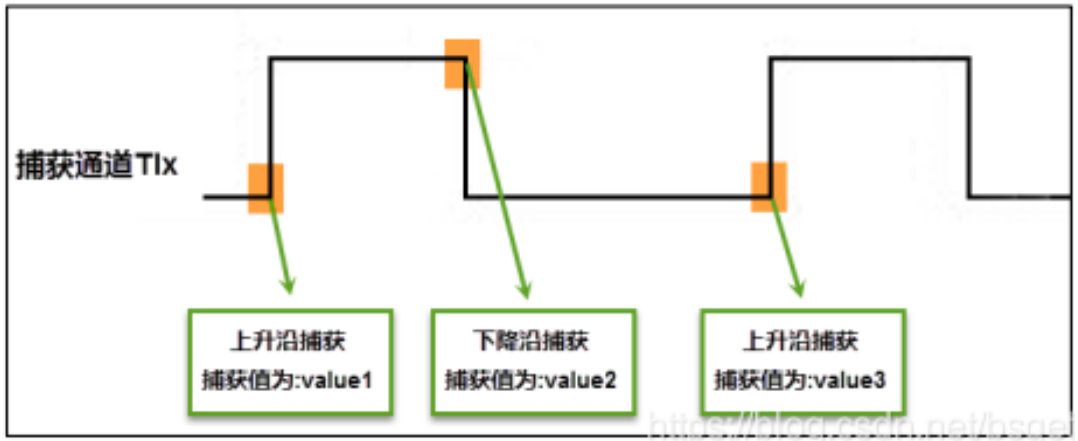


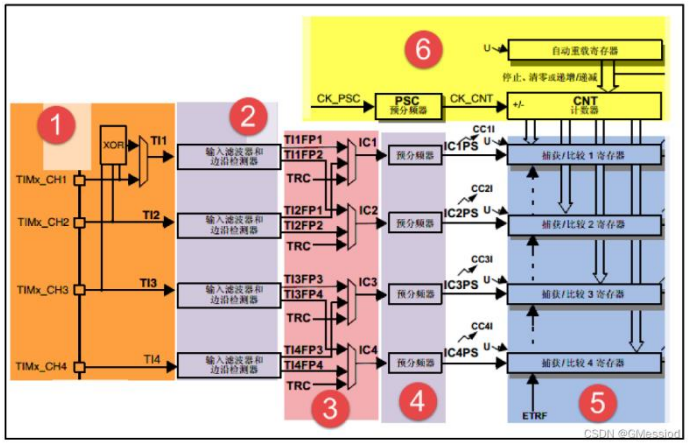


测量频率

当捕获通道 TIx 上出现上升沿时，发生第一次捕获，计数器 CNT 的值会被锁存到捕获寄存器 CCR 中，而且还会进入捕获中断，在中断服务程序中记录一次捕获（可以用一个标志变量来记录），并把捕获寄存器中的值读取到 value1 中。当出现第二次上升沿时，发生第二次捕获，计数器 CNT 的值会再次被锁存到捕获寄存器 CCR 中，并再次进入捕获中断，在捕获中断中，把捕获寄存器的值读取到 value3 中，并清除捕获记录标志。利用 value3 和value1 的差值我们就可以算出信号的周期（频率）。

4. 输入捕获



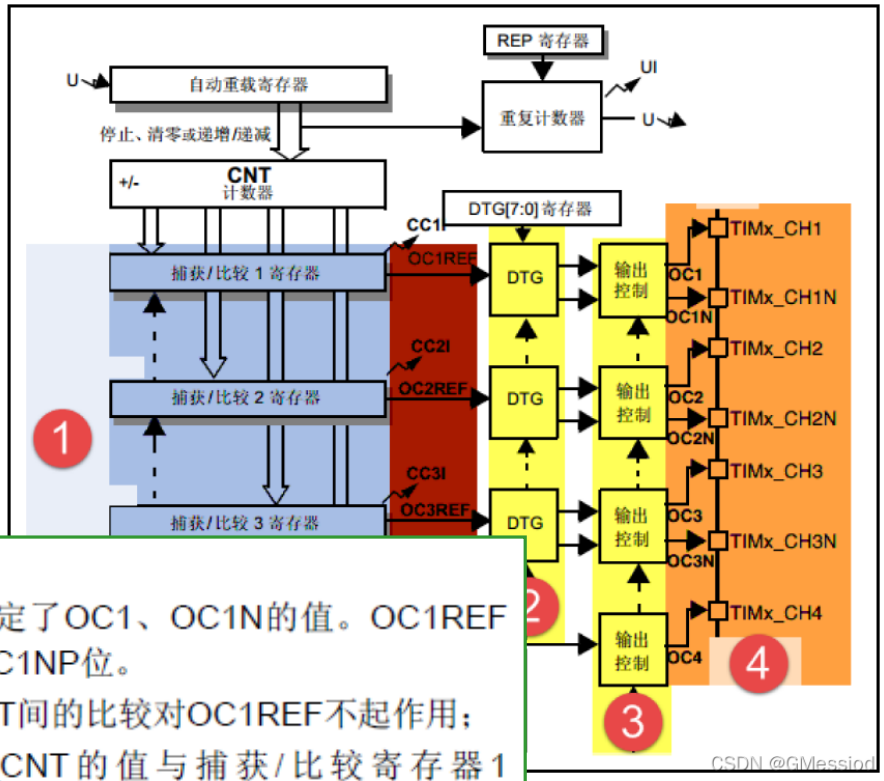


测量脉宽

当捕获通道 TIx 上出现上升沿时，发生第一次捕获，计数器 CNT 的值会被锁存到捕获寄存器 CCR 中，在中断服务程序中记录一次捕获，并把捕获寄存器中的值读取到 value1 中。然后把捕获边沿改变为下降沿捕获，目的是捕获后面的下降沿。当下降沿到来的时候，发生第二次捕获，计数器 CNT 的值会再次被锁存到捕获寄存器 CCR 中，并再次进入捕获中断，在捕获中断中，把捕获寄存器的值读取到 value2 中，并清除捕获记录标志。

5. 输出比较

输出比较模式总共有 8 种，具体的由寄存器 CCMRx 的位 OCxM[2:0] 配置。其中，最常用的是 PWM 模式。



OC1M[2:0]：输出比较1模式 (Output Compare 1 mode)

该3位定义了输出参考信号OC1REF的动作，而OC1REF决定了OC1、OC1N的值。OC1REF是高电平有效，而OC1、OC1N的有效电平取决于CC1P、CC1NP位。

000：冻结。输出比较寄存器TIMx_CCR1与计数器TIMx_CNT间的比较对OC1REF不起作用；

001：匹配时设置通道1为有效电平。当计数器TIMx_CNT的值与捕获/比较寄存器1(TIMx_CCR1)相同时，强制OC1REF为高。

010：匹配时设置通道1为无效电平。当计数器TIMx_CNT的值与捕获/比较寄存器1(TIMx_CCR1)相同时，强制OC1REF为低。

011：翻转。当TIMx_CCR1=TIMx_CNT时，翻转OC1REF的电平。

100：强制为无效电平。强制OC1REF为低。

101：强制为有效电平。强制OC1REF为高。

110：PWM模式1— 在向上计数时，一旦TIMx_CNT<TIMx_CCR1时通道1为有效电平，否则为无效电平；在向下计数时，一旦TIMx_CNT>TIMx_CCR1时通道1为无效电平(OC1REF=0)，否则为有效电平(OC1REF=1)。

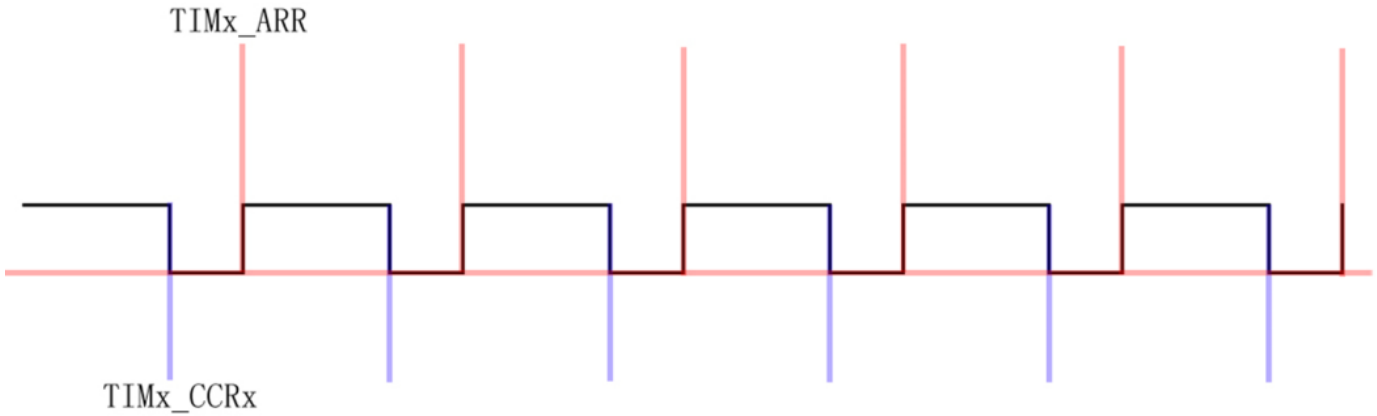
111：PWM模式2— 在向上计数时，一旦TIMx_CNT<TIMx_CCR1时通道1为无效电平，否则为有效电平；在向下计数时，一旦TIMx_CNT>TIMx_CCR1时通道1为有效电平，否则为无效电平。

PWM 模式

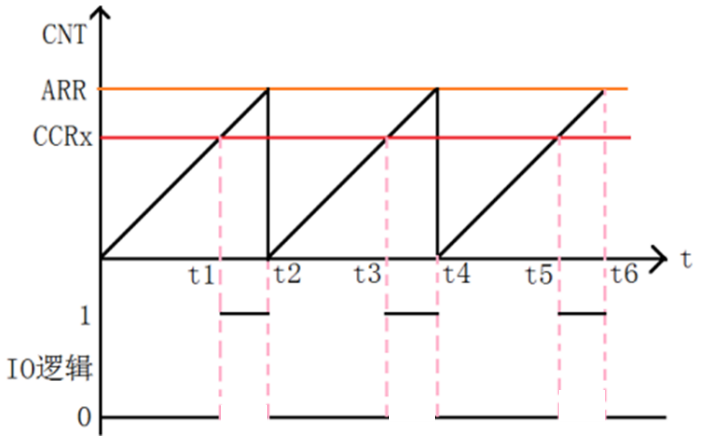
PWM 是 Pulse Width Modulation 的缩写，中文意思就是**脉冲宽度调制**，简称**脉宽调制**。它是利用微处理器的数字输出来对模拟电路进行控制的一种非常有效的技术，其控制简单、灵活和动态响应好等优点而成为电力电子技术最广泛应用的控制方式，其应用领域包括测量、通信、功率控制与变换，电动机控制、伺服控制、调光、开关电源，甚至某些音频放大器。

PWM输出模式的工作过程

STM32F103微控制器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，其产生原理如图所示。



STM32F103微控制器PWM产生原理



PWM输出模式的工作过程

通用定时器PWM输出模式的工作过程如下：

1）若配置脉冲计数器TIMx_CNT为向上计数模式，自动重装载寄存器TIMx_ARR的预设为N，则脉冲计数器TIMx_CNT的当前计数值X在时钟CK_CNT的驱动下从0开始不断累加计数。

2）在脉冲计数器TIMx_CNT随着时钟CK_CNT触发进行累加计数的同时，**脉冲计数器TIMx_CNT的当前计数值X与捕获/比较寄存器TIMx_CCR的预设值A进行比较；如果X<A，输出低电平（或高电平）；如果X≥A，输出高电平（或低电平）。**

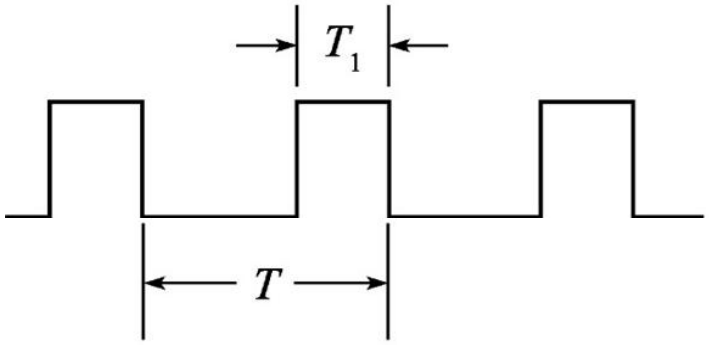
3）当脉冲计数器TIMx_CNT的计数值X大于自动重装载寄存器TIMx_ARR的预设值N时，脉冲计数器TIMx_CNT的计数值清零并重新开始计数。

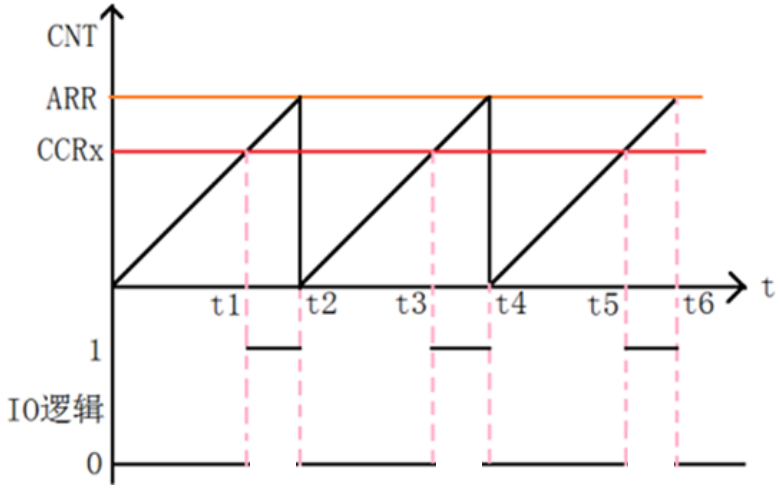
退出

PWM占空比的设置

PWM信号保持高电平的时间百分比称为占空比。如果信号始终为高电平，则它处于100%占空比，如果它始终处于低电平，则占空比为0%。如图所示，T1/T为占空比，T为一个PWM周期。

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。**





void TIM_SetCompare1(TIM_TypeDef TIMx, u16 Compare1)*

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
保留						CKD[1:0]		ARPE	CMS[1:0]		DIR	OPM	URS	UDIS	CEN
						rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

PWM占空比的设置

位7	**ARPE**：自动重装载预装载允许位 (Auto-reload preload enable) 0：TIMx_ARR寄存器没有缓冲； 1：TIMx_ARR寄存器被装入缓冲器。

图105　计数器时序图，当ARPE=0时的更新事件(TIMx_ARR没有预装入)

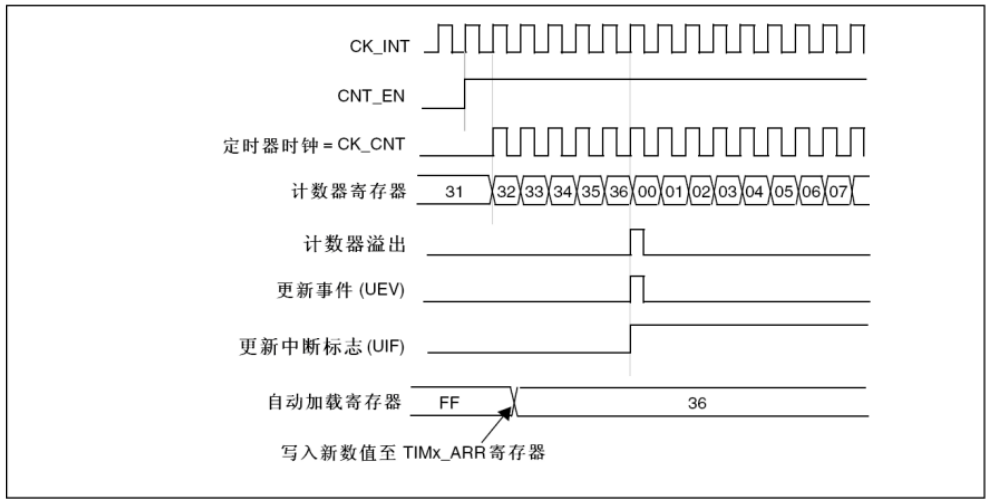
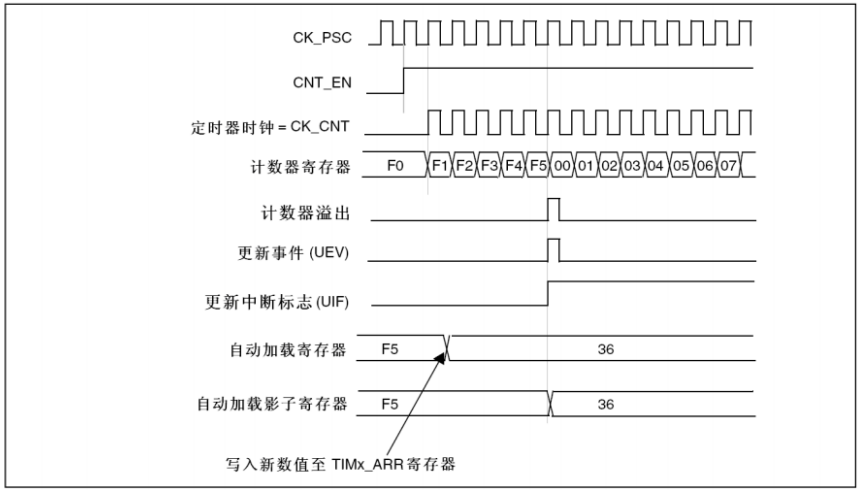


图106　计数器时序图，当ARPE=1时的更新事件(预装入了TIMx_ARR)



对于TIM_ARR寄存器的修改何时生效，这跟一个**TIMx_CR1寄存器的ARPE位**有关。如果ARPE=0，即不使用ARR的预装功能，则修改TIMx_ARR 寄存器的值，新的ARR的值立即生效。否则，如果ARPE=1，即使用ARR的预装功能，则修改TIMx_ARR 寄存器的值就是操作预装寄存器，直到发生更新事件后，新的ARR的值生效。

void TIM_OC2PreloadConfig(TIM_TypeDef TIMx, uint16_t TIM_OCPreload);*

void TIM_ARRPreloadConfig(TIM_TypeDef TIMx, FunctionalState NewState)*

7.4 高级定时器

高级控制定时器(TIM1和TIM8)由一个16位的自动装载计数器组成，它由一个可编程的预分频器驱动。它适合多种用途，包含测量输入信号的脉冲宽度(输入捕获)，或者产生输出波形(输出比较、PWM、嵌入死区时间的互补PWM等)。使用定时器预分频器和RCC时钟控制预分频器，可以实现脉冲宽度和波形周期从几个微秒到几个毫秒的调节。高级控制定时器(TIM1和TIM8)和通用定时器(TIMx)是完全独立的，它们不共享任何资源，它们可以同步操作。

高级定时器特性

TIM1和TIM8定时器的功能包括：

- 16位向上、向下、向上/下自动装载计数器
- 16位可编程(可以实时修改)预分频器
- 多达4个独立通道：
- 死区时间可编程的互补输出
- 使用外部信号控制定时器和定时器互联的同步电路
- 允许在指定数目的计数器周期之后更新定时器寄存器的重复计数器
- 刹车输入信号可以将定时器输出信号置于复位状态或者一个已知状态
- 如下事件发生时产生中断/DMA：

  ◆ 更新◆ 触发事件◆ 输入捕获◆ 输出比较◆ 刹车信号输入
- 支持针对定位的增量(正交)编码器和霍尔传感器电路
- 触发输入作为外部时钟或者按周期的电流管理

退出

高级定时器结构

STM32F103高级定时器的内部结构要比通用定时器复杂一些，但其核心仍然与基本定时器、通用定时器相同，是一个由可编程的预分频器驱动的具有自动重装载功能的16位计数器。与通用定时器相比，STM32F103高级定时器主要多了BRK和DTG两个结构，因而具有了死区时间的控制功能。

因为高级定时器的特殊功能，在普通应用中一般较少使用，所以不作为课程的重点，如需详细了解可以查阅STM32中文参考手册。

7.5 定时器相关库函数

函数 TIM_DeInit

函数名	TIM_DeInit
函数原形	void TIM_DeInit(TIM_TypeDef* TIMx)
功能描述	将外设 TIMx 寄存器重设为缺省值
输入参数	TIMx： x 可以是 1~8 ， 来选择 TIM 外设
输出参数	无
返回值	无
先决条件	无
被调用函数	RCC_APB1PeriphClockCmd().

函数TIM_TimeBaseInit

函数名	TIM_TimeBaseInit
函数原形	void TIM_TimeBaseInit(TIM_TypeDef* TIMx, TIM_TimeBaseInitTypeDef* TIM_TimeBaseInitStruct)
功能描述	根据 TIM_TimeBaseInitStruct 中指定的参数初始化 TIMx 的时间基数单位
输入参数 1	TIMx： x 可以是 1~8 ， 来选择 TIM 外设
输入参数 2	TIMTimeBase_InitStruct：指向结构 TIM_TimeBaseInitTypeDef 的指针，包含了 TIMx 时间基数单位的配置信息
输出参数	无
返回值	无
先决条件	无
被调用函数	无

```
TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;
TIM_TimeBaseStructure.TIM_Period = 0xFFFF;
TIM_TimeBaseStructure.TIM_Prescaler = 0xF;
TIM_TimeBaseStructure.TIM_ClockDivision = 0x0;
TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
TIM_TimeBaseInit(TIM2, & TIM_TimeBaseStructure);
```

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。

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。

```
typedef struct
{
  uint16_t TIM_Prescaler;
  uint16_t TIM_CounterMode;
  uint16_t TIM_Period;
  uint16_t TIM_ClockDivision;
  uint8_t TIM_RepetitionCounter; //仅高级定时器TIM1和TIM8有效
} TIM_TimeBaseInitTypeDef;
```

TIM_ClockDivision:

设置了时钟分割。该参数取值见下表

TIM_ClockDivision	描述
TIM_CKD_DIV1	TDTS = Tck_tim
TIM_CKD_DIV2	TDTS = 2Tck_tim
TIM_CKD_DIV4	TDTS = 4Tck_tim

TIM_CounterMode:

选择了计数器模式。该参数取值见下表。

TIM_CounterMode	描述
TIM_CounterMode_Up	TIM 向上计数模式
TIM_CounterMode_Down	TIM 向下计数模式
TIM_CounterMode_CenterAligned1	TIM 中央对齐模式 1 计数模式
TIM_CounterMode_CenterAligned2	TIM 中央对齐模式 2 计数模式
TIM_CounterMode_CenterAligned3	TIM 中央对齐模式 3 计数模式

函数TIM_Cmd

函数名	TIM_Cmd
函数原形	void TIM_Cmd(TIM_TypeDef* TIMx, FunctionalState NewState)
功能描述	使能或者失能 TIMx 外设
输入参数 1	TIMx：x可以是 1~8 ，用于选择TIM外设
输入参数 2	NewState: 外设 TIMx 的新状态 这个参数可以取： ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

函数 TIM_OC1Init

函数名	TIM_OC1Init
函数原形	void TIM_OC1Init(TIM_TypeDef* TIMx, TIM_OCInitTypeDef* TIM_OCInitStruct)
功能描述	根据 TIM_OCInitStruct 中指定的参数初始化 TIMx 通道 1
输入参数1	TIMx：x可以是1、2、3、4、5或8，用于选择TIM外设
输入参数2	TIM_OCInitStruct：指向结构 TIM_OCInitTypeDef 的指针，包含了 TIMx 时间基数单位的配置信息。
输出参数	无
返回值	无
先决条件	无
被调用函数	无

函数 TIM_OC1Init

TIM_OCInitTypeDef 定义于文件

“STM32F10x_StdPeriph_Lib_V3.5.0\Libraries\STM32F10x_StdPeriph_Driver\inc \ stm32f10x_tim.h”

老版定义：

```
typedef struct
{
    u16 TIM_OCMode;
    u16 TIM_Channel;
    u16 TIM_Pulse;
    u16 TIM_OCPolarity;
} TIM_OCInitTypeDef;
```

新版定义：

```
typedef struct
{
    uint16_t TIM_OCMode;  //PWM模式1或者模式2
    uint16_t TIM_OutputState; //输出使能 OR失能
    uint16_t TIM_OutputNState;
    uint16_t TIM_Pulse;  //比较值，写CCRx
    uint16_t TIM_OCPolarity; //比较输出极性
    uint16_t TIM_OCNPolarity;
    uint16_t TIM_OCIdleState;
    uint16_t TIM_OCNIdleState;
} TIM_OCInitTypeDef;
```

TIM_OCMode:

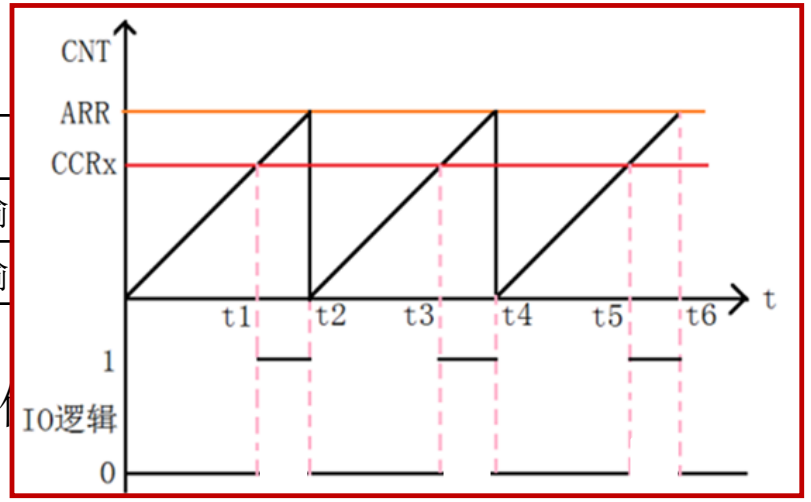
110：PWM模式1— 在向上计数时，一旦TIMx_CNT<TIMx_CCR1时通道1为有效电平，否则为无效电平；在向下计数时，一旦TIMx_CNT>TIMx_CCR1时通道1为无效电平(OC1REF=0)，否则为有效电平(OC1REF=1)。

111：PWM模式2— 在向上计数时，一旦TIMx_CNT<TIMx_CCR1时通道1为无效电平，否则为有效电平；在向下计数时，一旦TIMx_CNT>TIMx_CCR1时通道1为有效电平，否则为无效电平。

TIM_OCM	
TIM_OCM	
TIM_OCMode_Inactive	TIM 输出比较非主动模式
TIM_OCMode_Toggle	TIM 输出比较触发模式
TIM_OCMode_PWM1	TIM 脉冲宽度调制模式 1
TIM_OCMode_PWM2	TIM 脉冲宽度调制模式 2

TIM_OutputState: 选择输出比较状态

TIM_OutputState	
TIM_OutputState_Disable	失能输
TIM_OutputState_Enable	使能输



TIM_Pulse: 设置待装入捕获比较寄存器的脉冲

TIM_OCPolarity: 输出极性。

TIM_OCPolarity	描述
TIM_OCPolarity_High	TIM 输出比较极性高
TIM_OCPolarity_Low	TIM 输出比较极性低

函数TIM_OC2Init

函数名	TIM_OC2Init
函数原形	void TIM_OC2Init(TIM_TypeDef* TIMx, TIM_OCInitTypeDef* TIM_OCInitStruct)
功能描述	根据 TIM_OCInitStruct 中指定的参数初始化 TIMx 通道 1
输入参数1	TIMx：x可以是1、2、3、4、5或8，用于选择TIM外设
输入参数2	TIM_OCInitStruct：指向结构 TIM_OCInitTypeDef 的指针，包含了 TIMx 时间基数单位的配置信息。
输出参数	无
返回值	无
先决条件	无
被调用函数	无

函数TIM_SetCompare1

函数名	TIM_SetCompare1
函数原形	void TIM_SetCompare1(TIM_TypeDef* TIMx, u16 Compare1)
功能描述	设置 TIMx 捕获比较 1 寄存器值
输入参数 1	TIMx：x可以是1、2、3、4、5或8，用于选择TIM外设
输入参数 2	Compare1：捕获比较 1 寄存器新值
输出参数	无
返回值	无
先决条件	无
被调用函数	无

```
/* Sets the TIM2 new Output Compare 1 value */
u16 TIMCompare1 = 0x7FFF;
TIM_SetCompare1(TIM2, TIMCompare1);
```

函数TIM_OC1PreloadConfig

函数名	TIM_OC1PreloadConfig
函数原形	void TIM_OC1PreloadConfig(TIM_TypeDef* TIMx, u16 TIM_OCPreload)
功能描述	使能或者失能 TIMx 在 CCR1 上的预装载寄存器
输入参数 1	TIMx：x可以是1、2、3、4、5或8，用于选择TIM外设
输入参数 2	TIM_OCPreload：输出比较预装载状态 参阅 Section： TIM_OCPreload 查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

TIM_OCPreload: 输出比较预装载状态可以使能或者失能如下表

TIM_OCPreload	描述
TIM_OCPreload_Enable	TIMx 在 CCR1 上的预装载寄存器使能
TIM_OCPreload_Disable	TIMx 在 CCR1 上的预装载寄存器失能

函数 TIM _ITConfig

函数名	TIM_ITConfig
函数原形	void TIM_ITConfig(TIM_TypeDef* TIMx, u16 TIM_IT, FunctionalState NewState)
功能描述	使能或者失能指定的 TIM 中断
输入参数 1	TIMx： x 可以是 1~8 ， 来选择 TIM 外设
输入参数 2	TIM_IT：待使能或者失能的 TIM 中断源 参阅 Section： TIM_IT 查阅更多该参数允许取值范围
输入参数 3	NewState： TIMx 中断的新状态 这个参数可以取： ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

```
/* Enables the TIM2 Capture Compare channel 1 Interrupt source */
TIM_ITConfig(TIM2, TIM_IT_CC1, ENABLE );
```

TIM_IT: 输入参数 TIM_IT 使能或者失能 TIM 的中断。可以取下表的一个或者多个取值的组合作为该参数的值。

TIM_IT	描述
TIM_IT_Update	TIM 中断源
TIM_IT_CC1	TIM 捕获/比较 1 中断源
TIM_IT_CC2	TIM 捕获/比较 2 中断源
TIM_IT_CC3	TIM 捕获/比较 3 中断源
TIM_IT_CC4	TIM 捕获/比较 4 中断源
TIM_IT_Trigger	TIM 触发中断源

```
/* Enables the TIM2 Capture Compare channel 1 Interrupt source */
TIM_ITConfig(TIM2, TIM_IT_CC1, ENABLE );
```

函数 TIM_ClearFlag

函数名	TIM_ ClearFlag
函数原形	void TIM_ClearFlag(TIM_TypeDef* TIMx, uint16_t TIM_FLAG)
功能描述	清除 TIMx 的待处理标志位
输入参数 1	TIMx： x 可以是 1~8 ， 来选择 TIM 外设
输入参数 2	TIM_FLAG：待清除的 TIM 标志位
输出参数	无
返回值	无
先决条件	无
被调用函数	无

7.6 定时器秒计时项目实施

项目分析

核心功能是实现精确的 1秒定时

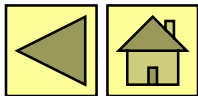
定时器初始化主要步骤包括

- 打开定时器所挂接的时钟
- 利用TIM_TimeBaseInit函数对定时器进行初始化。
- 启动定时器
- 清除中断标志位
- 配置定时器中断
- 设置中断优先级

延时时间 =（TIMx_ARR+1）*（TIMx_PSC+1）/ TIMxCLK

```
void TIM6Init()
{
  TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;
  NVIC_InitTypeDef NVIC_InitStructure;
  //打开TIM6的APB1时钟
  RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM6 , ENABLE);
  //设置自动重装载寄存器周期的值 寄存器的值为周期值-1
  TIM_TimeBaseStructure.TIM_Period = 36000-1;
  //设置预分频系数 预分频寄存器的值为分频系数-1
  TIM_TimeBaseStructure.TIM_Prescaler = 2000-1;
  //设置时钟分割:TDTS = Tck_tim
  TIM_TimeBaseStructure.TIM_ClockDivision = 0;
  //TIM向上计数模式
  TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
  //初始化TIM6定时器
  TIM_TimeBaseInit(TIM6, &TIM_TimeBaseStructure);
  //清除TIMx的中断待处理位:TIMx 中断源
  TIM_ClearFlag(TIM6,TIM_FLAG_Update);
  /* 设置中断参数，并打开中断 */
  TIM_ITConfig(TIM6,TIM_IT_Update,ENABLE);
  //使能或者失能TIMx外设
  TIM_Cmd(TIM6,ENABLE);
  /* 设置NVIC参数 设优先级 开中断*/
  NVIC_InitStructure.NVIC_IRQChannel = TIM6_IRQn; //指定中断通道
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;//配置抢占式优先级
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;//配置响应式优先级
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE; //中断使能
  NVIC_Init(&NVIC_InitStructure);
}
```

TIM_IT	描述
TIM_IT_Update	TIM 中断源
TIM_IT_CC1	TIM 捕获/比较 1 中断源
TIM_IT_CC2	TIM 捕获/比较 2 中断源
TIM_IT_CC3	TIM 捕获/比较 3 中断源
TIM_IT_CC4	TIM 捕获/比较 4 中断源
TIM_IT_Trigger	TIM 触发中断源



```
/* main.c */
int  main (void)
{
    TIM6Init();
    while (1) {
    }
}

/* stm32f10x_it.c */
void TIM6_IRQHandler(void)
{
            TIM_ClearFlag(TIM6,TIM_IT_Update);
            ;
}
```

7.7 PWM项目实施

项目分析

利用TIM3的通道1和通道2，把两个通道完全重映射到PC6和PC7引脚，产生两路PWM方波来控制开发板L7和L8指示灯的亮度。通过修改PWM波的占空比，控制L7和L8指示灯由亮到暗，再由暗到亮，实现PWM呼吸灯的效果。

表42　TIM3复用功能重映像

复用功能	TIM3_REMAP[1:0] = 00 (没有重映像)	TIM3_REMAP[1:0] = 10 (部分重映像)	TIM3_REMAP[1:0] = 11 (完全重映像)(1)
TIM3_CH1	PA6	PB4	PC6
TIM3_CH2	PA7	PB5	PC7
TIM3_CH3	PB0		PC8
TIM3_CH4	PB1		PC9

① 使能定时器3和相关IO口时钟。

使能定时器3时钟：RCC_APB1PeriphClockCmd();

使能GPIOC时钟：RCC_APB2PeriphClockCmd();

② 初始化IO口为复用功能输出。函数：GPIO_Init();

GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;

③ 这里我们是要把PC6和PC7用作定时器的PWM输出引脚，所以要重映射配置，所以需要开启AFIO时钟，同时设置重映射。

RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO,ENABLE);

GPIO_PinRemapConfig(GPIO_FullRemap_TIM3, ENABLE);

④ 初始化定时器：ARR,PSC等：*TIM_TimeBaseInit();*

⑤ 初始化输出比较参数: *TIM_OC1Init()， TIM_OC2Init();*

⑥ 使能预装载寄存器： *TIM_OC2PreloadConfig(TIM3, TIM_OCPreload_Enable);*

⑦ 使能定时器：*TIM_Cmd();*

⑧ 不断改变比较值CCRx，达到不同的占空比效果: *TIM_SetCompare2();*

```
void TIM3_PWMInit()
{
    GPIO_InitTypeDef GPIO_InitStructure;  //声明结构体变量，用来初始化GPIO
    TIM_TimeBaseInitTypeDef TIM_TimeBaseInitStructure; //声明结构体变量，用来初始化定时器
    TIM_OCInitTypeDef TIM_OCInitStructure; //根据TIM_OCInitStruct中指定的参数初始化外设
TIMx

     /* 开启时钟 */
     RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC,ENABLE);
     RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO,ENABLE);
     RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3,ENABLE);

      /*  配置GPIO的模式和IO口 */
     GPIO_InitStructure.GPIO_Pin=GPIO_Pin_6|GPIO_Pin_7;
     GPIO_InitStructure.GPIO_Speed=GPIO_Speed_50MHz;
     GPIO_InitStructure.GPIO_Mode=GPIO_Mode_AF_PP;  //复用推挽输出
     GPIO_Init(GPIOC,&GPIO_InitStructure);
```

```
/* TIM3定时器初始化 */
TIM_TimeBaseInitStructure.TIM_Period = 900-1;
//设置自动重装载寄存器周期的值，不分频，PWM 频率=72000/900=8Khz
TIM_TimeBaseInitStructure.TIM_Prescaler = 1-1;
//设置用来作为TIMx时钟频率预分频值，此处分频系数为1，即不分频
TIM_TimeBaseInitStructure.TIM_ClockDivision = 0; //设置时钟分割:TDTS = Tck_tim
TIM_TimeBaseInitStructure.TIM_CounterMode = TIM_CounterMode_Up;  //TIM向上计数模式
TIM_TimeBaseInit(TIM3, & TIM_TimeBaseInitStructure);

GPIO_PinRemapConfig(GPIO_FullRemap_TIM3, ENABLE);
//改变指定管脚的映射PC6、PC7
```

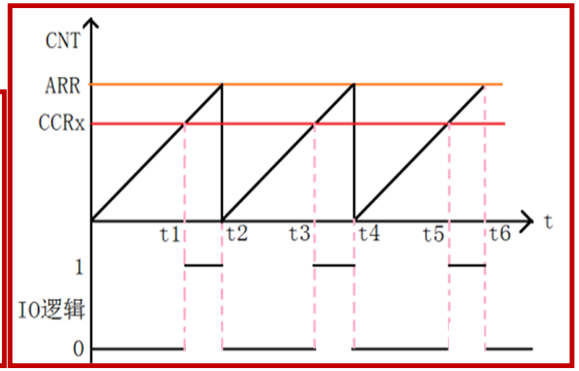
```
/* PWM初始化 */
TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;
TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable; //PWM输出使能
TIM_OCInitStructure. TIM_Pulse = 0;    //捕获比较寄存器初始值为0
TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_Low;
TIM_OC1Init(TIM3,&TIM_OCInitStructure);
TIM_OC2Init(TIM3,&TIM_OCInitStructure);
//注意此处初始化时TIM_OC2Init而不是TIM_OCInit，固件库的版本不一样，否则会出错。

TIM_OC1PreloadConfig(TIM3, TIM_OCPreload_Enable); //使能TIMx在CCR1上的预装载寄存器
TIM_OC2PreloadConfig(TIM3, TIM_OCPreload_Enable); //使能TIMx在CCR2上的预装载寄存器

TIM_Cmd(TIM3,ENABLE);//使能或者失能TIMx外设
}
```

110：PWM模式1— 在向上计数时，一旦TIMx_CNT<TIMx_CCR1时通道1为有效电平，否则为无效电平；在向下计数时，一旦TIMx_CNT>TIMx_CCR1时通道1为无效电平(OC1REF=0)，否则为有效电平(OC1REF=1)。

111：PWM模式2— 在向上计数时，一旦TIMx_CNT<TIMx_CCR1时通道1为无效电平，否则为有效电平；在向下计数时，一旦TIMx_CNT>TIMx_CCR1时通道1为有效电平，否则为无效电平。



```
/* main.c */
int  main (void)
{
    u8 dir=1; //方向
    u32 Duty=0;
    TIM3_PWMInit();  //PWM初始化
    while(1) {
        delay_ms(10);
        if(dir==1) {
            Duty=Duty+10;
            if(Duty>600)
                dir=0;
        }
        else{
            Duty=Duty-10;
            if(Duty==0)
                dir=1;
        }
        TIM_SetCompare1(TIM3, Duty);
        //设置TIMx捕获比较1寄存器值
        TIM_SetCompare2(TIM3, Duty);
        //设置TIMx捕获比较2寄存器值
    }
}
```

本章小结

STM32F103微控制器内部集成三类可编程定时器：基本定时器（TIM6和TIM7）、通用定时器（TIM2~TIM5）和高级定时器（TIM1、TIM8），其主要功能如下：

1. 通过内部时钟计数，实现定时

2. 计算外部脉冲的个数

3. 测量外部脉冲的宽度和频率

4. 输出一个宽度可控的脉冲

5. 输出PWM波形

6. 正交编码器输入

7. 霍尔传感器输入