南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Embedded System and Microcomputer Principle

LAB5 Timer Interrupt

2021 Fall
wangq9@mail.sustech.edu.cn

南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

CONTENTS

01

General-purpose Timers Description

1. General-purpose Timers Description -- Timers of STM32F103RCT6

- There are three kinds of timer in STM32: advanced timers, general purpose timers and basic timers
  - General-purpose timers are the most common timers in STM32, which supports PWM generation, input capture, time-base generation(update interrupt) and output compare
  - Basic timers don’t have IOs channels for input capture/PWM generation so that they are only used in time-base generation purposes
  - Advanced timers have more features than general purpose timers like complementary PWM generation

1. General-purpose Timers Description -- Timers of STM32F103RCT6(continued)

定时器种类	位数	计数器模式	产生DMA请求	捕获/比较通道	互补输出	特殊应用场景
高级定时器(TIM1,TIM8)	16	向上，向下，向上/下	可以	4	有	带死区控制盒紧急刹车，可应用于PWM电机控制
通用定时器(TIM2~TIM5)	16	向上，向下，向上/下	可以	4	无	通用。定时计数，PWM输出，输入捕获，输出比较
基本定时器(TIM6,TIM7)	16	向上，向下，向上/下	可以	0	无	主要应用于驱动DAC

1. General-purpose Timers Description

-- TIM2 to TIM5 main features

- 位于低速的APB1总线上(APB1)
- 16 位向上、向下、向上/向下(中心对齐)计数模式
- 自动装载计数器（TIMx_ARR）
- 16 位可编程(可以实时修改)预分频器(TIMx_PSC)，计数器时钟频率的分频系数为1～65535之间的任意数值。
- 4个独立通道（TIMx_CH1~4），这些通道可以用来作为：输入捕获，输出比较，PWM 生成，单脉冲模式输出

1. General-purpose Timers Description

-- TIM2 to TIM5 main features(continued)

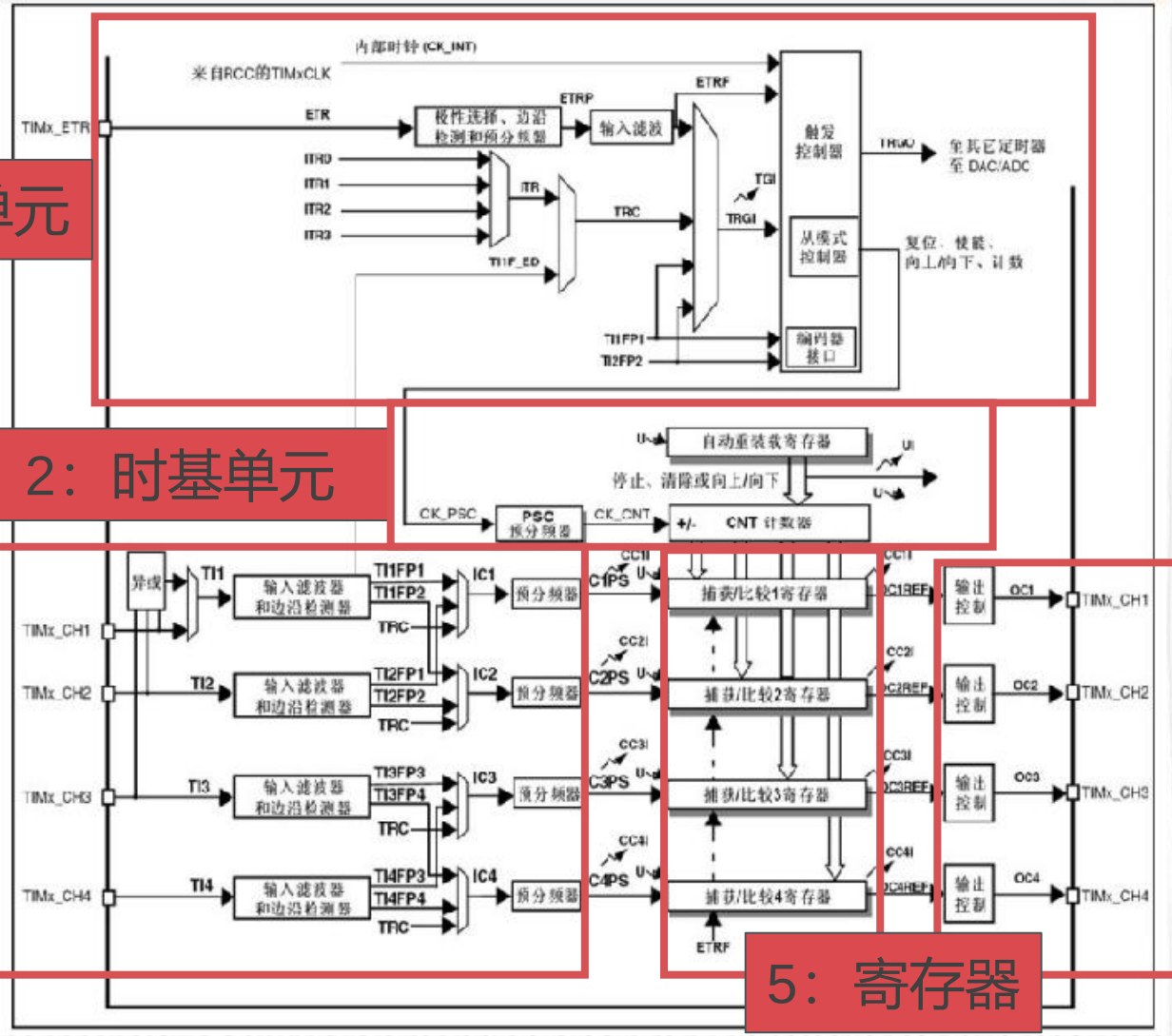
- 可使用外部信号（TIMx_ETR）控制定时器和定时器互连（可以用 1 个定时器控制另外一个定时器）的同步电路
- 6个独立的IRQ/DMA请求生成器，对于更新、触发事件、输入捕获、输出比较事件发生时产生中断/DMA请求
- 支持针对定位的增量(正交)编码器和霍尔传感器电路
- 触发输入作为外部时钟或者按周期的电流管理

1. General-purpose Timers Description

-- TIM2 to TIM5 main features(continued)

- STM32 的通用定时器可以被用于：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)等
- 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
- STM32 的每个通用定时器都是完全独立的，没有互相共享的任何资源

1. General-purpose Timers Description
-- General-purpose timer block diagram



1：时钟选择单元

2：时基单元

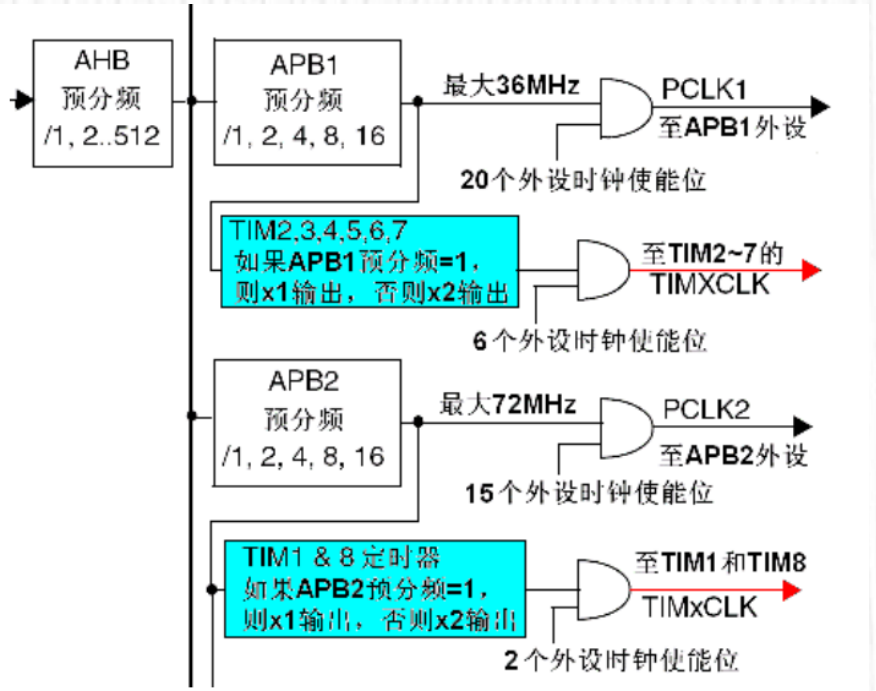
3： 输入捕获单元

4：输出比较单元

5：寄存器

1. General-purpose Timers Description -- Clock selection

- 计数器时钟可以由下列时钟源提供
  - 内部时钟(CK_INT)
  - 外部时钟模式1：外部输入脚(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，如可以配置一个定时器Timer1而作为另一个定时器Timer2的预分频器。



1. General-purpose Timers Description

-- Time-base unit

- 可编程通用定时器的主要部分是一个16位计数器和与其相关的自动装载寄存器。这个计数器可以向上计数、向下计数或者向上向下双向计数。此计数器时钟由预分频器分频得到。
- 计数器、自动装载寄存器和预分频器寄存器可以由软件读写，在计数器运行时仍可以读写。
- 时基单元包含
  - 计数器寄存器(TIMx_CNT)
  - 预分频器寄存器 (TIMx_PSC)
  - 自动装载寄存器 (TIMx_ARR)

1. General-purpose Timers Description

-- TIMx_ARR

- 自动装载寄存器是预先装载的，写或读自动重装载寄存器将访问预装载寄存器。根据在TIMx_CR1寄存器中的自动装载预装载使能位(ARPE)的设置，预装载寄存器的内容被立即或在每次的更新事件UEV时传送到影子寄存器。当计数器达到溢出条件(向下计数时的下溢条件)并当TIMx_CR1寄存器中的UDIS位等于’0’时，产生更新事件。更新事件也可以由软件产生。

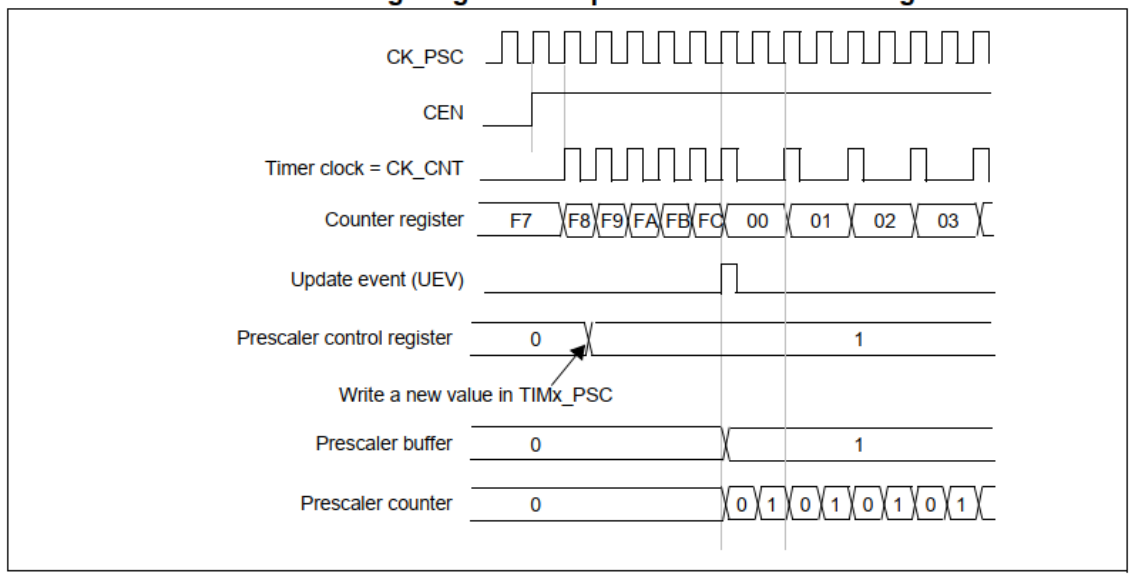
1. General-purpose Timers Description

-- Prescaler description

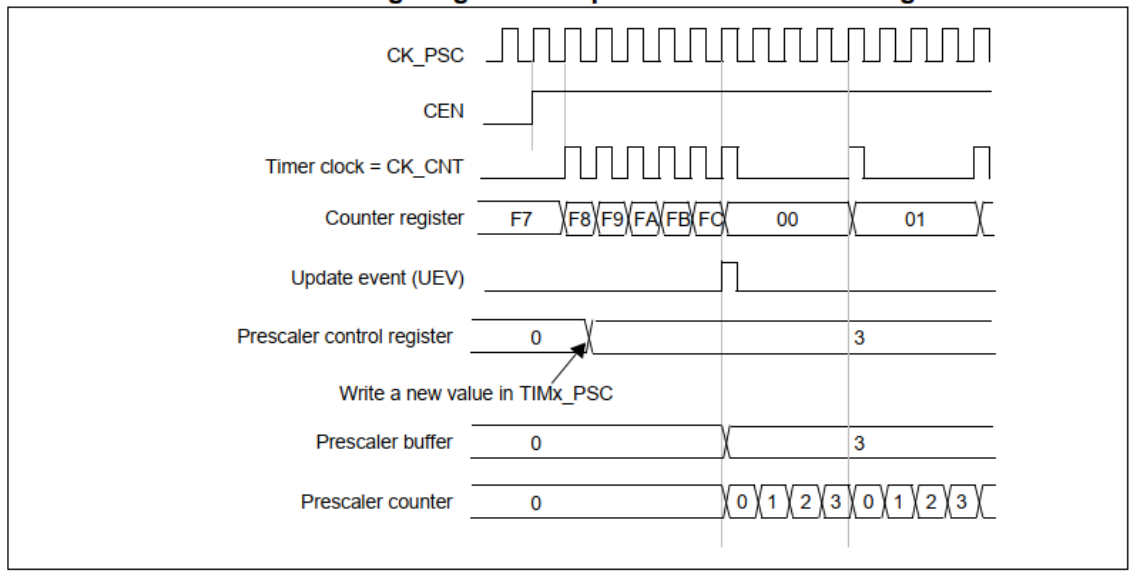
- 预分频器可以将计数器的时钟频率按1到65536之间的任意值分频。它是基于一个(在TIMx_PSC寄存器中的)16位寄存器控制的16位计数器。这个控制寄存器带有缓冲器，它能够在工作时被改变。新的预分频器参数在下一次更新事件到来时被采用

1. General-purpose Timers Description -- Prescaler description(continued)

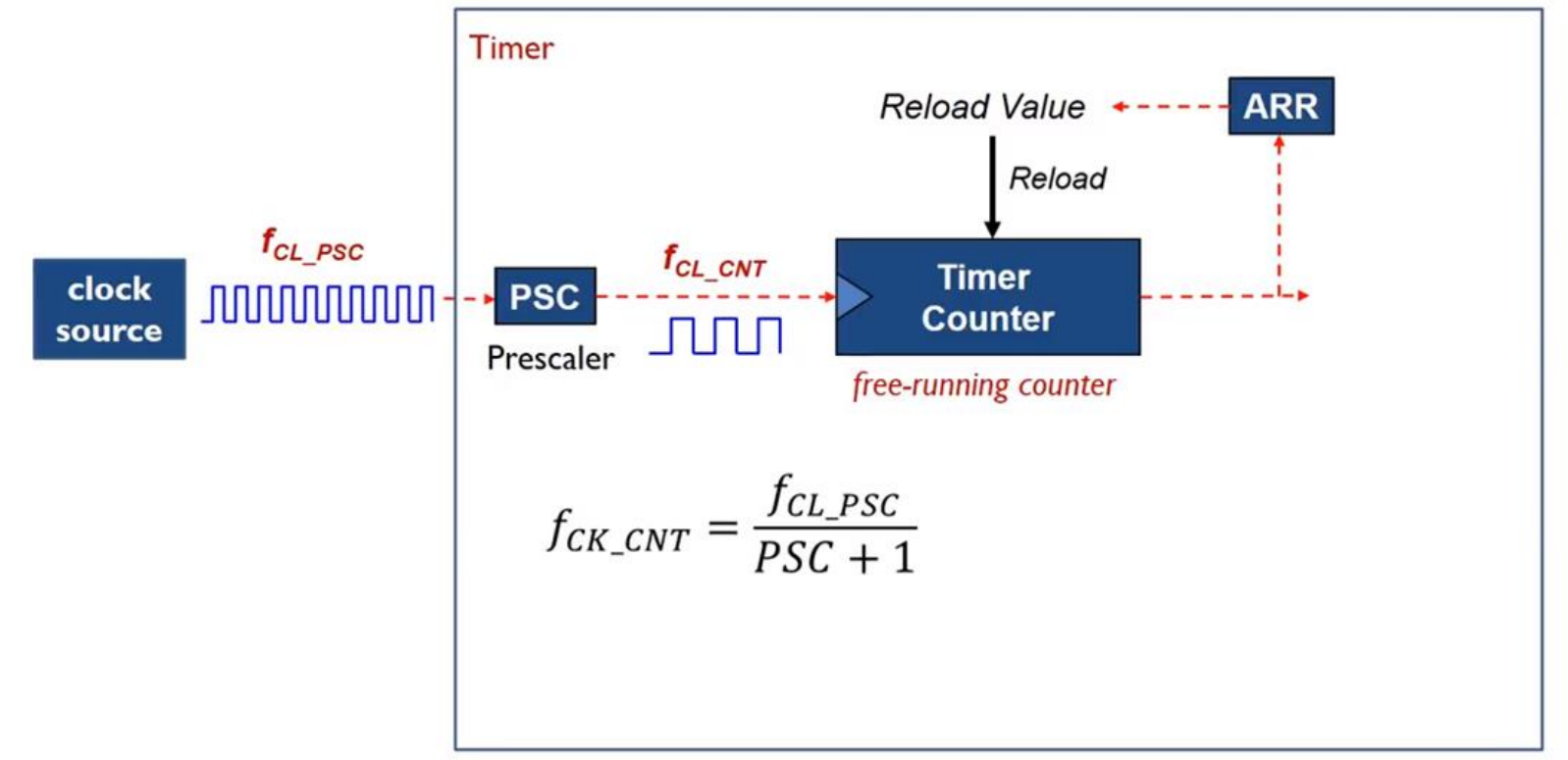
Counter timing diagram with prescaler division change from 1 to 2



Counter timing diagram with prescaler division change from 1 to 4



1. General-purpose Timers Description -- CK_CNT

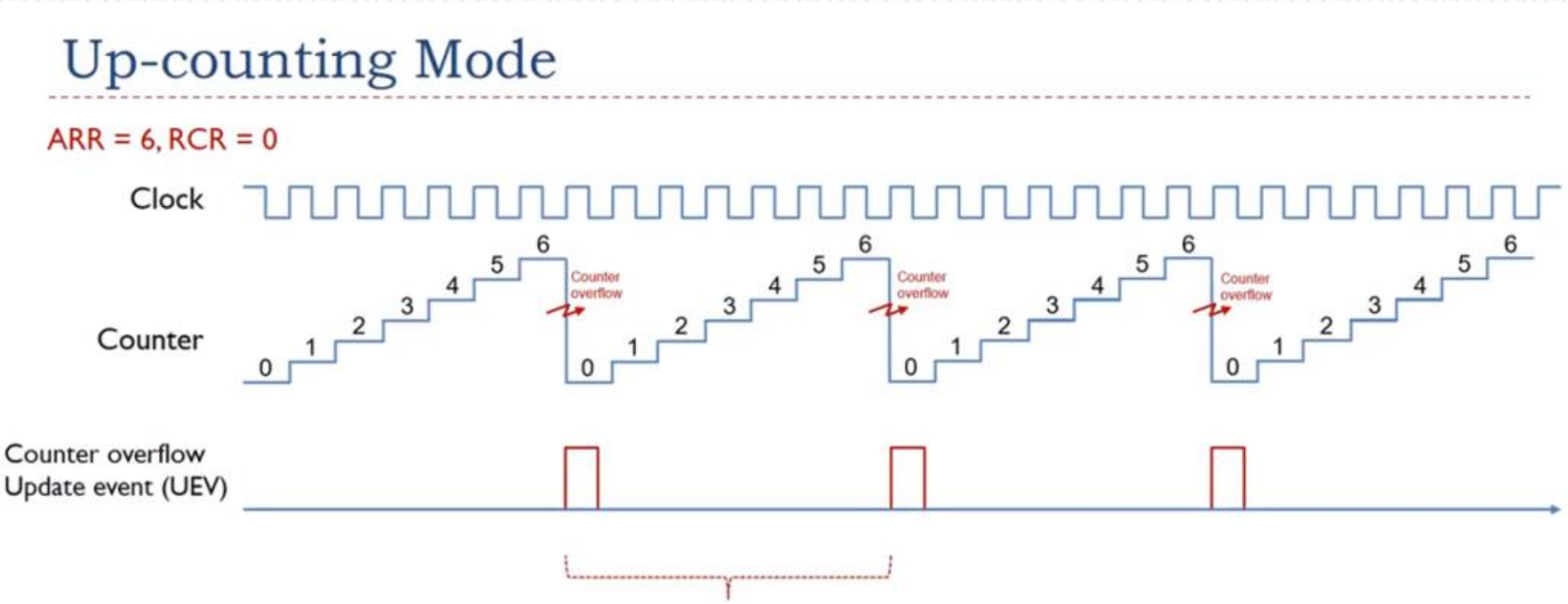


1. General-purpose Timers Description

-- Up-counting mode

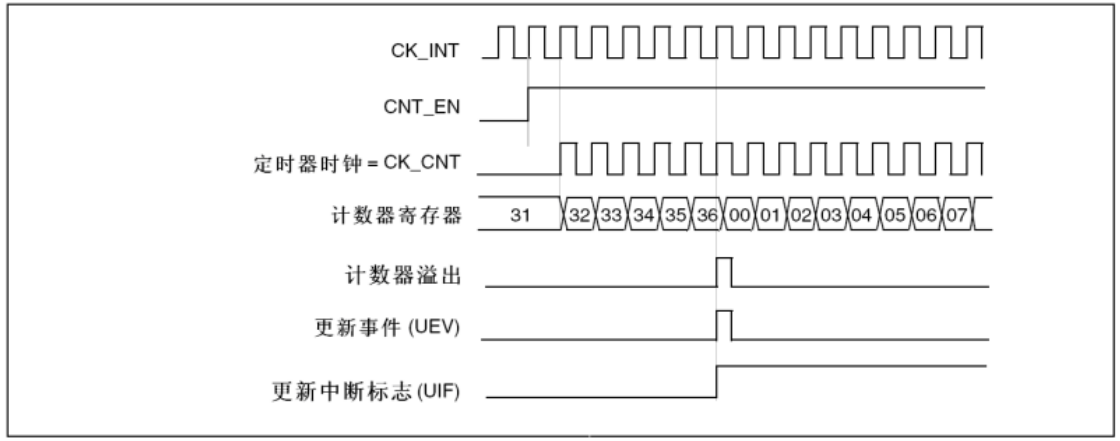
- 在向上计数模式中，计数器从0计数到自动加载值（TIMx_ARR计数器的内容），然后重新从0开始计数并且产生一个计数器溢出事件
- 每次计数器溢出时可以产生更新事件，在TIMx_EGR寄存器中(通过软件方式或者使用从模式控制器)设置UG位，也同样可以产生一个更新事件
- 发生一个更新事件时，所有的寄存器都被更新，硬件同时(依据URS位)设置更新标志位(TIMx_SR寄存器中的UIF位)
  - 预分频器的缓冲区被置入预装载寄存器的值(TIMx_PSC寄存器的内容)
  - 自动装载影子寄存器被重新置入预装载寄存器的值(TIMx_ARR)

1. General-purpose Timers Description -- Up-counting mode(continued)

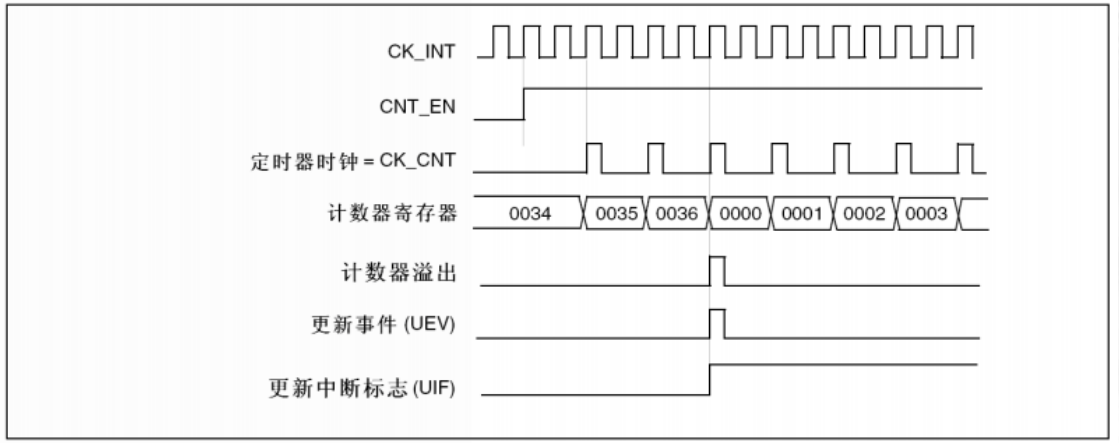


1. General-purpose Timers Description -- Up-counting mode(continued)

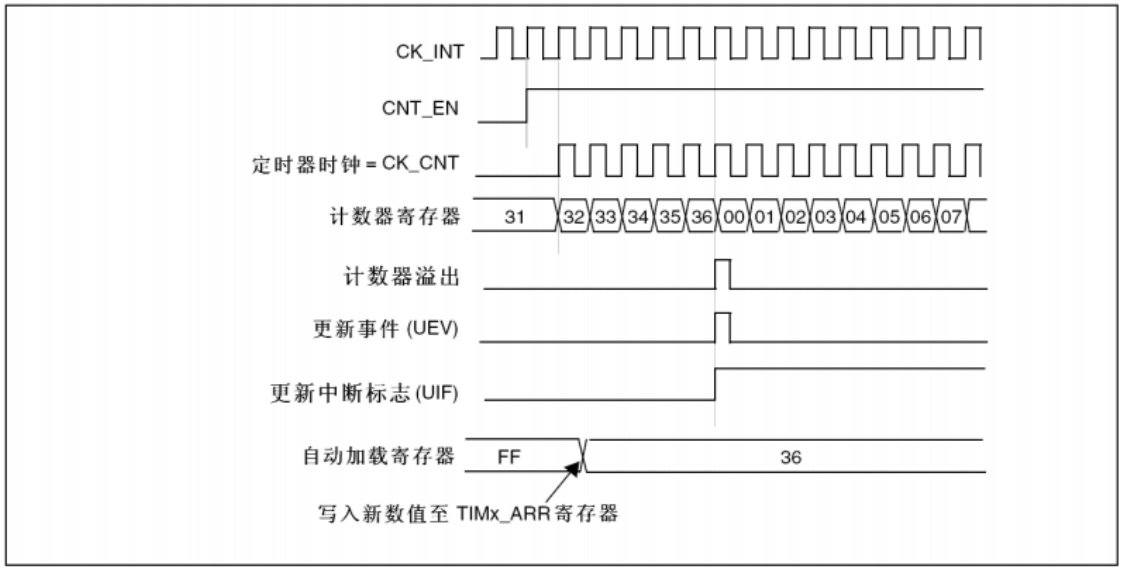
计数器时序图，内部时钟分频因子为1



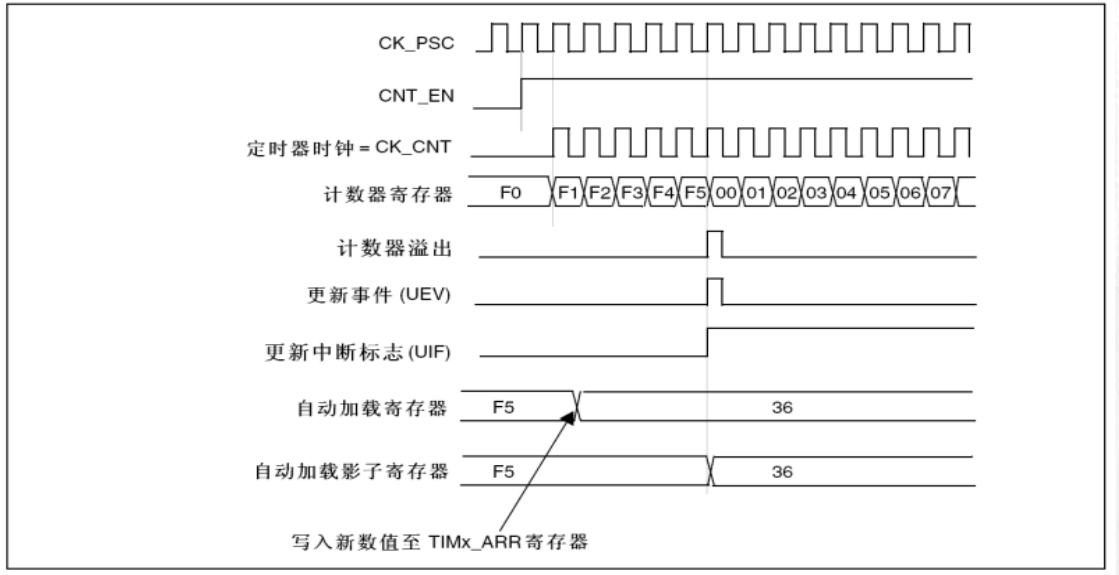
计数器时序图，内部时钟分频因子为2



计数器时序图，当ARPE=0时的更新事件(TIMx_ARR没有预装入)



计数器时序图，当ARPE=1时的更新事件(预装入了TIMx_ARR)

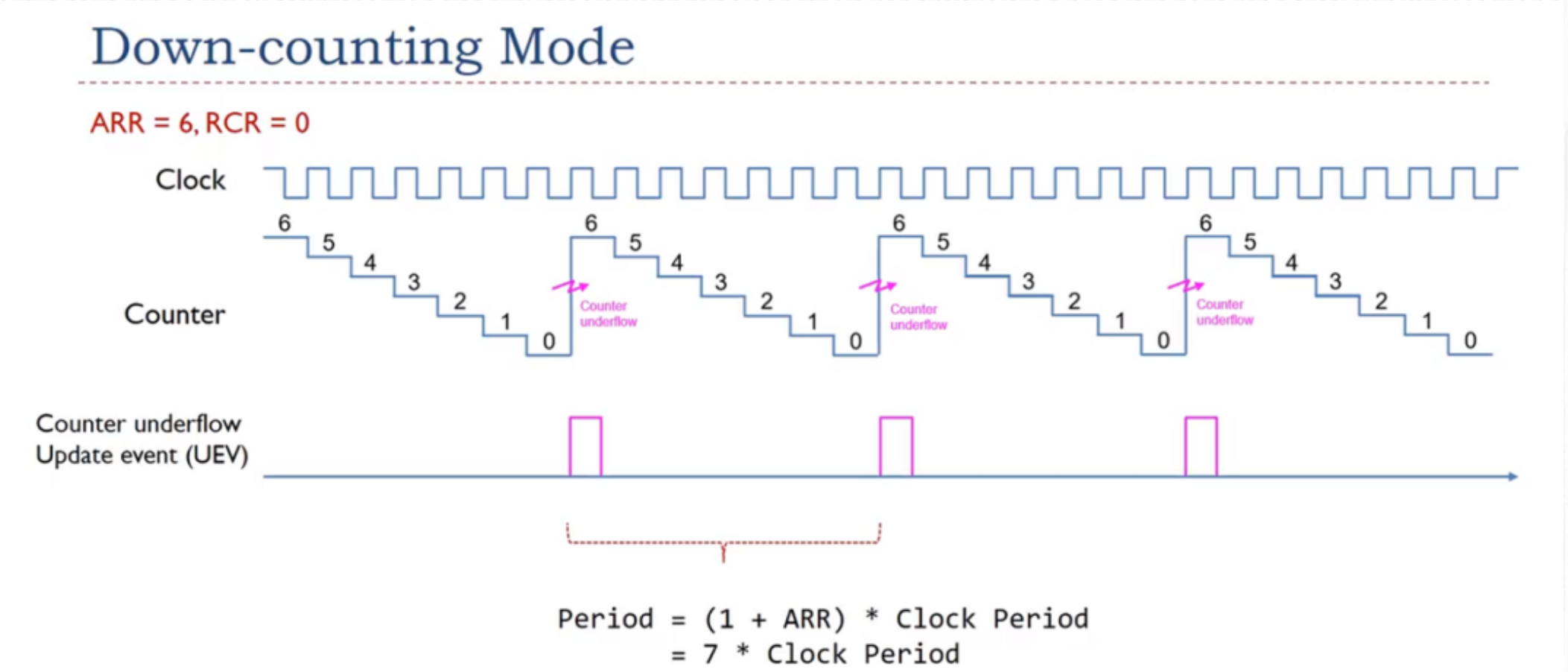


1. General-purpose Timers Description

-- Down-counting mode

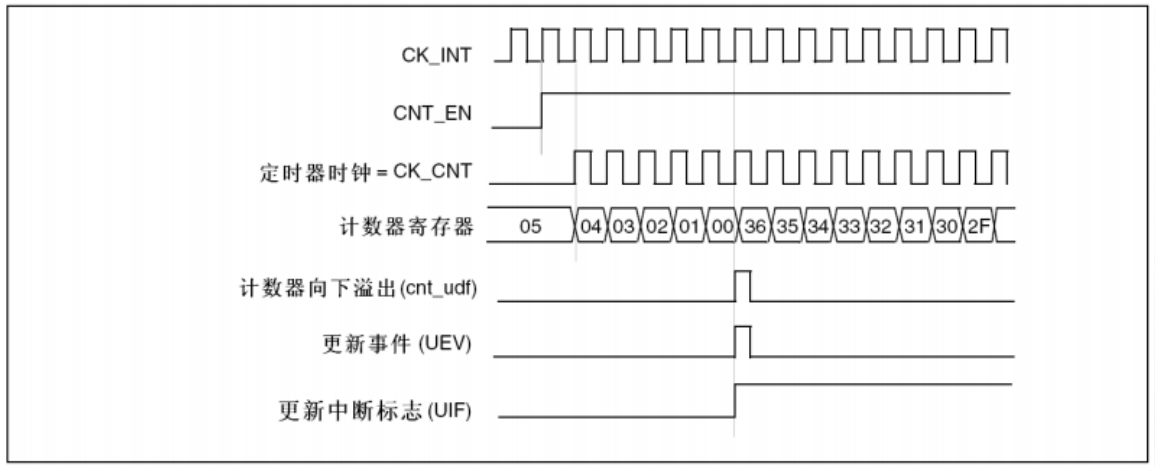
- 在向下计数模式中，计数器从自动加载值（TIMx_ARR计数器的值）计数到0，然后从自动装入的值重新开始计数并且产生一个计数器向下溢出事件
- 每次计数器溢出时可以产生更新事件，在TIMx_EGR寄存器中(通过软件方式或者使用从模式控制器)设置UG位，也同样可以产生一个更新事件
- 发生一个更新事件时，所有的寄存器都被更新，并且设置(根据URS位的设置) 更新标志位(TIMx_SR寄存器的UIF位)
  - 预分频器的缓冲区被置入预装载寄存器的值(TIMx_PSC寄存器的值)
  - 当前的自动加载寄存器被更新为预装载值(TIMx_ARR寄存器中的值)

1. General-purpose Timers Description -- Down-counting mode

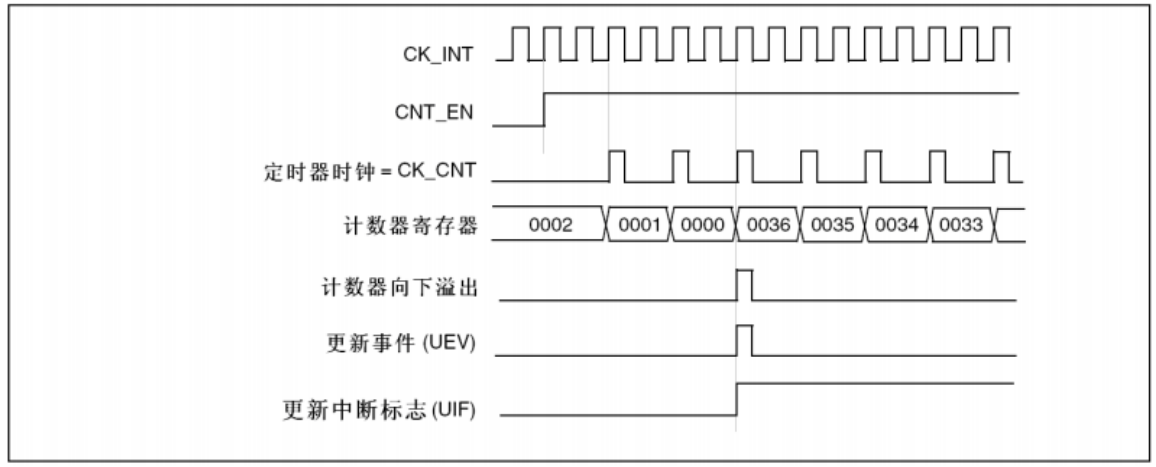


1. General-purpose Timers Description -- Down-counting mode

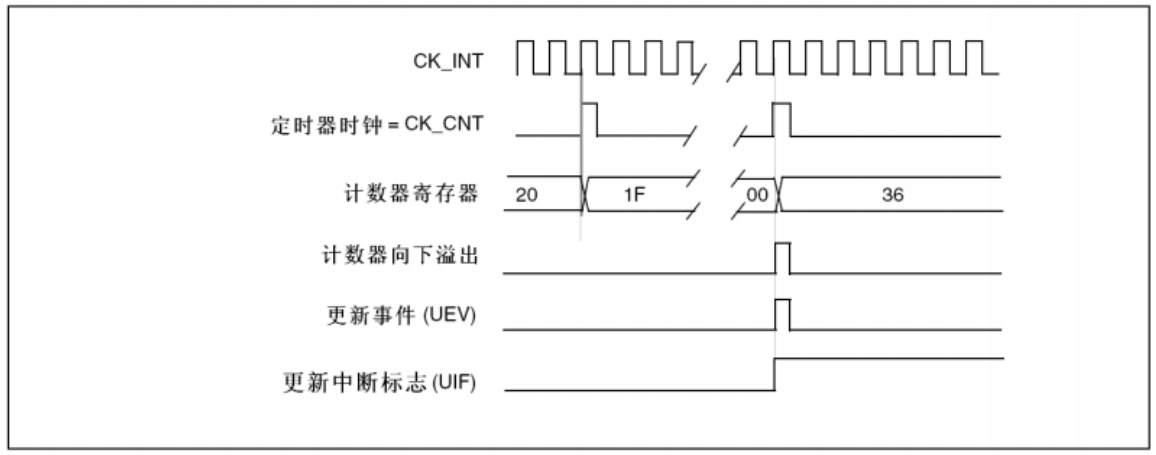
计数器时序图，内部时钟分频因子为1



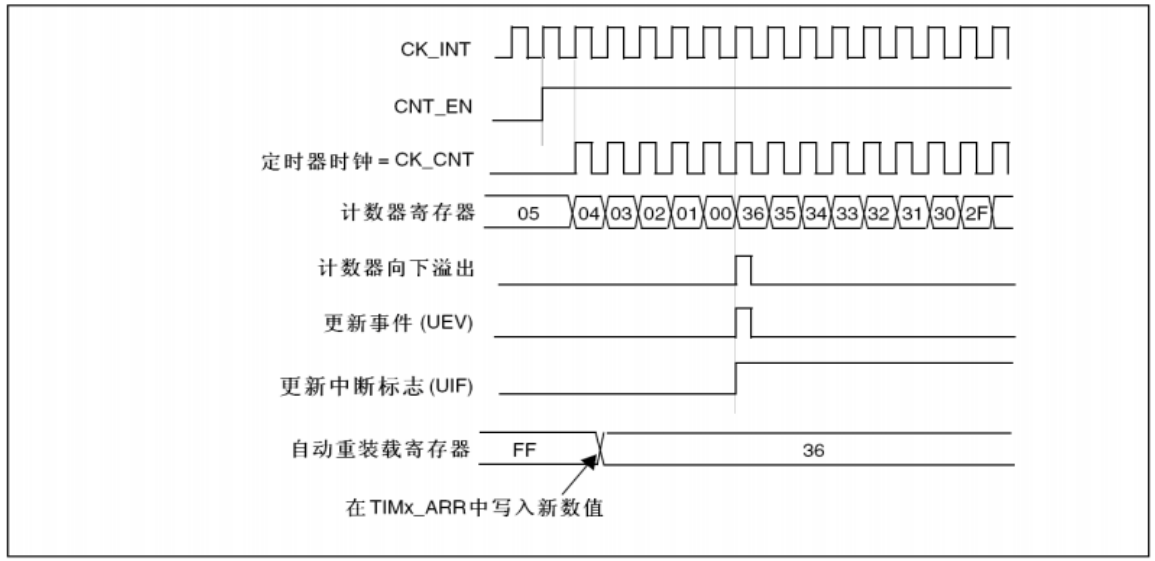
计数器时序图，内部时钟分频因子为2



计数器时序图，内部时钟分频因子为N



计数器时序图，当没有使用重复计数器时的更新事件

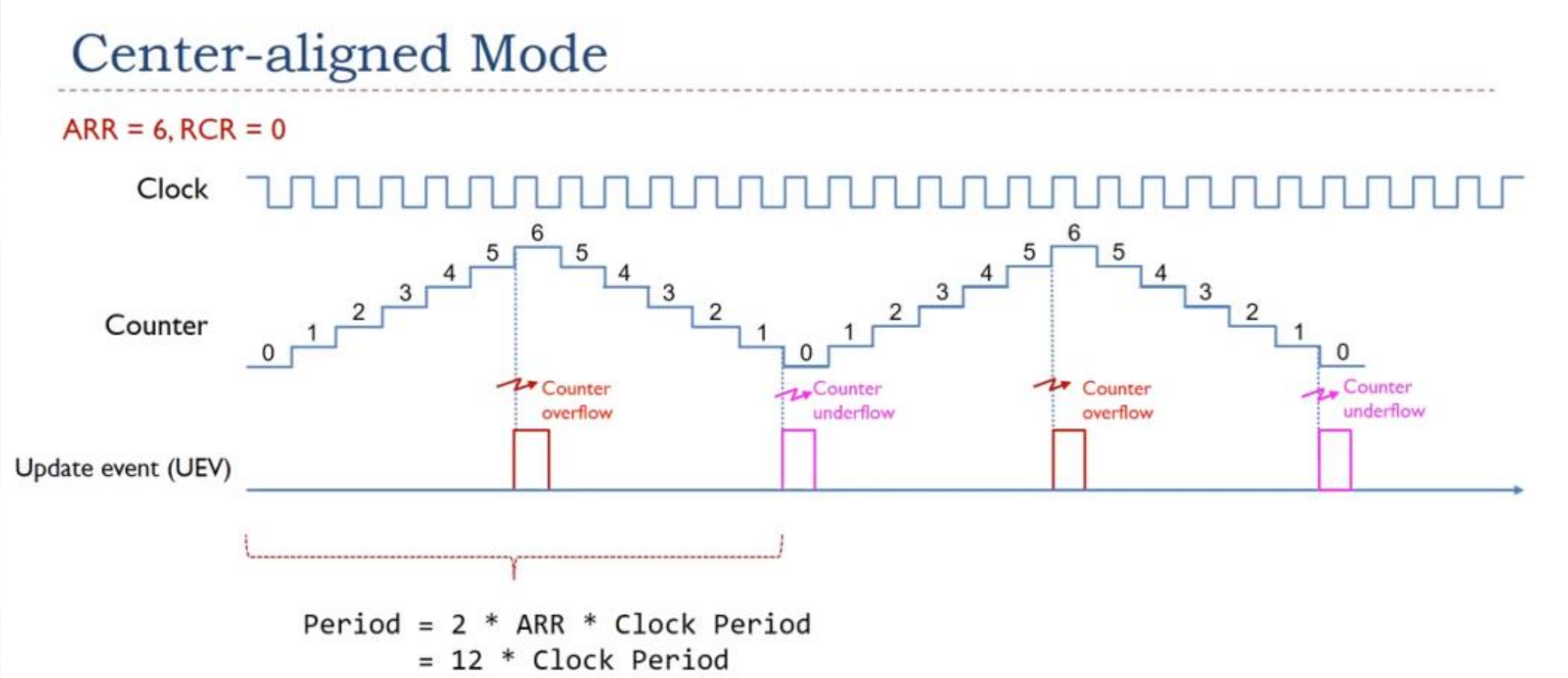


1. General-purpose Timers Description

-- Center-aligned mode

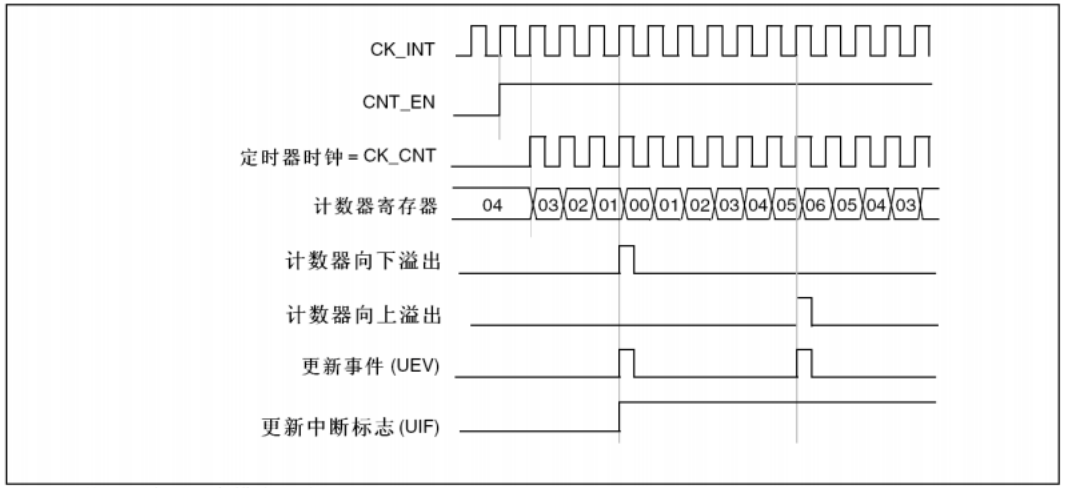
- 在中央对齐(向上/向下计数)模式，计数器从0开始计数到自动加载的值(TIMx_ARR寄存器)-1，产生一个计数器溢出事件，然后向下计数到1并且产生一个计数器下溢事件；然后再从0开始重新计数。
- 在这个模式，不能写入TIMx_CR1中的DIR方向位。它由硬件更新并指示当前的计数方向。
- 可以在每次计数上溢和每次计数下溢时产生更新事件；也可以通过(软件或者使用从模式控制器)设置TIMx_EGR寄存器中的UG位产生更新事件。然后，计数器重新从0开始计数，预分频器也重新从0开始计数。

1. General-purpose Timers Description -- Center-aligned mode

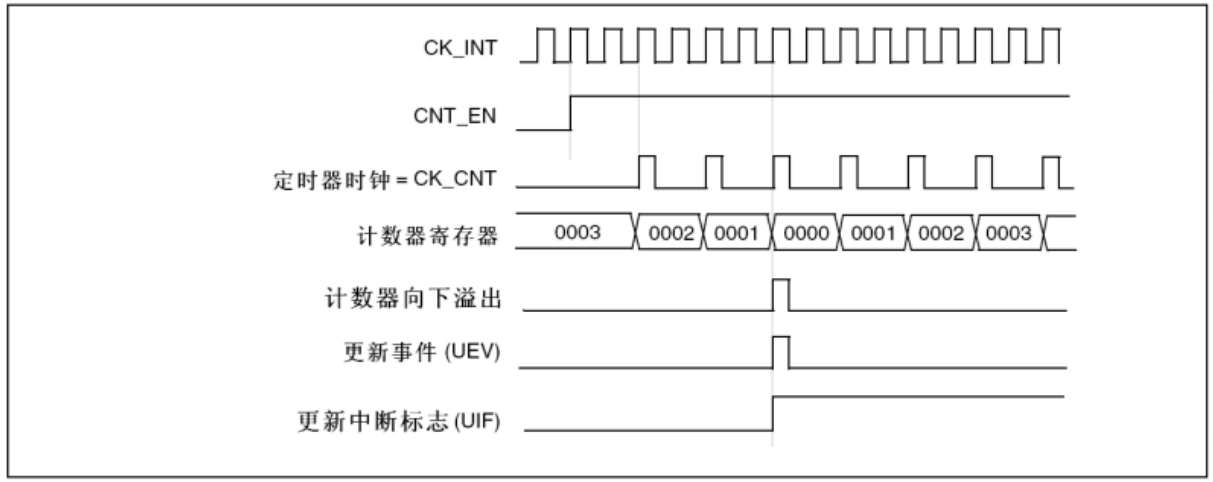


1. General-purpose Timers Description -- Center-aligned mode

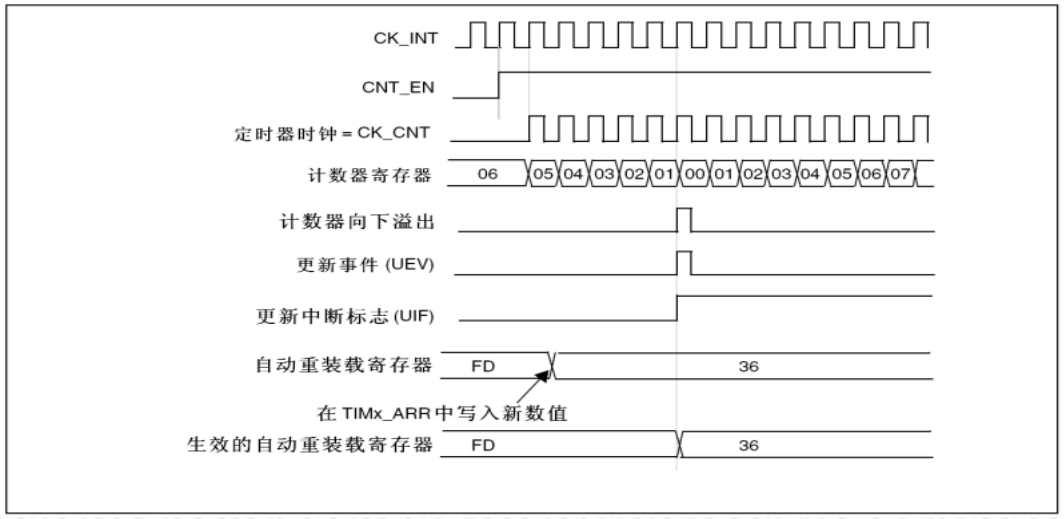
计数器时序图，内部时钟分频因子为1，TIMx_ARR=0x6



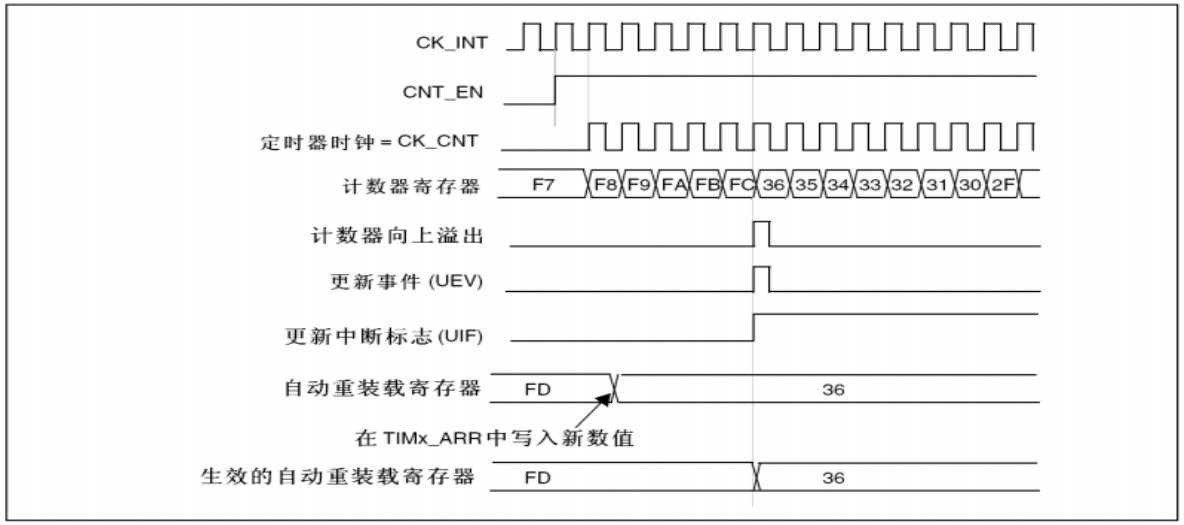
计数器时序图，内部时钟分频因子为2



计数器时序图，ARPE=1时的更新事件(计数器下溢)



计数器时序图，ARPE=1时的更新事件(计数器溢出)



02

TIM2 to TIM5 Registers

2. TIM2 to TIM5 Registers -- TIMx_CNT

- 计数器当前值寄存器

计数器(TIMx_CNT)

偏移地址：0x24

复位值：0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CNT[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

位15:0	**CNT[15:0]**：计数器的值 (Counter value)

2. TIM2 to TIM5 Registers
-- TIMx_PSC

- 预分频寄存器

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PSC[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

位15:0	**PSC[15:0]**：预分频器的值 (Prescaler value) 计数器的时钟频率CK_CNT等于f_{CK_PSC}/(PSC[15:0]+1)。 PSC包含了当更新事件产生时装入当前预分频器寄存器的值。

2. TIM2 to TIM5 Registers
-- TIMx_ARR

- 自动重装载寄存器

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ARR[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

位15:0	**ARR[15:0]**: 自动重装载的值 (Auto reload value) ARR包含了将要传送至实际的自动重装载寄存器的数值。 详细参考14.3.1节：有关ARR的更新和动作。 当自动重装载的值为空时，计数器不工作。

2. TIM2 to TIM5 Registers -- TIMx_CR1

- 控制寄存器1

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 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。 在单脉冲模式下，当发生更新事件时，CEN被自动清除。

2. TIM2 to TIM5 Registers
-- TIMx_CR1(continued)

位15:10	保留，始终读为0。
位9:8	**CKD[1:0]**: 时钟分频因子 (Clock division) 定义在定时器时钟(CK_INT)频率与数字滤波器(ETR，TIx)使用的采样频率之间的分频比例。 00：$t_{DTS} = t_{CK\_INT}$ 01：$t_{DTS} = 2 \times t_{CK\_INT}$ 10：$t_{DTS} = 4 \times t_{CK\_INT}$ 11：保留
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6:5	**CMS[1:0]**：选择中央对齐模式 (Center-aligned mode selection) 00：边沿对齐模式。计数器依据方向位(DIR)向上或向下计数。 01：中央对齐模式1。计数器交替地向上和向下计数。配置为输出的通道(TIMx_CCMRx寄存器中CCxS=00)的输出比较中断标志位，只在计数器向下计数时被设置。 10：中央对齐模式2。计数器交替地向上和向下计数。配置为输出的通道(TIMx_CCMRx寄存器中CCxS=00)的输出比较中断标志位，只在计数器向上计数时被设置。 11：中央对齐模式3。计数器交替地向上和向下计数。配置为输出的通道(TIMx_CCMRx寄存器中CCxS=00)的输出比较中断标志位，在计数器向上和向下计数时均被设置。 注：在计数器开启时(CEN=1)，不允许从边沿对齐模式转换到中央对齐模式。

2. TIM2 to TIM5 Registers
-- TIMx_DIER

- DMA中断使能寄存器

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
保留	TDE	保留	CC4DE	CC3DE	CC2DE	CC1DE	UDE	保留	TIE	保留	CC4IE	CC3IE	CC2IE	CC1IE	UIE
	rw		rw	rw	rw	rw	rw		rw		rw	rw	rw	rw	rw

位5	保留，始终读为0。
位4	**CC4IE**：允许捕获/比较4中断 (Capture/Compare 4 interrupt enable) 0：禁止捕获/比较4中断； 1：允许捕获/比较4中断。
位3	**CC3IE**：允许捕获/比较3中断 (Capture/Compare 3 interrupt enable) 0：禁止捕获/比较3中断； 1：允许捕获/比较3中断。
位2	**CC2IE**：允许捕获/比较2中断 (Capture/Compare 2 interrupt enable) 0：禁止捕获/比较2中断； 1：允许捕获/比较2中断。
位1	**CC1IE**：允许捕获/比较1中断 (Capture/Compare 1 interrupt enable) 0：禁止捕获/比较1中断； 1：允许捕获/比较1中断。
位0	**UIE**：允许更新中断 (Update interrupt enable) 0：禁止更新中断； 1：允许更新中断。

2. TIM2 to TIM5 Registers
-- Timer Interrupt Configuration Steps

- 使能定时器时钟
- 初始化定时器，配置ARR，PSC
- 开启定时器中断，配置NVIC
- 使能定时器
- 编写中断服务函数

南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

03

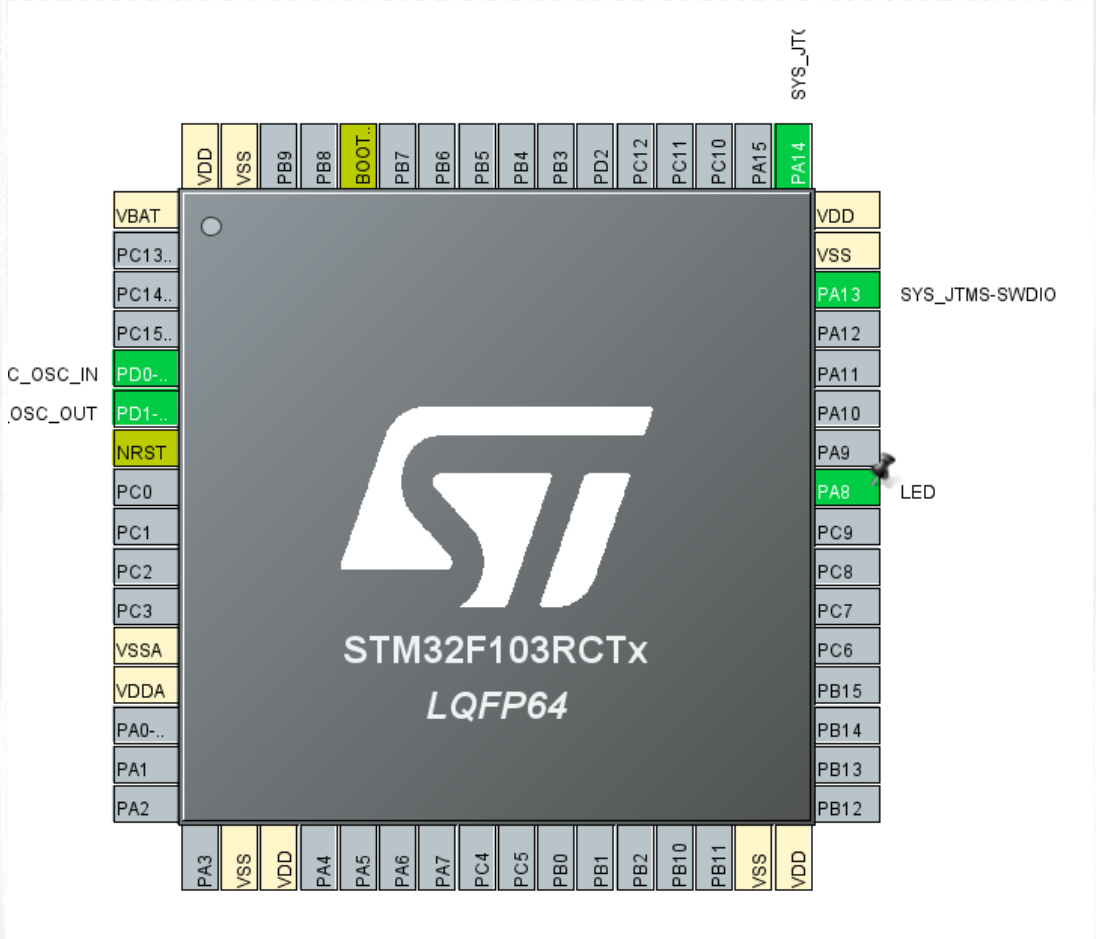
How to Program

3. How to Program

- Our Goal
  - Use timer interrupt to blink the LED every 1 second
  - Do not use any delay function

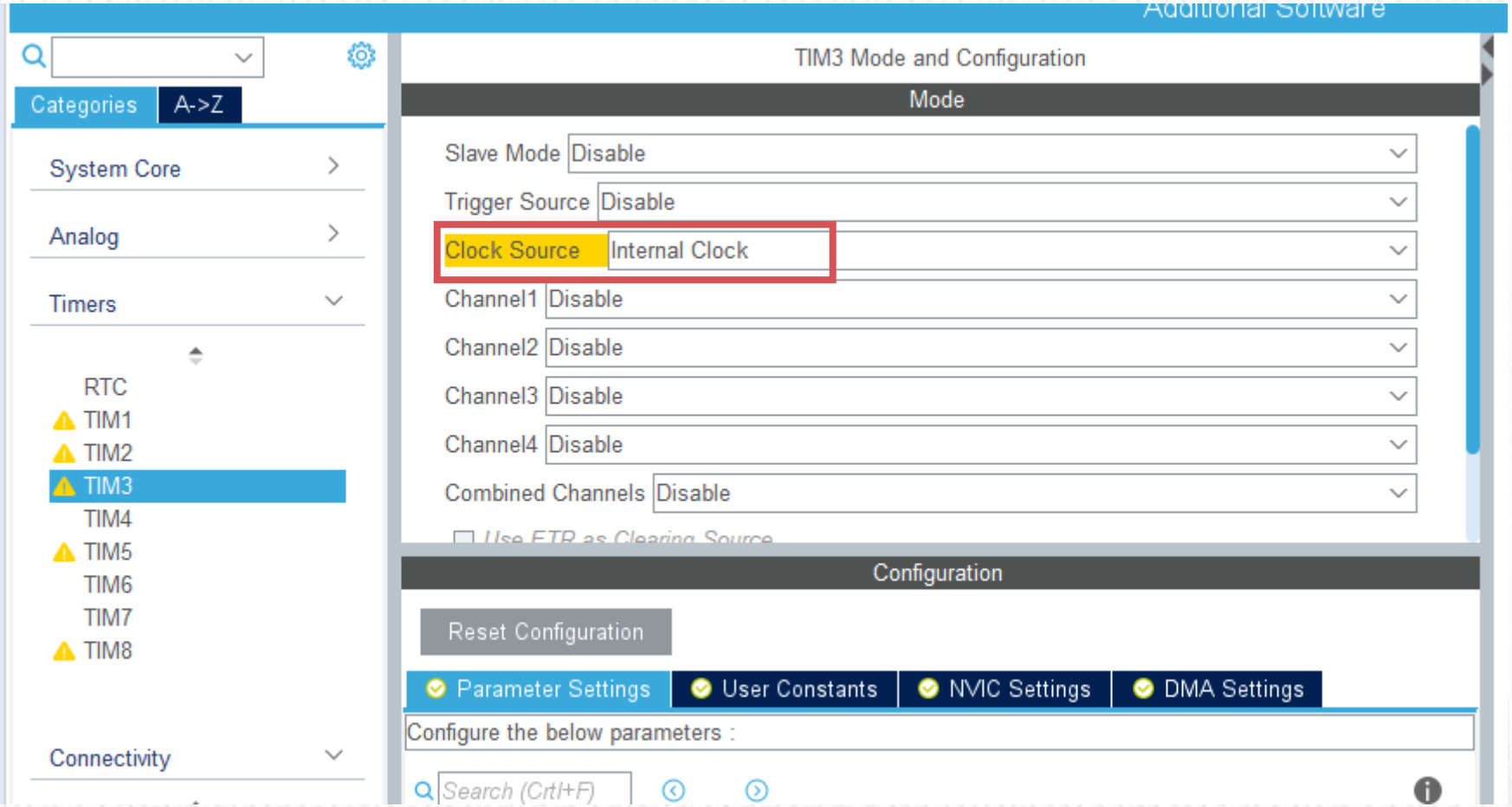
3. How to Program

- Configure GPIO: use either led is OK



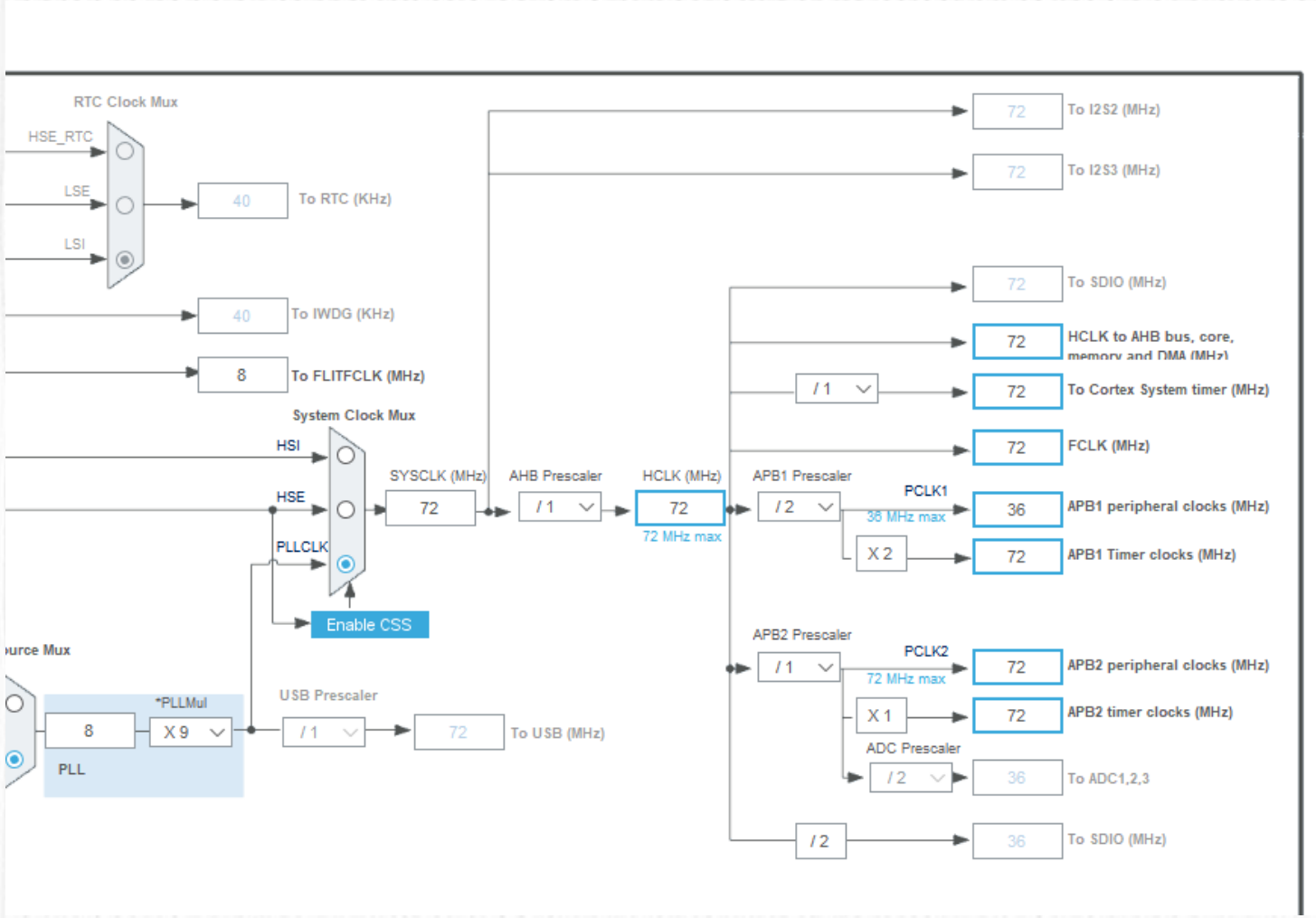
3. How to Program

- Configure TIM3 in STM32CubeIDE
  - Select the clock source as **Internal Clock**



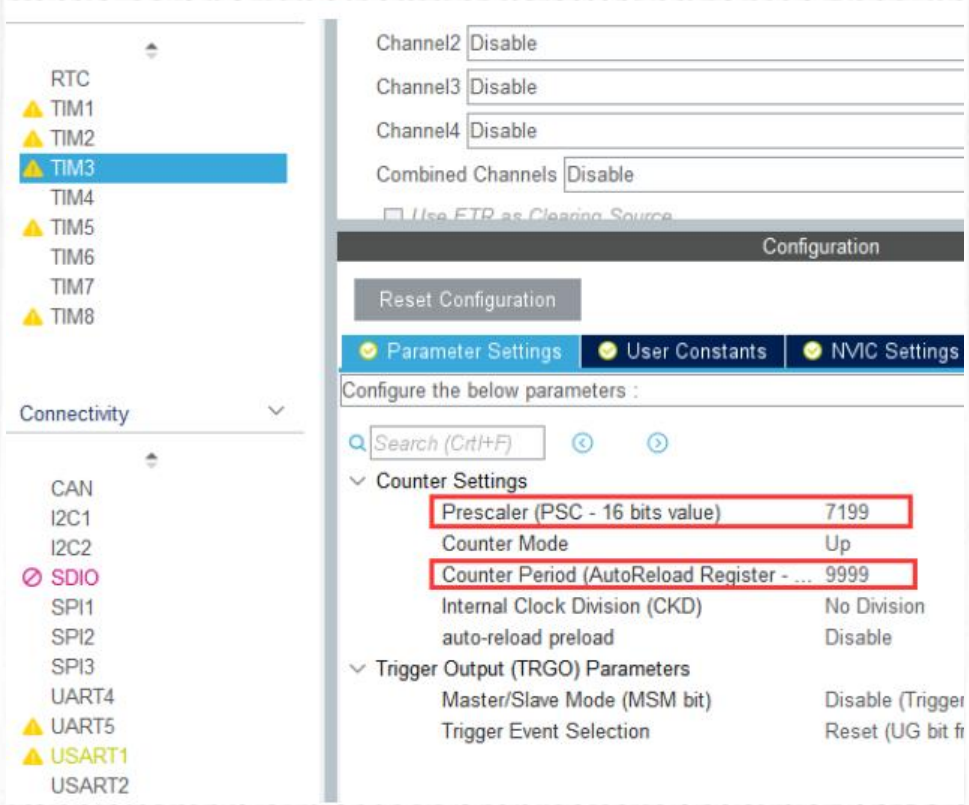
3. How to Program

- What is the frequency of the Internal Clock?
  - SYCLK is 72MHz
  - f_{AHB} = 72MHz
  - f_{APB1} = 36MHz
  - APB1 Prescaler = f_{AHB} / f_{APB1} = 2
  - TIM3 is connected on the APB1 bus
  - f_{ck_psc} = f_{APB1} * 2 = 72MHz



3. How to Program

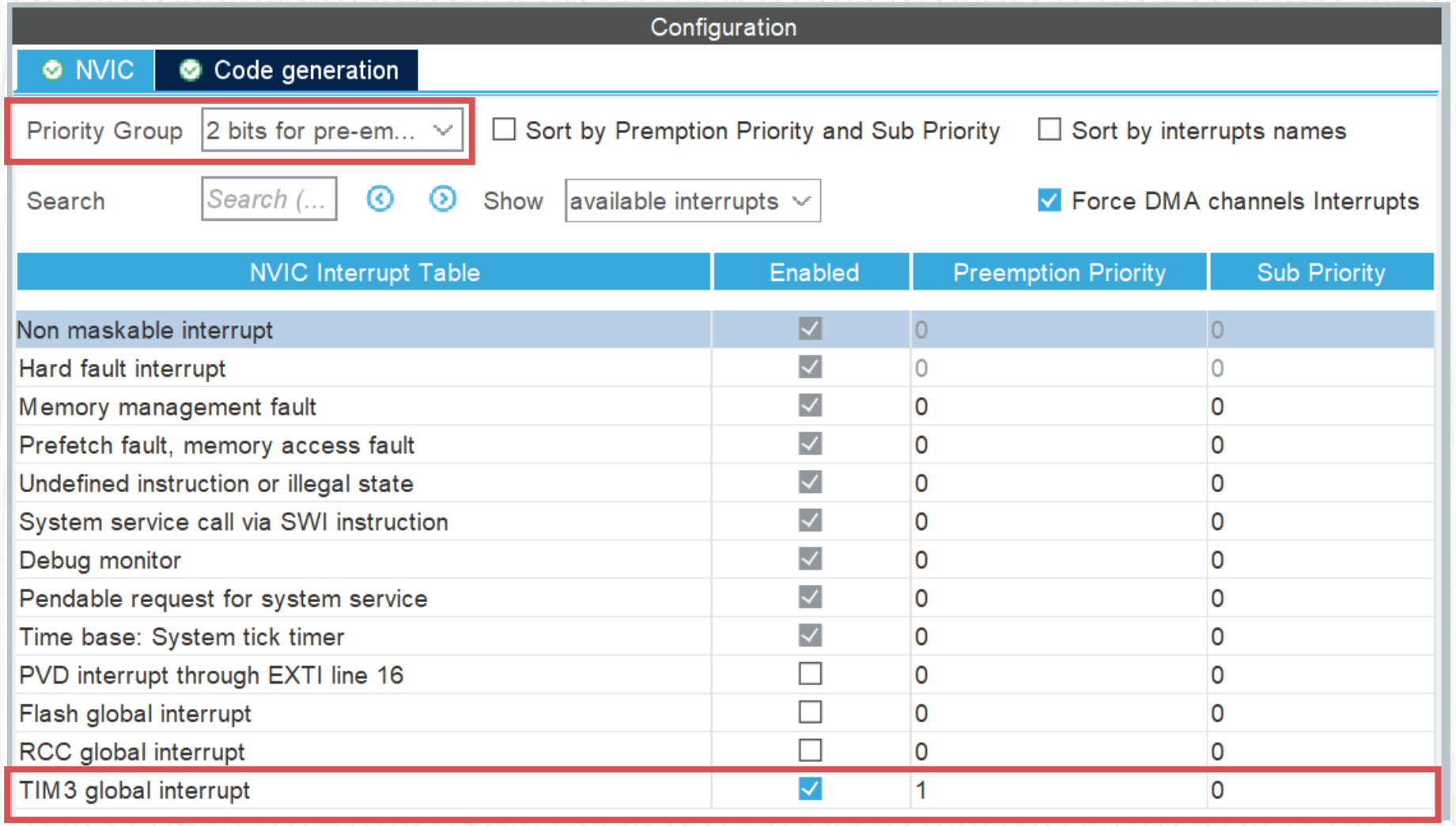
- Counter setting
- $f_{ck\_cnt} = f_{ck\_psc} / (arr+1)(psc+1)$
- $T = (arr+1)(psc+1)/ f_{ck\_psc}$
- where:
  - T is the update interrupt period(us)
  - f_{ck_psc} is the frequency of the internal clock(MHz)
  - arr is the auto-reload value
  - psc is the prescaler



- As we know, f_{ck_psc} = 72Mhz, if we want to set T as 1 second, we can set arr = 9999 and psc = 7199. Other settings will be all right as long as both ranges from 1 to 65535.

3. How to Program

- Configure NVIC in STM32CubeIDE



3. How to Program

- Some functions we used
  - Call the following function to start/stop timer in interrupt mode(for example in the main routine)

```
HAL_StatusTypeDef HAL_TIM_Base_Start_IT(TIM_HandleTypeDef *htim)
HAL_StatusTypeDef HAL_TIM_Base_Stop_IT(TIM_HandleTypeDef *htim)
```

```
/* USER CODE BEGIN PV */
extern TIM_HandleTypeDef htim3;
/* USER CODE END PV */
```

```
int main(void)
{
  //...
  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_TIM3_Init();
  MX_USART1_UART_Init();
  /* USER CODE BEGIN 2 */
  HAL_TIM_Base_Start_IT(&htim3);
  /* USER CODE END 2 */
  //...
}
```

3. How to Program

- Some functions we used
  - All the interrupts of TIM3 are handled by TIM3_IRQHandler in stm32f1xx_it.c, it calls the public TIM interrupt handler HAL_TIM_IRQHandler, which will call the corresponding callback function according to the interrupt type and clear the corresponding interrupt pending bits.

```c
/**
  * @brief This function handles TIM3 global interrupt.
  */
void TIM3_IRQHandler(void)
{
  /* USER CODE BEGIN TIM3_IRQn 0 */

  /* USER CODE END TIM3_IRQn 0 */
  HAL_TIM_IRQHandler(&htim3);
  /* USER CODE BEGIN TIM3_IRQn 1 */

  /* USER CODE END TIM3_IRQn 1 */
}
```

```c
void HAL_TIM_IRQHandler(TIM_HandleTypeDef *htim)
{
  //...
  /* TIM Update event */
  if (__HAL_TIM_GET_FLAG(htim, TIM_FLAG_UPDATE) != RESET)
  {
    if (__HAL_TIM_GET_IT_SOURCE(htim, TIM_IT_UPDATE) != RESET)
    {
      __HAL_TIM_CLEAR_IT(htim, TIM_IT_UPDATE);
#if (USE_HAL_TIM_REGISTER_CALLBACKS == 1)
      htim->PeriodElapsedCallback(htim);
#else
      HAL_TIM_PeriodElapsedCallback(htim);
#endif /* USE_HAL_TIM_REGISTER_CALLBACKS */
    }
  }
  //..
}
```

```c
/**
  * @brief  Period elapsed callback in non-blocking mode
  * @param  htim TIM handle
  * @retval None
  */
__weak void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
  /* Prevent unused argument(s) compilation warning */
  UNUSED(htim);

  /* NOTE : This function should not be modified, when the callback
            the HAL_TIM_PeriodElapsedCallback could be implemented i
file
   */
}
```

南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

04

Practice

4. Practice

- Configure a timer (except TIM3) triggers an interrupt with period of 0.5s and use it to blink the LED
- Re-implement the HAL_TIM_PeriodElapsedCallback()
- Do not use any delay function