STM32 定时器基本原理及常见问题之培训资料v3.10

时基单元、捕捉比较功能、主从触发与级联、案例分享

Edited by Miler Shao

Nov 2018

life.augmented

培训内容

- STM32定时器概述
- 定时器时基单元
- 定时器输入捕捉功能
- 定时器比较输出功能
- 定时器触发同步与级联
- 定时器DMA批量传送
- 定时器产生的触发输出与其它外设的关联
- STM32F334/SMT32F7/SMT32L4等新增的定时器功能
- 案例分享【穿插在上述内容中】
- 注：针对本PPT内容，如有建议或疑问，可邮件至mcu.china@st.com反馈或交流。

STM32 TIMER 分类概述

- 整体讲，STM32家族的定时器众多，按照核内、核外标准大致

分为两部分：

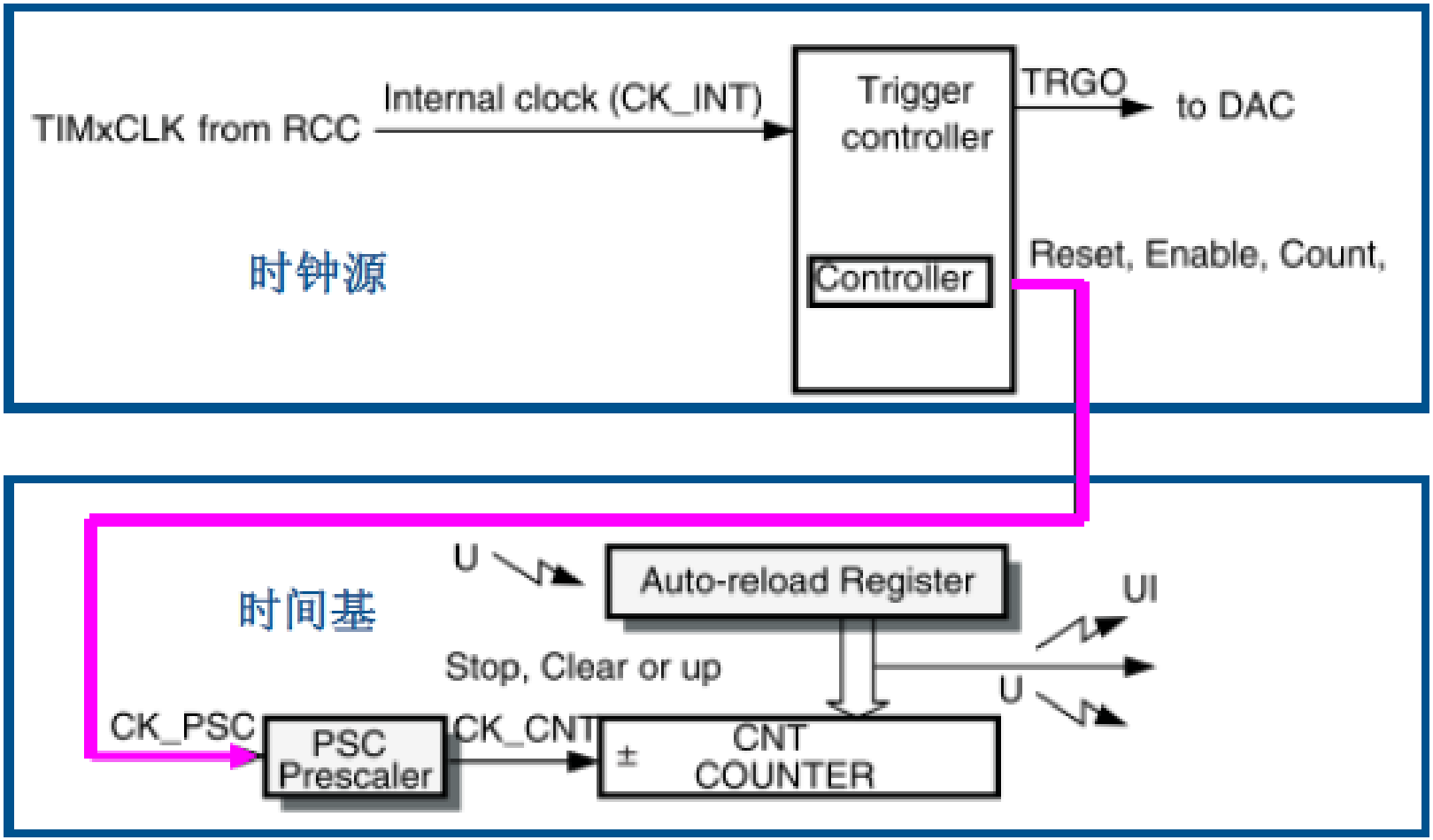
- 核内定时器 + 外设定时器
  - 核内定时器 ： Systick
  - 外设定时器: 特定应用定时器+常规定时器
    - 特定应用定时器：LPTIM;RTC;WTD;HRTIM
    - 常规定时器：基本定时器、通用定时器、高级定时器 【本篇介绍重点】

- 按计数器位宽来分：
  - 16位定时器
  - 32位定时器 【TIM2 TIM5】
  - 24位定时器 【Systick】

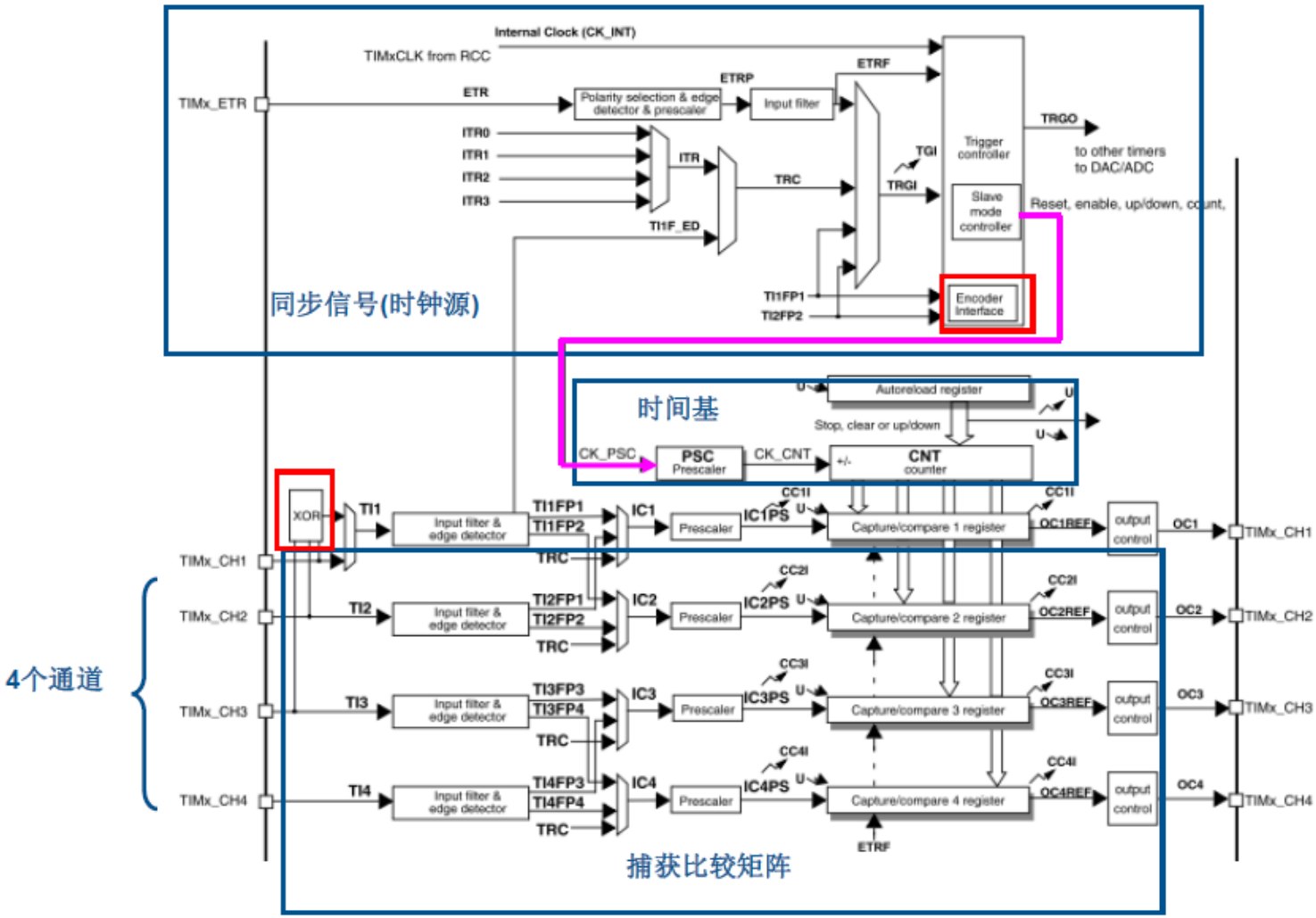
常规定时器

- **基本定时器**：几乎没有任何对外输入/输出，常用作时基，实现基本的计数、定时功能。
- **通用定时器**：除了基本定时器的时基功能外，还可对外做输入捕捉、输出比较以及连接其它传感器接口【编码器和霍尔传感器】。
- **高级定时器**：此类定时器的功能最为强大，除了具备通用定时器的功能外，还包含一些与电机控制和数字电源应用相关的功能，比方带死区控制的互补信号输出、紧急刹车关断输入控制。

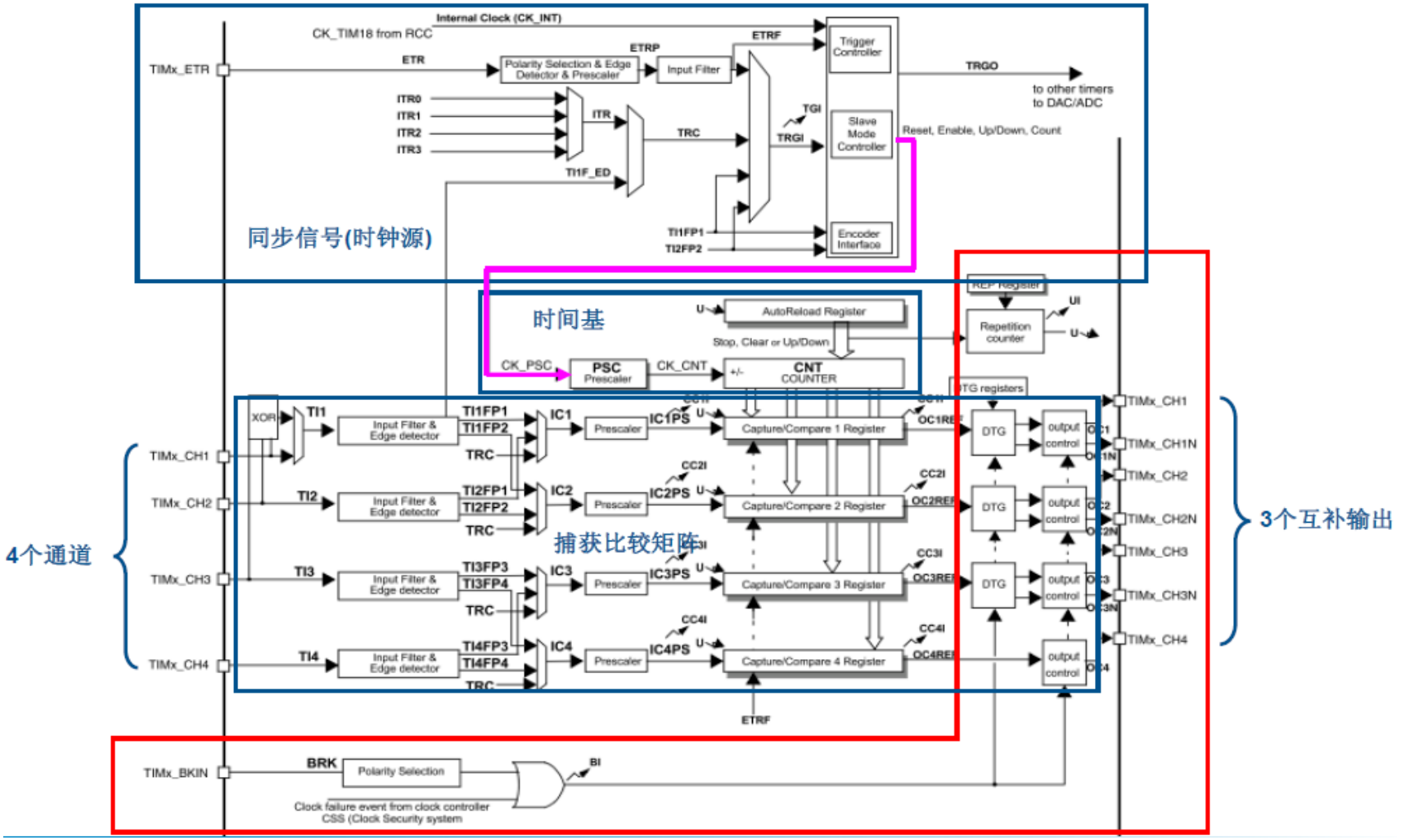
基本定时器框图


TIMxCLK from RCC
Internal clock (CK_INT)
Trigger controller
TRGO
to DAC
时钟源
Controller
Reset, Enable, Count,
时间基
U
Auto-reload Register
UI
Stop, Clear or up
U
CK_PSC
PSC Prescaler
CK_CNT
±
CNT COUNTER

通用定时器框图



高级定时器框图



写在前面

- 1、整体上讲,stm32定时器资料零散而庞大，概念或术语众多，尤其概念的跨度极大，给我们阅读和理解STM32定时器的工作原理带来了很大的困难或困扰。

- 2、先从整体上对STM32定时器的原理及特性若能先有个大致的了解，对我们学习和了解STM32定时器的各个具体功能**非常有必要！**如果只是看到些零散的东西而看不到各个概念、功能块的相互关联以及用途时，很容易让人中途举手投降而半途而废。

- 3、下面从不同角度来对STM32定时器来做些框架性介绍，最后达到对STM32定时器的各个概念、功能及各个信号或模块间的相互关联，有个整体性的认识和了解。

- **注：**下面整体介绍的内容，可能不够精确，主要为了在我们脑海里建立起一个针对STM32定时器的大致**模块结构、功能框架**，对关键**概念元素**建立初步印象。

从功能模块大体了解STM32定时器

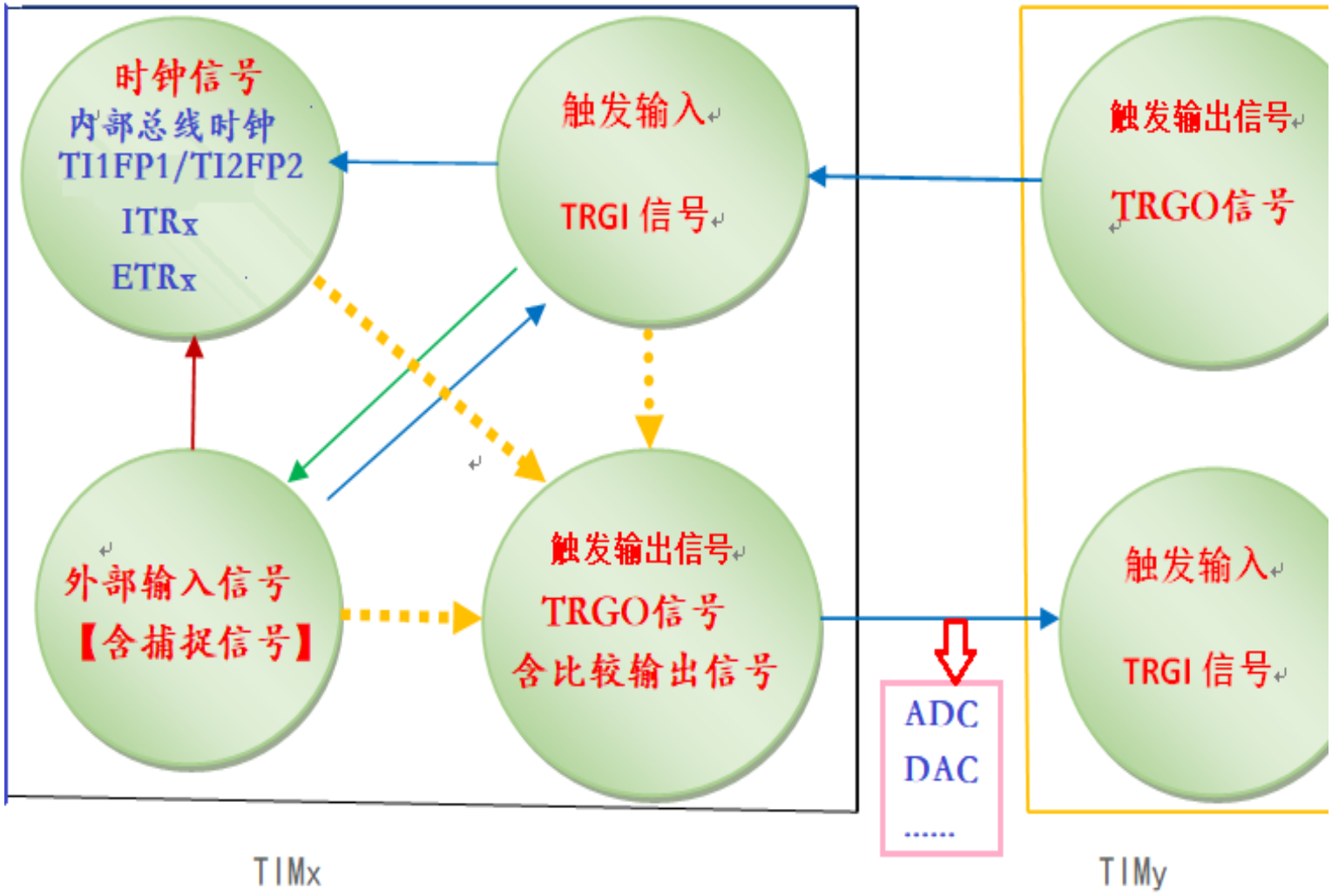
从功能模块大体了解STM32定时器

- 1、这里把stm32【通用或高级】定时器大致分为六个功能单元：
- 从模式控制单元：负责时钟源、触发信号源的选择；控制计数器的启停、复位、门控等；
- 时基单元：定时器核心单元。负责时钟源的分频、计数、溢出重装等。
- 输入单元：为部分的**时钟信号**、**捕捉信号**、**触发信号**提供信号源。
- 比较输出单元：通过对比较寄存器与计数器的数值匹配比较，实现不同输出波形。
- 触发输出单元：输出触发信号给到其它定时器或外设。
- 捕捉比较单元： 是输入捕捉或比较输出的公共执行单元。

从寄存器特色了解STM32定时器

- 定时器中的PSC/ARR/RCR/CCR寄存器具有预装载功能，即每类寄存器具有双寄存器机制,分别由各自的影子寄存器和预装载寄存器组成；
- 影子寄存器是真正起作用的寄存器，预装载寄存器为影子寄存器提供缓冲，提前做数据或指令准备；因为定时器工作往往具有一定周期性，如果每次我们的参数修改都直接作用于实际寄存器，往往不可避免会影响到当前周期的正常计数以及相关的输出动作。
- 用户操作的永远只是 预装载寄存器！包括DMA的访问。
- ARR/CCR影子寄存器的预装功能可软件开启或关闭。在开启预装载功能时，影子寄存器的内容必须借助更新事件完成更新！
- 关于影子寄存器的预装载功能的开启或关闭，往往也犹如影子、如幽灵般影响到我们的定时器应用开发。充分理解预装载机制与更新事件的功能很重要！对于这个概念，我们做定时器开发时，要力争心中有数。

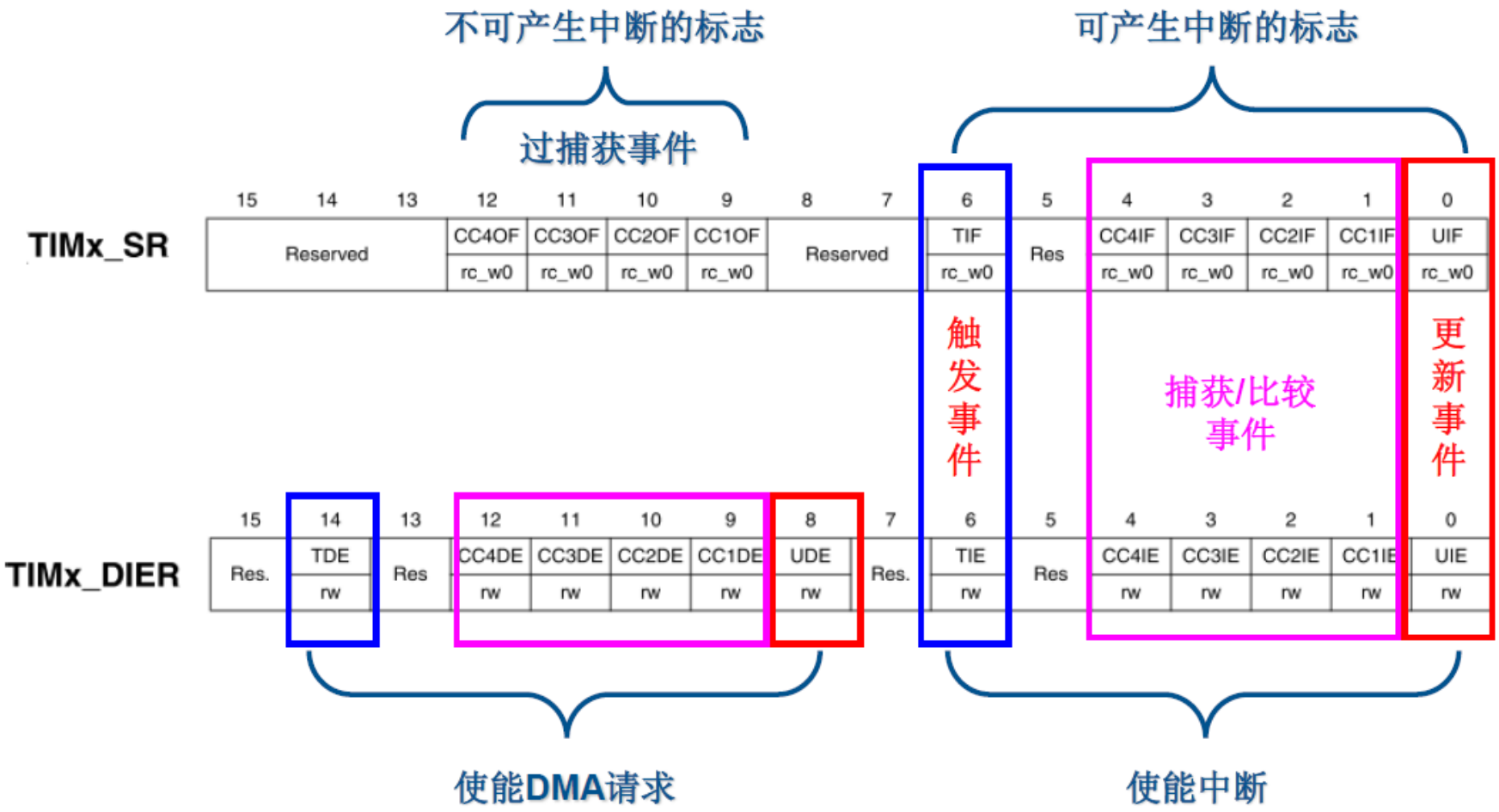
从信号链角度大体了解STM32定时器



从信号链角度大体了解STM32定时器

- STM32定时器中存在着几种基本的信号，输入信号、时钟信号、触发输入信号、触发输出信号，它们之间相互关联形成相应的信号链，从而衍生出各种定时器的功能。
- 弄清这几类信号的来龙去脉、以及相互关联后所产生的功能，对我们整体把握STM32定时器功能框架非常有帮助。
- 触发输入信号的出现，就产生了定时器的从模式特性，基于定时器之间的触发输出与触发输入连接，衍生出定时器之间的触发与同步。
- 有些信号的多角色，比方某些来自定时器输入通道TI1/TI2的信号，有时可能只是作为输入捕捉信号；有时可能只是作为个单纯的触发信号；有时可能兼做触发信号和捕捉信号；有时可能兼做时钟信号与触发信号等，让信号链变得错综复杂起来，相应的也让定时器功能随之变得灵活多变了。其中，关于时钟信号与触发信号以及二者的交叉关系的理解，是个难过的关，过了这个关，您定会有豁然开朗的feeling!

从定时器相关事件整体了解STM32定时器



从定时器相关事件整体了解STM32定时器

- 更新事件：比如影子寄存器更新往往需借助该事件；
- 触发事件：定时器收到各类触发输入信号时往往激发该事件；
- 捕获、比较事件： 发生输入捕捉或比较输出时会产生该事件；
- 上面几类事件都可触发中断或DMA请求；
- 要想充分发挥STM32定时器的功能，除了解其基本原理外，还得善用各类定时器事件以及中断、DMA功能。
- 其中更新事件的产生及功能，须重点了解和掌握。

从比较输出功能方面整体了解STM32定时器

毋庸置疑，STM32定时器的强大功能很大程度上通过其灵活的比较输出功能来体现。基本的比较输出模式：

- PWM输出：PWM1模式或PWM2模式
- 比较匹配输出模式：比较匹配时输出有效或无效电平
- 强制输出模式：　强制输出有效或无效
- 比较输出切换模式：功能极为灵活和强大。
- 基本的比较输出功能 结合 定时器事件、主从触发模式、中断、DMA等，使得STM32定时器的输出功能十分灵活多样。
- 后面会有各类相关案例分享。

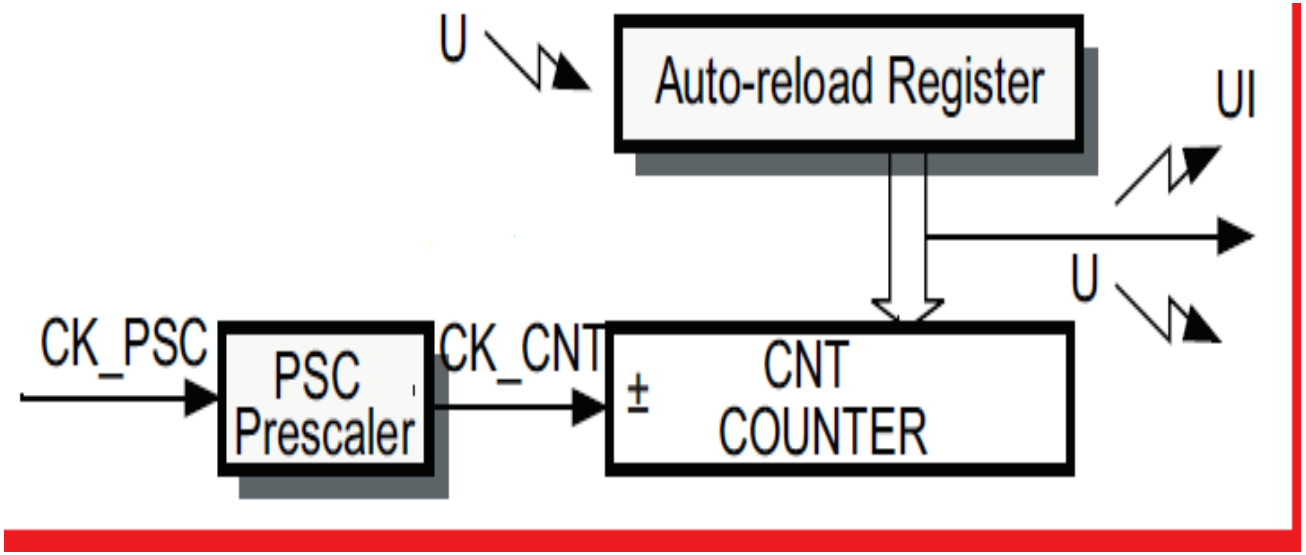
整体把握STM32定时器功能之小结

- 1、六类功能单元
- 【时基、从模式控制、输入、输出、捕捉比较、触发输出】
- 2、四类信号
- 【时钟信号、外部输入信号、触发输入信号、触发输出信号】
- 3、四类事件
- 【更新事件、捕捉、比较事件、触发事件】
- 4、一大特性
- 【影子寄存器的预装载特性】

一、时基单元

- 内容要点：
  - 时基单元是计数器的核心部分
  - 我们要弄清基本组成，各自功能是什么？
  - 涉及到哪些寄存器，哪些寄存器只是高级定时器才有的
  - 弄清影子寄存器的预装特性，是**重点**！
  - 计数器的自动重装是个**难点**，不容易理解透彻

时基单元框图



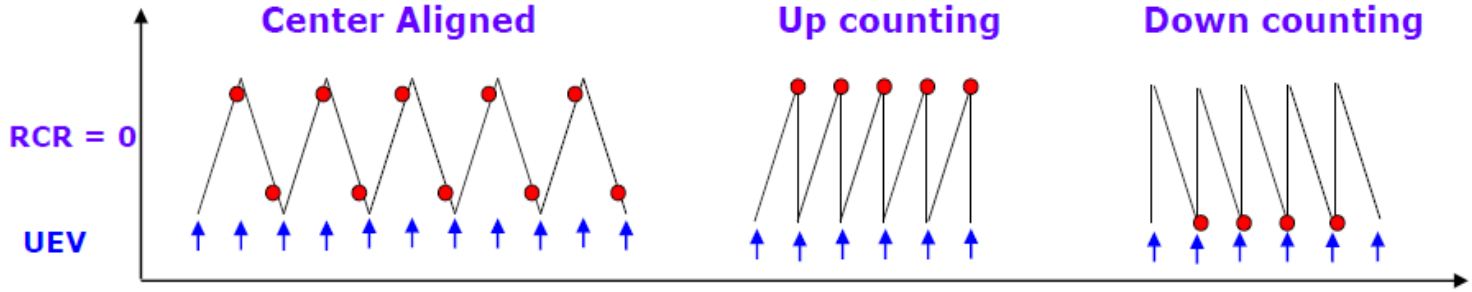
- 基本定时器、通用定时器的**时基单元由3个重要模块组成：**
- **分频器、【Prescaler Counter+ Prescaler Register】**
- **核心计数器、 [Counter + Counter Register]**
- **自动重装载器、[Auto Reloader + Auto Reloader Register]**

常规定时器时基单元寄存器介绍

- TIMx_PSC分频寄存器　　设置分频器对时钟源的分频比或分频系数
- TIMx_CNT 核心计数器　　对从分频器过来的时钟进行计数
- TIMx_ARR自动重装寄存器　为计数器设置计数边界或初始值，决定计数脉冲的多少或计时周期长短。比如计数器向上计数时，记到多少发生溢出；向下计数时从多少开始往下计数。**它的具体含义及使用需要配合计数器的计数模式予以确定。（*）**
- TIMx_ARR/ TIMx_PSC 都分别由**影子寄存器**和**预装寄存器**组成。
- **影子寄存器：乃定时器运行中真正起作用的寄存器，即实际寄存器。**
- 预装寄存器：乃用户操作的寄存器，常用来提前为实际影子寄存器做数据或指令准备。发生更新事件时，预装寄存器的数据拷贝到影子寄存器而发挥作用。

计数器之计数模式

		CMS[1:0]	DIR	UEV的产生
向上计数 Upcounting		00	0	从0开始自加计数直到ARR，产生UEV； 又从0开始循环……
向下计数 Downcouting		00	1	从ARR开始自减计数直到0，产生UEV； 又从ARR开始循环……
中间对齐计数 Center-aligned	模式1	01	硬件置位，软件只读	从0开始自加计数直到ARR-1，产生UEV； 接着从ARR开始自减计数到1，产生UEV； 又从0开始循环……
	模式2	10		
	模式3	11		

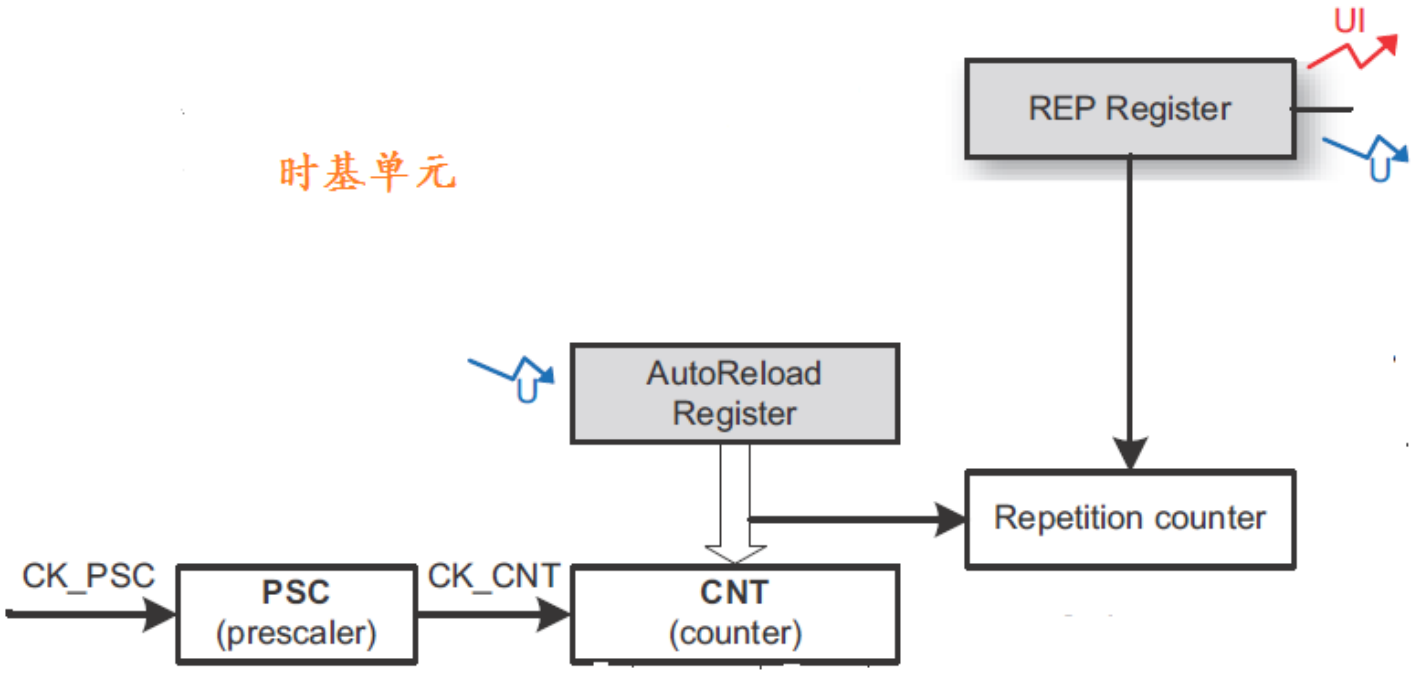


计数器溢出与重装

- 不同计数模式下的溢出与重装操作。不理解下面细节，在做比较输出时有些波形可能就看不明白。【提醒：看清溢出点与重装值】

描述 / 计数模式	溢出点	CNT 重装值	计数示意图
向上计数模式	CNT==ARR	CNT=0	
向下计数模式	CNT== 0	CNT=ARR	
中心对齐模式	CNT==ARR-1	CNT=ARR	
	CNT==1	CNT=0	

STM32高级定时器时基单元



- 相比通用定时器，高级定时器时基单元增加了 重复计数器 和对应的寄存器TIMx_RCR，重复计数器是个向下计数器
- 当计数器发生 TIMx_RCR+1次溢出动作后 会触发 更新事件

影子寄存器和预装寄存器【1】

- STM32定时器中四个带影子特性的寄存器组【**影子+预装**】：
- ***TIMx_PSC*** 分频寄存器　　***TIMx_ARR*** 自动重装载寄存器
- ***TIMx_CCR***捕捉比较寄存器　　***TIMx_RCR*** 重复计数寄存器
- 其中**ARR、CCR**寄存器带预装载使能控制位，PSC、RCR无预装使能控制位，所以对于PSC/RCR实际寄存器的数据更新只能通过更新事件实现从预装寄存器数据到影子寄存器的拷贝更新。

影子寄存器和预装寄存器【2】

- TIMx_ARR　　带预装载使能控制位 ARPE@TIMx_CR1
- TIMx_CCR　　带预装载使能控制位 OCxPE@TIMx_CCMR

当关闭预装载使能位时，用户修改预装寄存器的数据后会立即被拷贝进影子寄存器【实际寄存器】，否则，修改过的预装寄存器的数据只能等到下次更新事件 来完成从预装寄存器数据到影子寄存器的拷贝更新。

TIMx_CCR捕捉寄存器：【做输入捕捉、比较输出时会用到】

更新操作与更新事件【1】

- 这里 更新操作不等于 更新事件！
- 更新操作是一种动作，是更新事件的源头，即事件源；
- 更新事件是基于更新操作所导致的后续影响或结果。
- 可能的更新操作【事件源】有3类：
  - 核心计数器的溢出【上溢或下溢】
  - 软件复位操作【对TIMX_EGR@UG】 （*）
  - 工作在复位模式下的定时器收到触发信号【即复位触发信号】

（*）关于软件复位操作【对TIMX_EGR@UG】在我们定时器的初始化代码中总是被用到。因为PSC/RCR寄存器的预装载功能一直打开的，他们的更新必须借助更新事件。

（*）关于软件复位操作【对TIMX_EGR@UG】，后面在谈到主模式定时器的TRGO输出时也会提到。

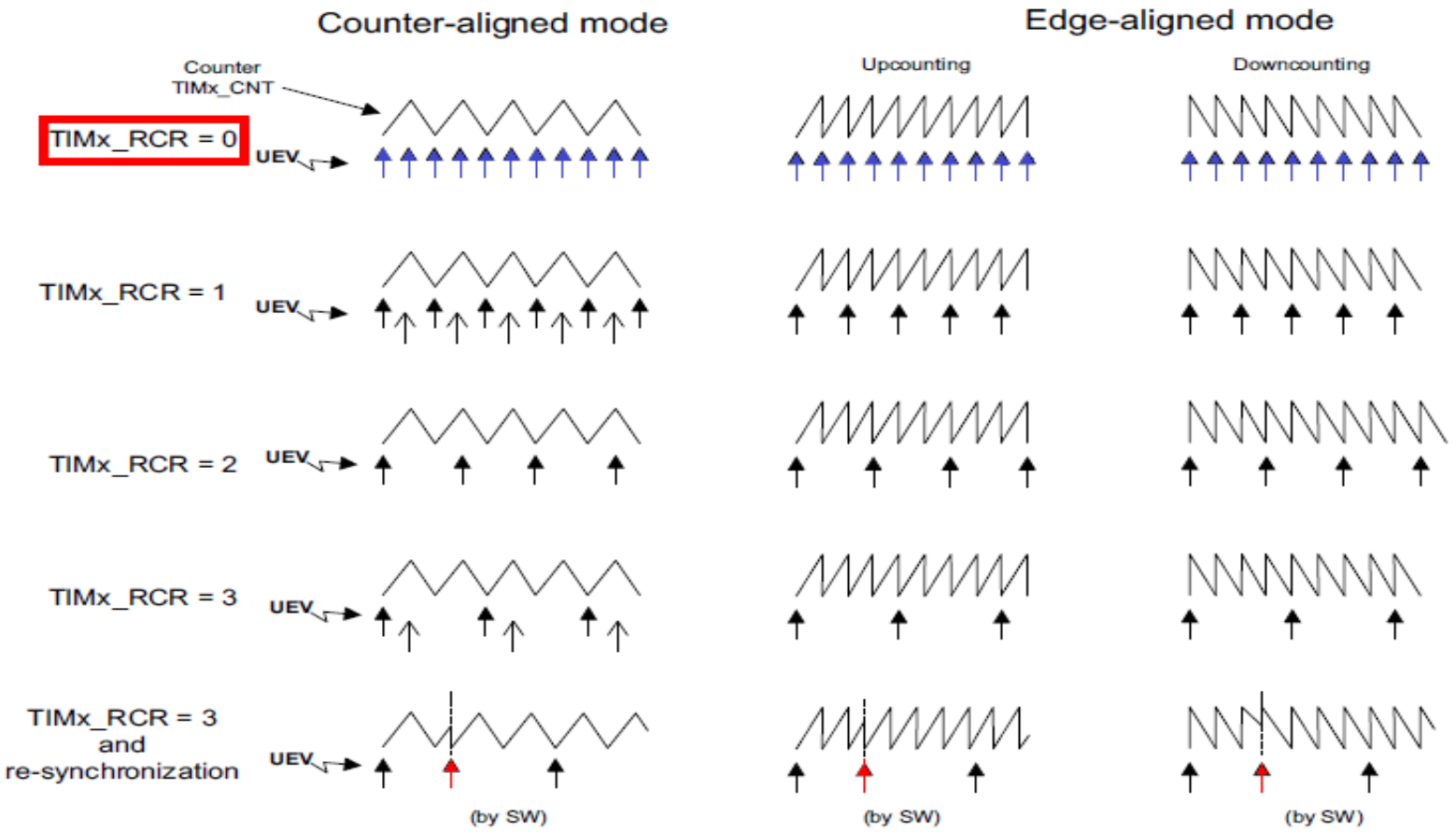
更新与更新事件【2】

- 控制寄存器的UDIS@TIMx_CR1位决定更新操作能否升级为事件、
- 其中，对于高级定时器必须每溢出**RCR+1**次时才能产生更新事件。
- 如果更新操作升级为更新事件
  - 实现从预装载寄存器的数据到影子寄存器的内容拷贝，即完成影子寄存器 的内容更新
  - 实现计数器【预分频计数器、核心计数器、重复计数器】的重装或重新初始化
  - 置位状态寄存器的UIF@TIMx_SR，并可以触发更新中断或触发DMA请求
- 如果更新操作不能升级为更新事件
  - 仅限于计数器的重装或重新初始化，不做影子寄存器的更新；
  - 无更新标志的置位，不触发中断或DMA请求

更新与更新事件【3】

- 发生更新操作时计数器的重新初始化
  - 分频计数器归0,然后重新计数
  - 重复计数器重装为RCR寄存器里的值；然后重新递减计数
  - 核心计数器与计数模式有关，如果是向上计数或中心对齐计数模式，CNT归0；如果是向下计数模式，CNT重装为ARR，然后重新计数
- 发生更新事件时
  - **影子寄存器**【ARR/CCR....】**的更新**操作在前，**计数器的重装**操作在后！！！
  - 因为这样可以保障了计数器的重装值使用更新过的数据。
  - 这个细节要特别注意！！！

RCR计数器的溢出与更新事件



对于通用或基本定时器，他们没有RCR计数器及寄存器，我们可以把二者看成RCR值始终等于0的情形加以统一理解和记忆，即每溢出一次就可以产生更新事件。【注意红色箭头所指的动作】

ARR、PSC、RCR寄存器数据的拟定

1、对于单向计数模式【向上或向下】：

一个计数周期为：ARR+1；实际分频比 = PSC +1

例：希望计数器每计数100个脉冲后重装并重新计数，则ARR=100-1;

希望计数器的计数脉冲是时钟源经过3分频后的脉冲，则PSC=3-1;

RCR寄存器： 16 位重复计数器，部分定时器具有。计数器每溢出RCR+1次就可以产生更新事件。此时更新事件的频率则为：

F_Update_event = TIM_CLK/((PSC + 1)*(ARR + 1)*(RCR + 1))

2、对于中央对齐计数模式，1个完整的计数周期为 ARR*2

案例1：定时器一使能就进中断的问题

- 问题描述：TIMER初始化阶段，**经常**有人反馈一使能更新中断就就进中断服务程序？给开发带来些困扰，原因可能是什么？如何解决？

- 因为在定时器的初始化代码里有软件更新操作触发了更新事件，并置位了更新中断标志，当使能更新中断时就立刻进入更新中断服务程序。

- 在STM32标准库里的TIM_TimeBaseInit( )函数里都有这句代码：
  TIMx->EGR = TIM_PSCReloadMode_Immediate;

- 在Cube库里的HAL_TIM_Base_Init( )函数里的 函数有这句代码：
  TIMx->EGR = TIM_EGR_UG;

- 显然，这两行代码使用到了前面提到的软件更新操作，触发了更新事件，置位更新中断标志。所以我们在使能定时器更新中断之前，可以先做更新中断标志的清除操作。

案例2:与预装载有关的定时输出异常

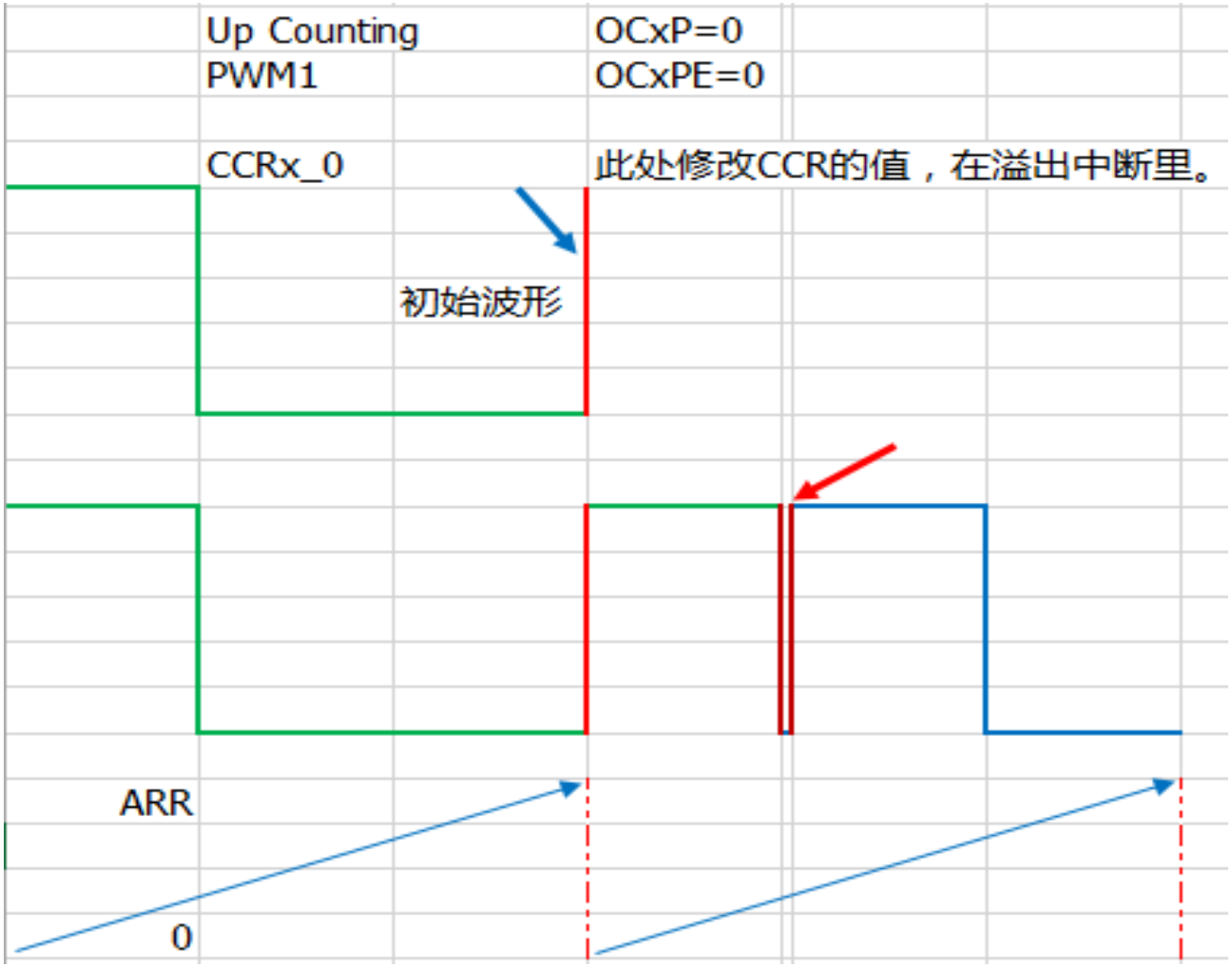
- 异常情形：配置STM32芯片TIMER1，PWM模式1，向上计数模式，周期10k，占空比每次在定时器上溢中断里调节。当打开定时器2使能，并且打开定时器2的中断时会出现输出异常的现象：

- 有一定几率出现尖窄脉冲现象。在正常占空比之后出现一个极短的小脉冲，大概200~400ns。测试中，CCR的比较值读出正常。关闭定时器2使能，输出正常，未捕捉到小脉冲。

- 问：1. 什么情况下会出现这样的现象，可以提出一些解决办法或建议吗？　排除外围硬件的问题，因为PWM输出脚的负载已全部切断，供电电源5V，直接给单片机供电，电源很稳定。

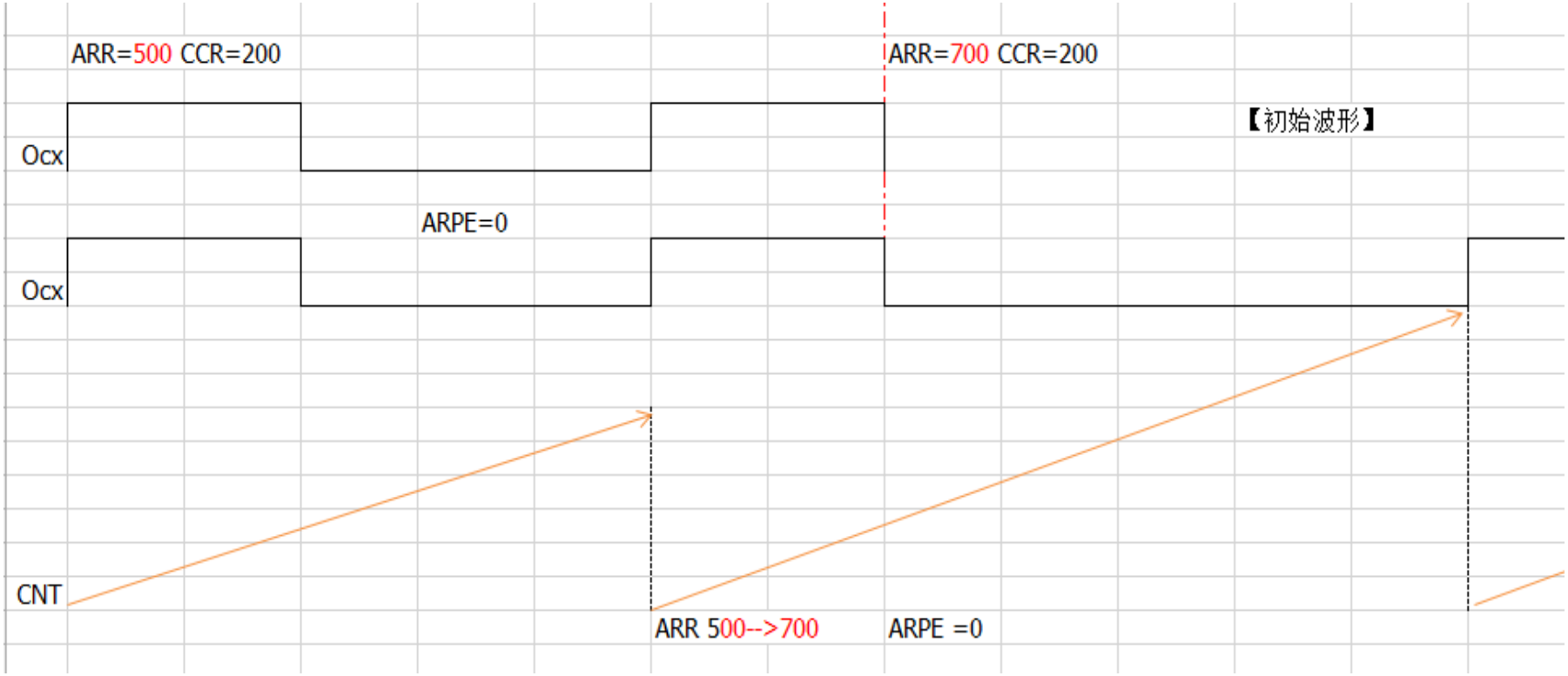
- 原因：经查看代码和分析验证，用户代码没有开启CCR的预装载功能，以及中断优先级安排方面有问题。

案例2:与预装载有关的定时输出异常【续】



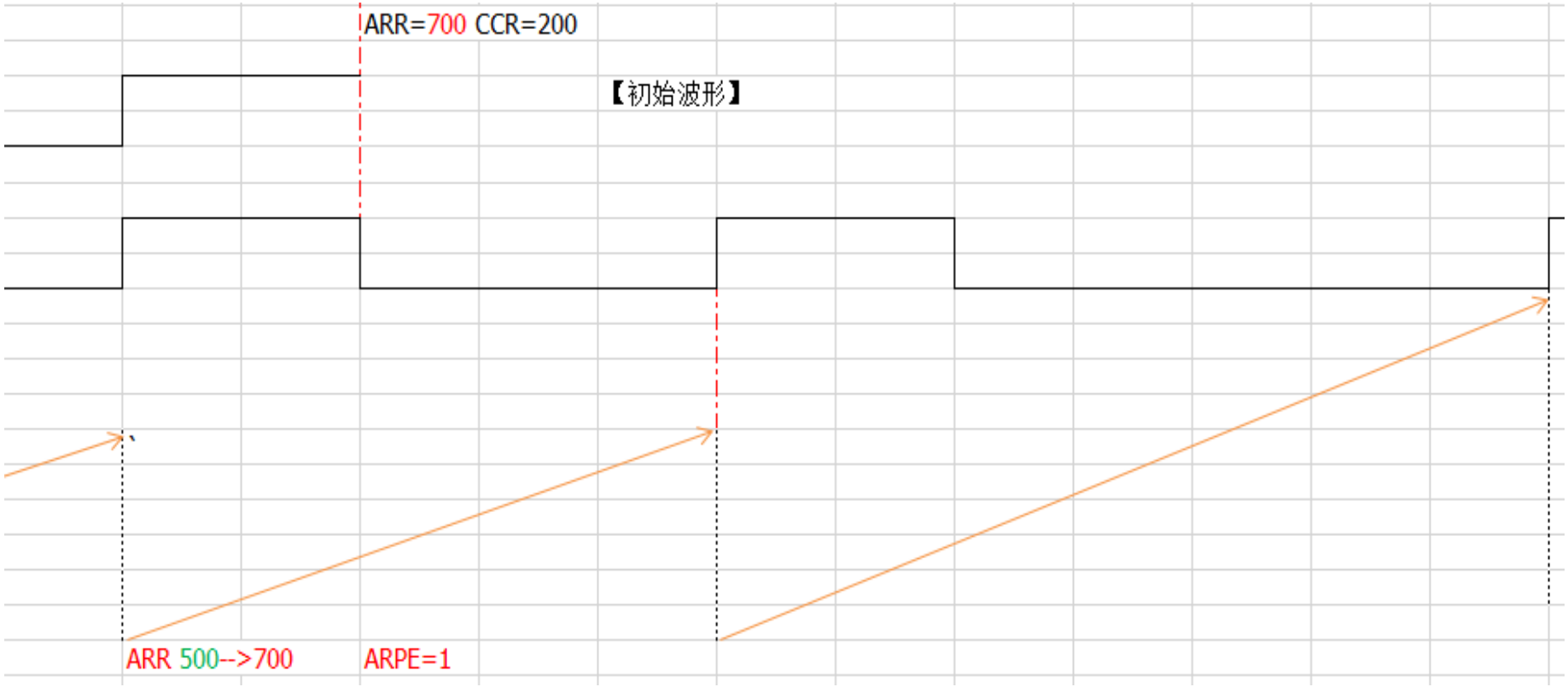
与预装载有关的案例分享【1】

- UP counting + pwm1； 极性选择：高有效 ； 更新事件不关闭



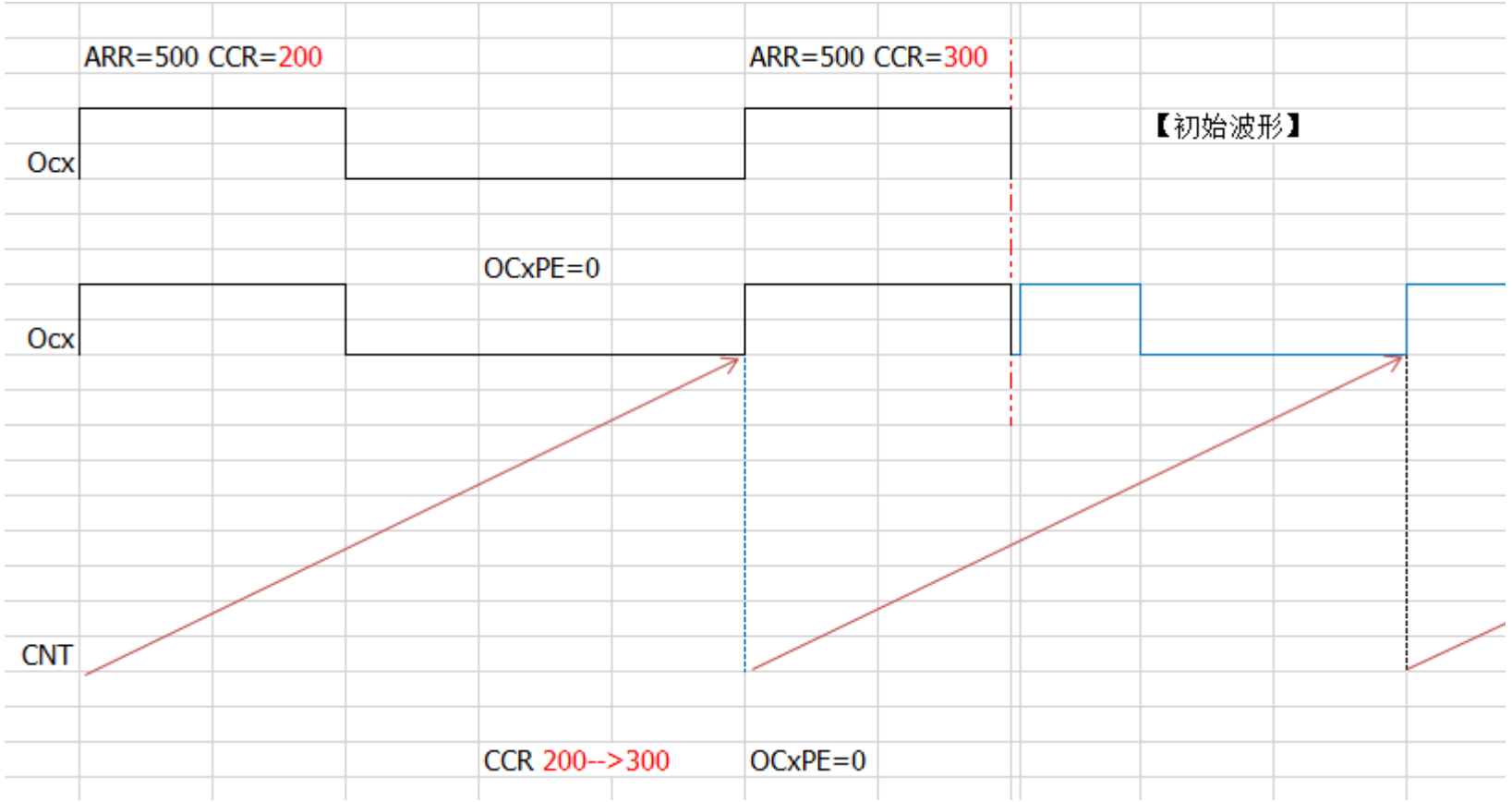
与预装载有关的案例分享【2】

- UP counting + pwm1； 极性选择：高有效 ； 更新事件不关闭



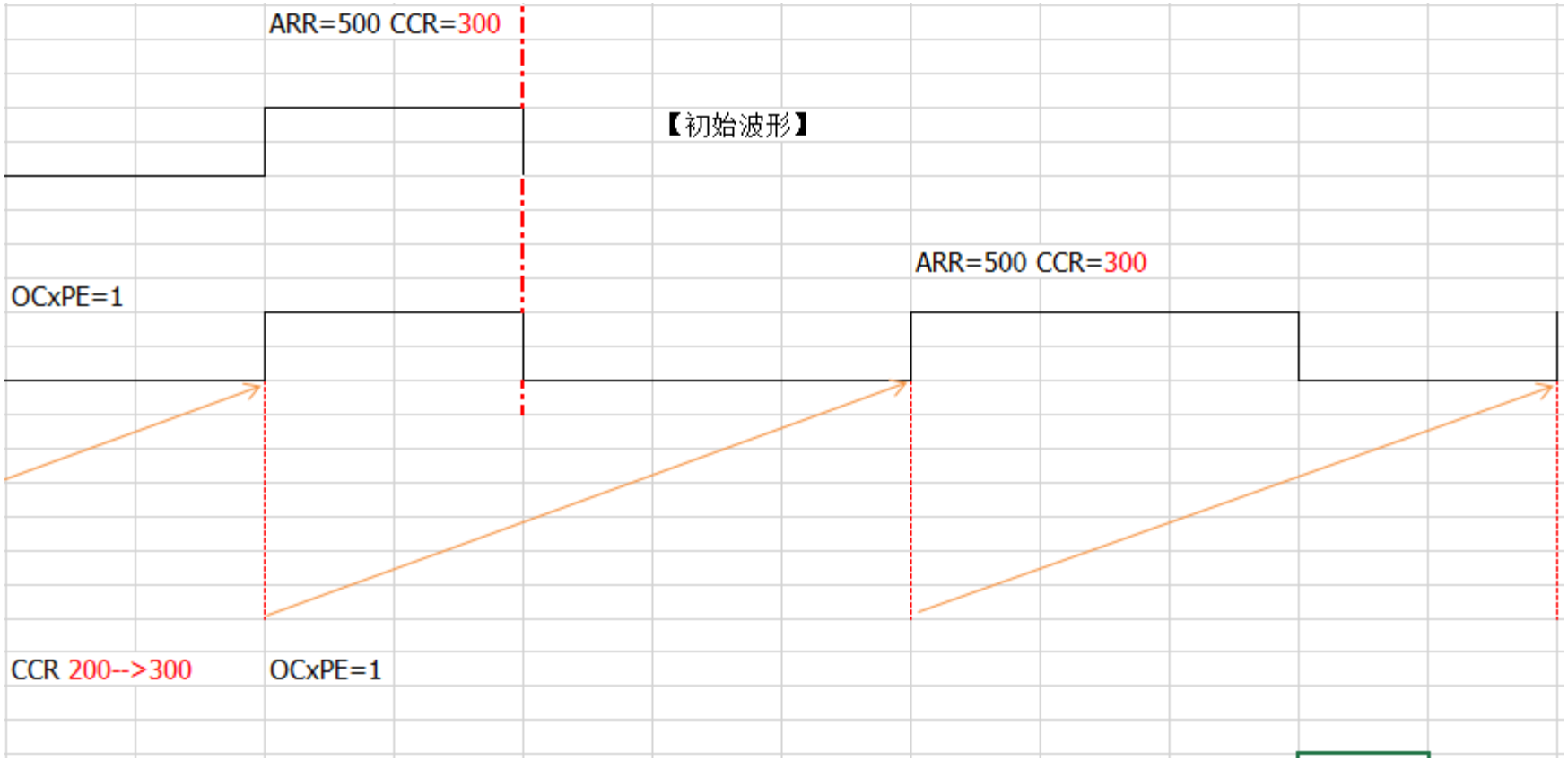
与预装载有关的案例分享【3】

- UP counting + pwm1； 极性选择：高有效 ； 更新事件不关闭



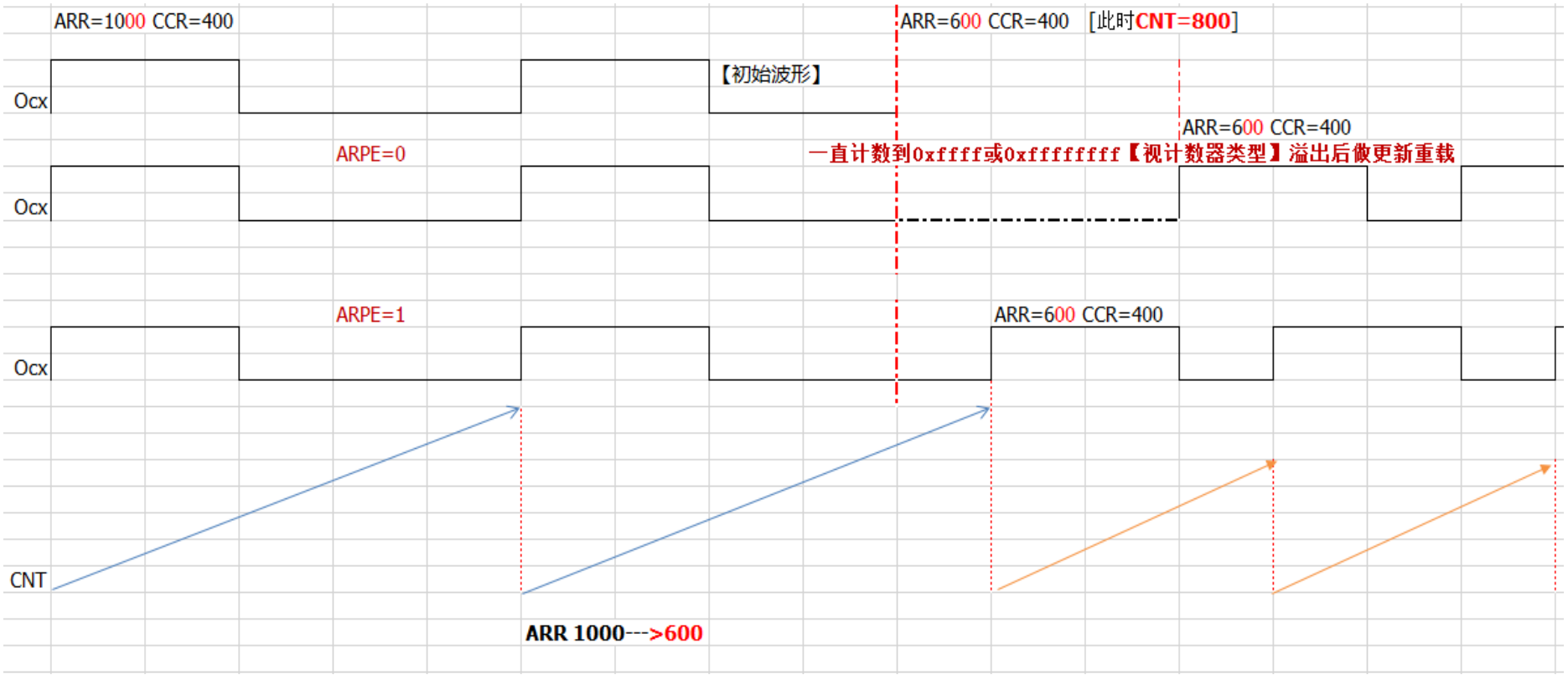
与预装载有关的案例分享【4】

- UP counting + pwm1； 极性选择：高有效 ； 更新事件不关闭



与预装载有关的案例分享【5】

- UP counting + pwm1； 极性选择：高有效 ； 更新事件不关闭



时基单元的小结与思考

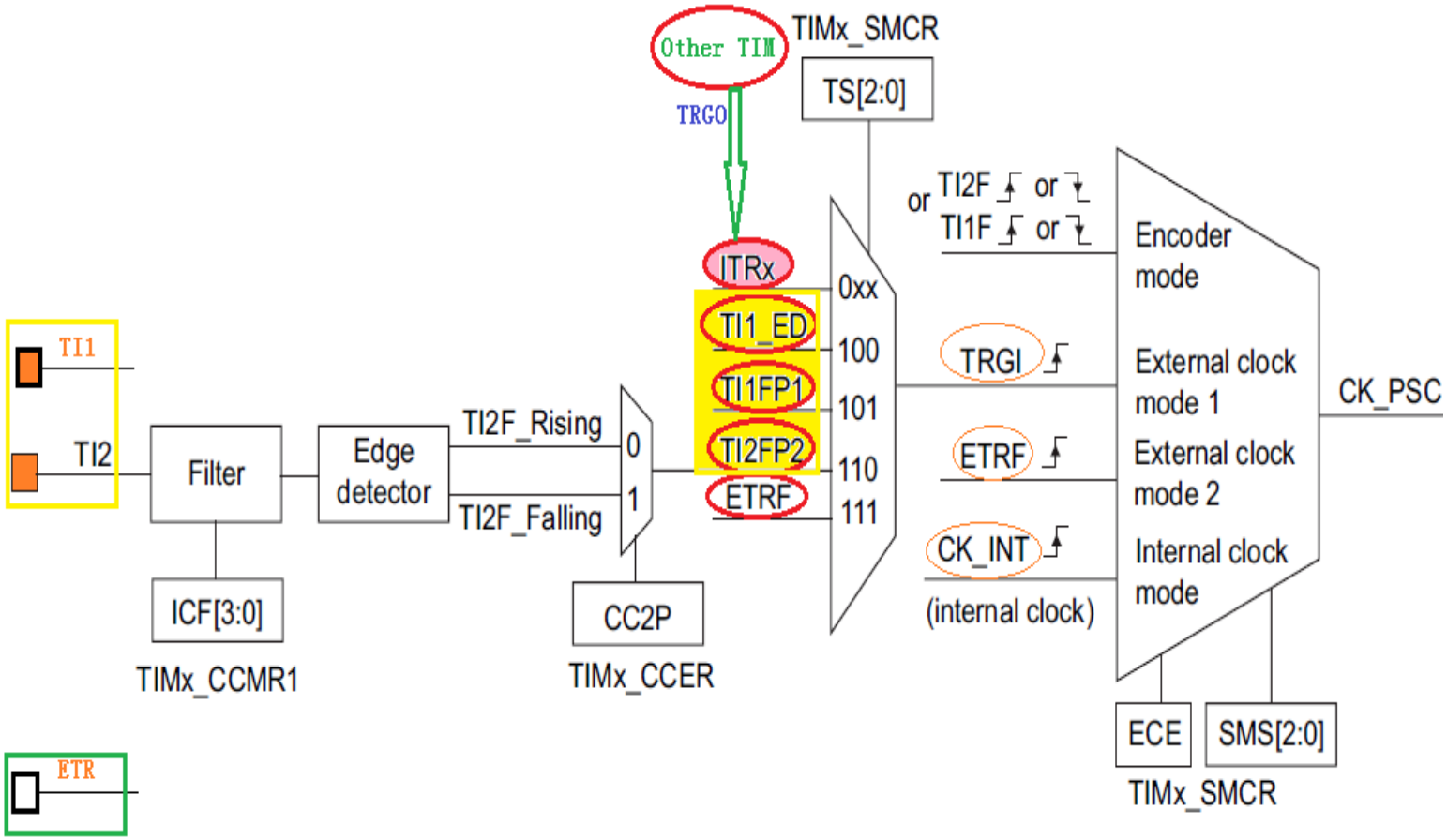
- 1、时基单元主要涉及哪几个重要寄存器？
- 2、CNT计数器的溢出与更新事件发生是否完全一致？
- 3、ARR能否为0？为0会怎么样？
- 4、我们说计数器重置指的是什么？
- 5、预装载寄存器的数据什么时候拷贝进影子寄存器？

二、计数器的时钟源

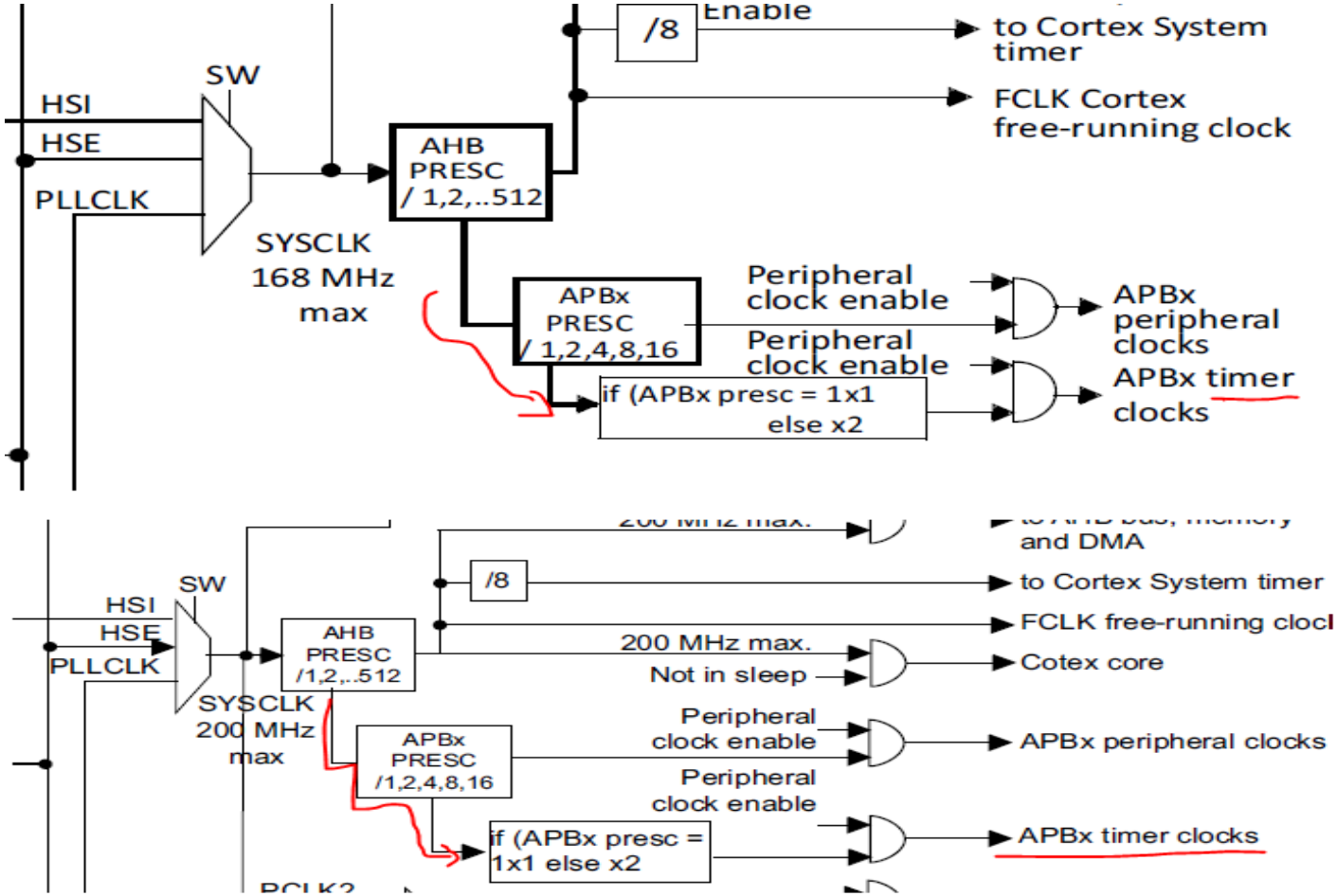
- 三个来源；内部时钟源、ETR脚、触发信号
- 1、内部时钟源， 来自芯片时钟系统给到定时器的时钟
- 2、来自于芯片ETR脚的外来时钟
- 3、来自于各类触发输入信号作为时钟
- 温馨提示：从定时器的时钟源开始，基本就进入了定时器概念、术语多发、频现区，最大的特色是各类跨功能概念开始不停交织在一起，其中有些概念又往往不是你正在阅读章节的内容。先保持冷静，耐心点。看到后面就会慢慢领会。此时脑子里不妨回想下开篇的整体介绍，坚持坚持，总会有拨云见日的时候。

二、计数器的时钟源

- 三个来源；内部时钟源、ETR脚、触发信号



内部时钟TIMx_CLK



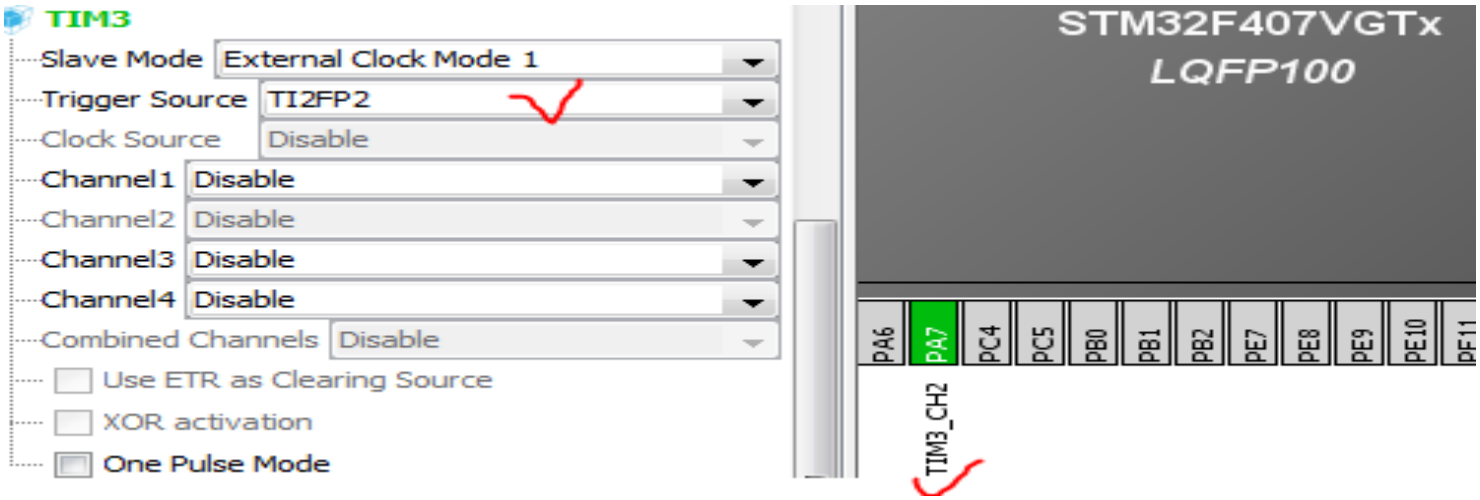
内部触发输入ITRx作为时钟源

- 下图中蓝色框住为从模式的定时器以及内部触发输入通道ITRx
- 橙色方框内是各个处于主模式的定时器的触发输出信号
- 主定时器的触发输出连接到从定时器内部的触发输入并作为时钟源

TIMx internal trigger connection

Slave TIM	ITR0 (TS = 000)	ITR1 (TS = 001)	ITR2 (TS = 010)	ITR3 (TS = 011)
TIM2	TIM1_TRGO	TIM8_TRGO	TIM3_TRGO	TIM4_TRGO
TIM3	TIM1_TRGO	TIM2_TRGO	TIM5_TRGO	TIM4_TRGO
TIM4	TIM1_TRGO	TIM2_TRGO	TIM3_TRGO	TIM8_TRGO
TIM5	TIM2_TRGO	TIM3_TRGO	TIM4_TRGO	TIM8_TRGO

外部时钟源模式1---TI2上升沿示例【1】



Counter Settings	
Prescaler (PSC - 16 bits value)	0
Counter Mode	Up
Counter Period (AutoReload Register - 16 bits value)	888
Internal Clock Division (CKD)	No Division
Slave Mode Controller	ETR mode 1
Trigger Output (TRGO) Parameters	
Master/Slave Mode (MSM bit)	Disable (Trigger input effect not delayed)
Trigger Event Selection	Reset (UG bit from TIMx_EGR)
Trigger	
Trigger Polarity	Rising Edge
Trigger Filter (4 bits value)	0

外部时钟源模式1---TI2上升沿示例【2】

1、方框内是时基参数基本配置【PSC/ARR/Counting Mode…】

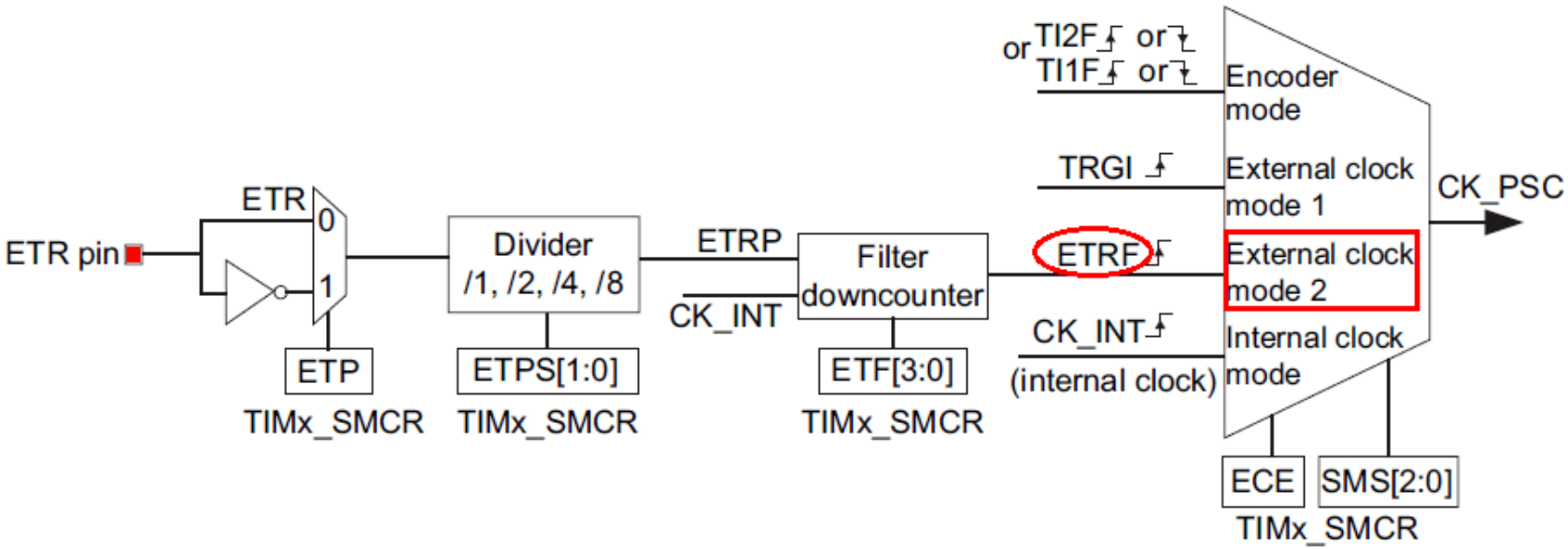
2、箭头所指是关于定时器模式、触发信号源、触发极性的选择

```
htim3.Instance = TIM3;
htim3.Init.Prescaler = 0;
htim3.Init.CounterMode = TIM_COUNTERMODE_UP;
htim3.Init.Period = 888;
htim3.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
if (HAL_TIM_Base_Init(&htim3) != HAL_OK)
{
  _Error_Handler(__FILE__, __LINE__);
}

sSlaveConfig.SlaveMode = TIM_SLAVEMODE_EXTERNAL1;
sSlaveConfig.InputTrigger = TIM_TS_TI2FP2;
sSlaveConfig.TriggerPolarity = TIM_TRIGGERPOLARITY_RISING;
sSlaveConfig.TriggerFilter = 0;
if (HAL_TIM_SlaveConfigSynchronization(&htim3, &sSlaveConfig) != HAL_OK)
{
```

外部时钟源模式2

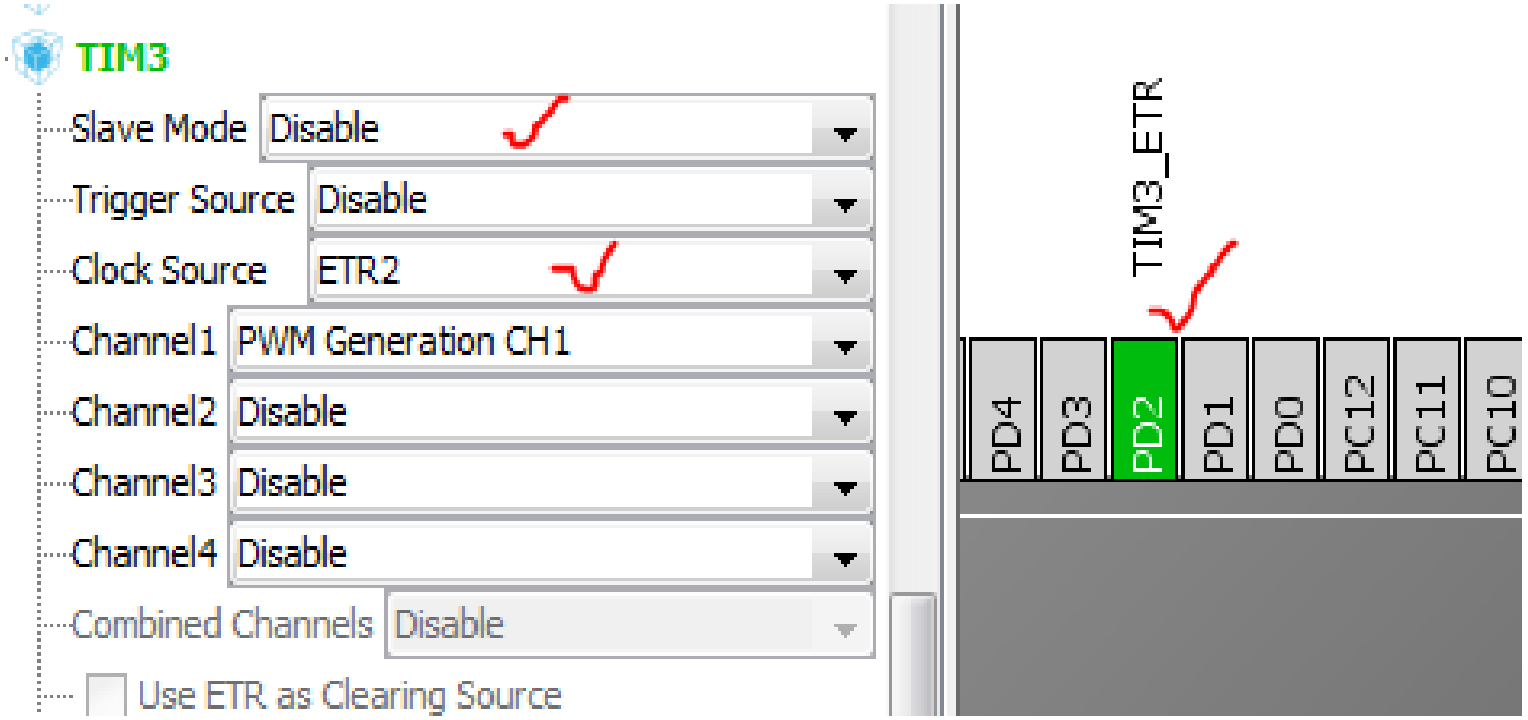
- 时钟信号源自外部ETR脚，经过极性选择、分频、滤波后的信号给到计数器作为其时钟源。【分频和滤波根据实际情况来设置】

- 此时，ETRF信号不经过从模式控制器，直接馈送给时钟控制单元。定时器无须工作在从模式。



外部时钟源模式2示例【1】

- 以来自ETR脚的时钟信号作为时钟源，上升沿触发，2分频后提供给计数器



外部时钟源模式2示例【2】

1、方框内是时基参数基本配置【PSC/ARR/Counting Mode…】

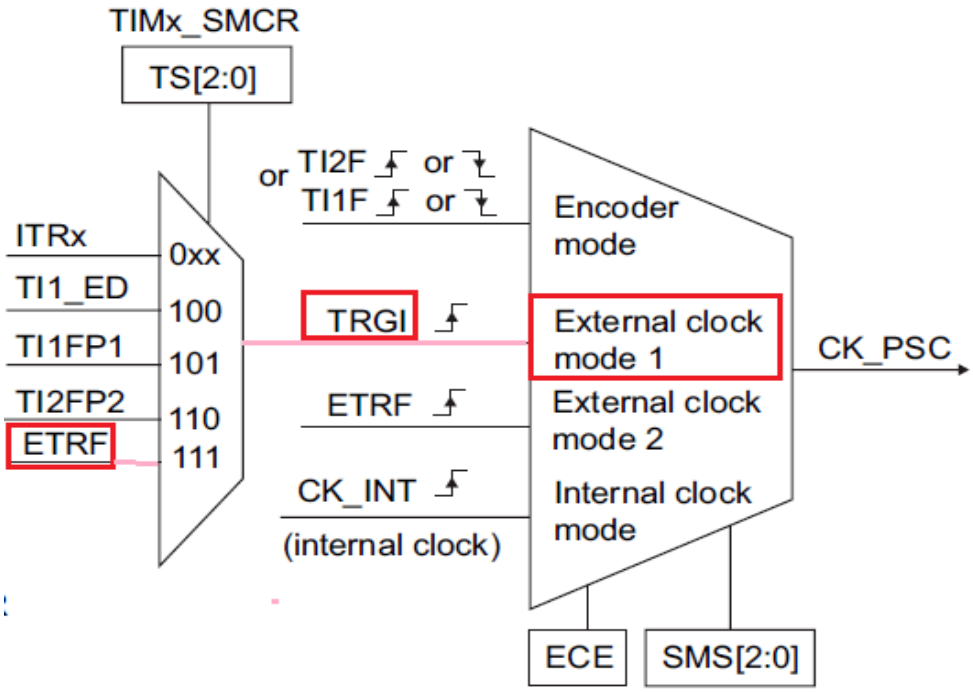
2、箭头所指是关于时钟源、时钟极性、ETR时钟的分频系数的选择

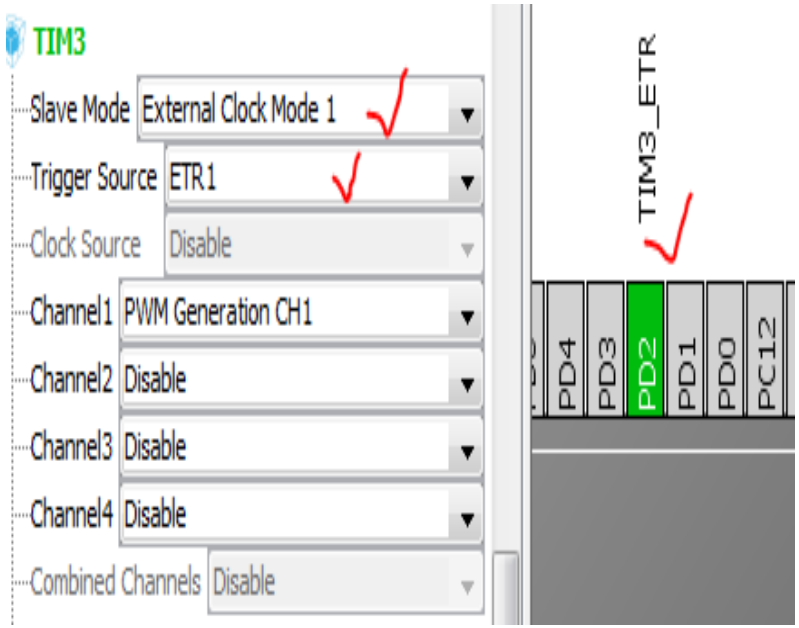
```
htim3.Instance = TIM3;
htim3.Init.Prescaler = 0;
htim3.Init.CounterMode = TIM_COUNTERMODE_UP;
htim3.Init.Period = 888;
htim3.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
if (HAL_TIM_Base_Init(&htim3) != HAL_OK)
{
  _Error_Handler(__FILE__, __LINE__);
}

sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_ETRMODE2;
sClockSourceConfig.ClockPolarity = TIM_CLOCKPOLARITY_NONINVERTED;
sClockSourceConfig.ClockPrescaler = TIM_CLOCKPRESCALER_DIV2;
sClockSourceConfig.ClockFilter = 0;
if (HAL_TIM_ConfigClockSource(&htim3, &sClockSourceConfig) != HAL_OK)
{
```

ETR时钟源工作在外部时钟从模式1

- 该话题本应该在介绍定时器从模式时介绍，但由于来自ETR脚的时钟信号既可以工作在外部时钟模式2，也可以工作在外部时钟模式1。

- 当来自ETR脚的时钟脉冲经过分频、滤波后连接从模式控制器，作为触发信号的同时并为计数器提供时钟，此时就工作在外部时钟模式1.





ETR时钟源工作在外部时钟从模式1

- 红框内对时基参数进行配置
- 箭头所指就从模式、触发信号源、信号分频等进行配置

```
htim3.Instance = TIM3;
htim3.Init.Prescaler = 0;
htim3.Init.CounterMode = TIM_COUNTERMODE_UP;
htim3.Init.Period = 888;
htim3.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
if (HAL_TIM_Base_Init(&htim3) != HAL_OK)
{
  _Error_Handler(__FILE__, __LINE__);
}

if (HAL_TIM_PWM_Init(&htim3) != HAL_OK)
{
  _Error_Handler(__FILE__, __LINE__);
}

sSlaveConfig.SlaveMode = TIM_SLAVEMODE_EXTERNAL1;
sSlaveConfig.InputTrigger = TIM_TS_ETRF;
sSlaveConfig.TriggerPolarity = TIM_TRIGGERPOLARITY_NONINVERTED;
sSlaveConfig.TriggerPrescaler = TIM_TRIGGERPRESCALER_DIV4;
sSlaveConfig.TriggerFilter = 0;
if (HAL_TIM_SlaveConfigSynchronization(&htim3, &sSlaveConfig) != HAL_OK)
{
```

各种时钟源的小结汇总

- 1、注意来自外部触发脚ETR的时钟信号有两种走法。一种经从模式控制器走触发信号的路径，此时定时器工作在外部时钟模式1。另一种走法类似内部时钟，不经触发信号选择器，而是直接给定时器提供时钟。

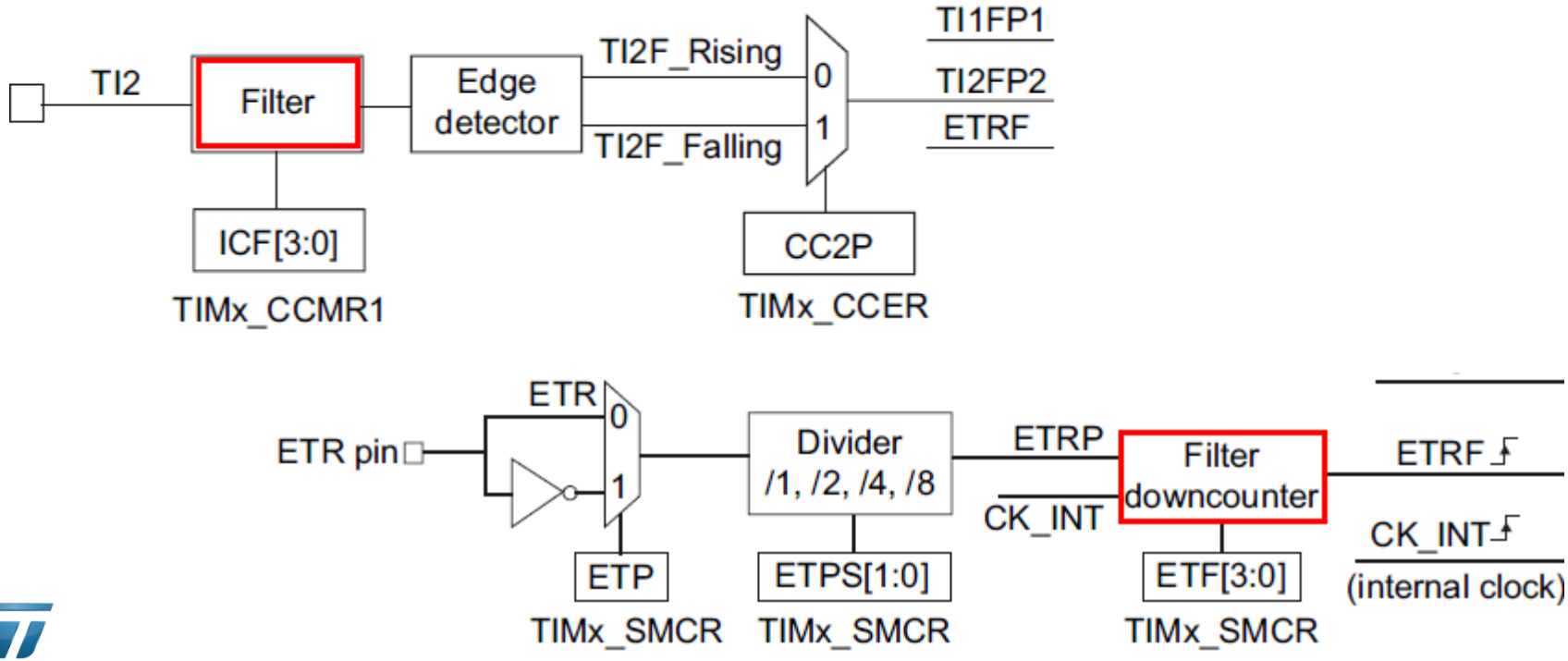
时钟源	SMS	CEN	ECE	描述	备注
内部时钟 CK_INT	000	1			视定时器所连接的APB总线时钟而定
外部时钟模式1 TIx	111	1		计数器在选定引脚的指定边沿计数	Max $f_{EXT}=f_{TIMxCLK}/2$
ETR	111	1			TS=111(ETRF)
外部时钟模式2 ETR		1	1	计数器在ETR的指定边沿计数	ETRP频率最高不能超过CK_INT的1/4，可通过ETPS分频
内部触发输入 ITRx	111	1		主计数器的输出作为从计数器的输入	

注意：外部时钟模式1和2同时使能的情况下，外部时钟信号来自ETRF

外部引脚输入信号的滤波

- 为了防止因为输入信号上的噪声或边沿抖动而导致误计数、误触发，我们可以针对外部输入信号进行合适的滤波。

- 用户要做的就是针对输入信号具体情况【频率、噪声等】配置适当的滤波参数，选择适当采样时钟和采样次数，其它事情交给定时器硬件去处理。这样也可以减少因为做滤波而导致的软件开销。



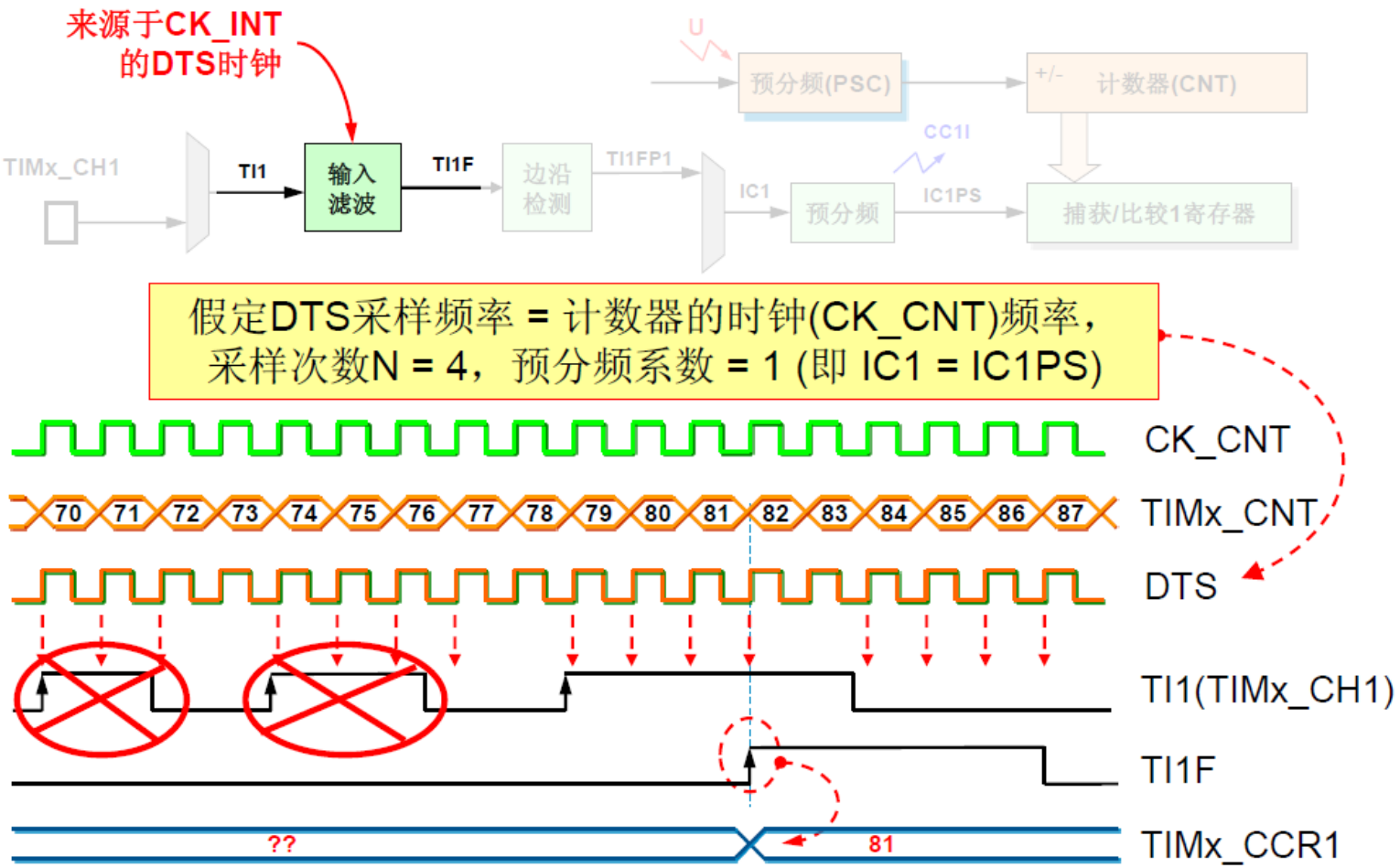
外部引脚的滤波设置

- 为了防止信号边沿的抖动，可以设置滤波参数
  - 采样时钟$f_{SAMPLING}$
    - f_{DTS} (f_{CK_INT} 1倍/2倍/四倍分频)的2倍/4倍/8倍/16倍/32倍分频
    - f_{CK_INT}
  - 滤波阶数N

0000: No filter, sampling is done at f_{DTS}	1000: $f_{SAMPLING}=f_{DTS}/8$, N=6
0001: $f_{SAMPLING}=f_{CK\_INT}$, N=2	1001: $f_{SAMPLING}=f_{DTS}/8$, N=8
0010: $f_{SAMPLING}=f_{CK\_INT}$, N=4	1010: $f_{SAMPLING}=f_{DTS}/16$, N=5
0011: $f_{SAMPLING}=f_{CK\_INT}$, N=8	1011: $f_{SAMPLING}=f_{DTS}/16$, N=6
0100: $f_{SAMPLING}=f_{DTS}/2$, N=6	1100: $f_{SAMPLING}=f_{DTS}/16$, N=8
0101: $f_{SAMPLING}=f_{DTS}/2$, N=8	1101: $f_{SAMPLING}=f_{DTS}/32$, N=5
0110: $f_{SAMPLING}=f_{DTS}/4$, N=6	1110: $f_{SAMPLING}=f_{DTS}/32$, N=6
0111: $f_{SAMPLING}=f_{DTS}/4$, N=8	1111: $f_{SAMPLING}=f_{DTS}/32$, N=8

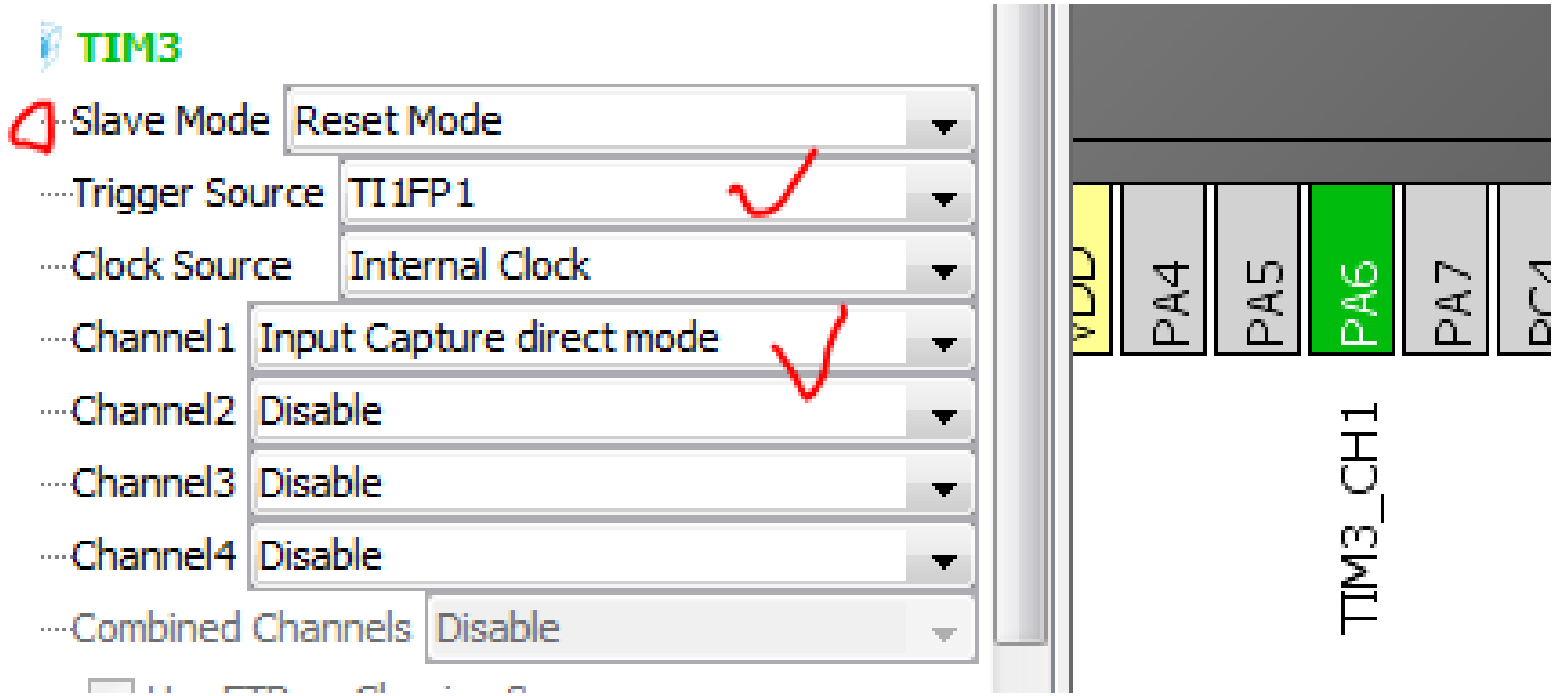
外部引脚	信号	边沿检测	分频	滤波
TIMx_ETR	ETR	ETP@SMCR	ETPS@SMCR	ETF@SMCR
TIMx_CH1	TI1	CC1P/CC1NP@CCER	IC1PSC@CCMR1	IC1F@CCMR1
TIMx_CH2	TI2	CC2P/CC2NP@CCER	IC2PSC@CCMR1	IC2F@CCMR1
TIMx_CH3	TI3	CC3P/CC3NP@CCER	IC3PSC@CCMR2	IC3F@CCMR2
TIMx_CH4	TI4	CC4P/CC4NP@CCER	IC4PSC@CCMR2	IC4F@CCMR2

输入滤波示例



基于不同时钟源应用的配置示例【1】

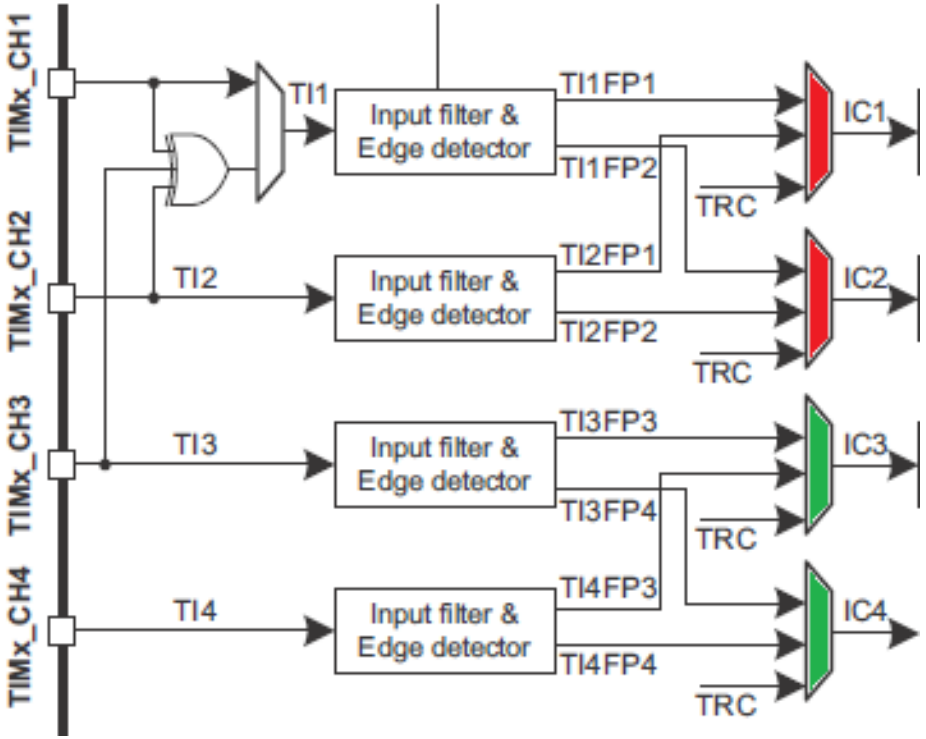
- TI1 做输入捕捉信号、TI1FP1触发信号，计数器工作在复位模式
- Int_clk内部时钟作为 时钟源



插播：信号的直接输入与间接输入

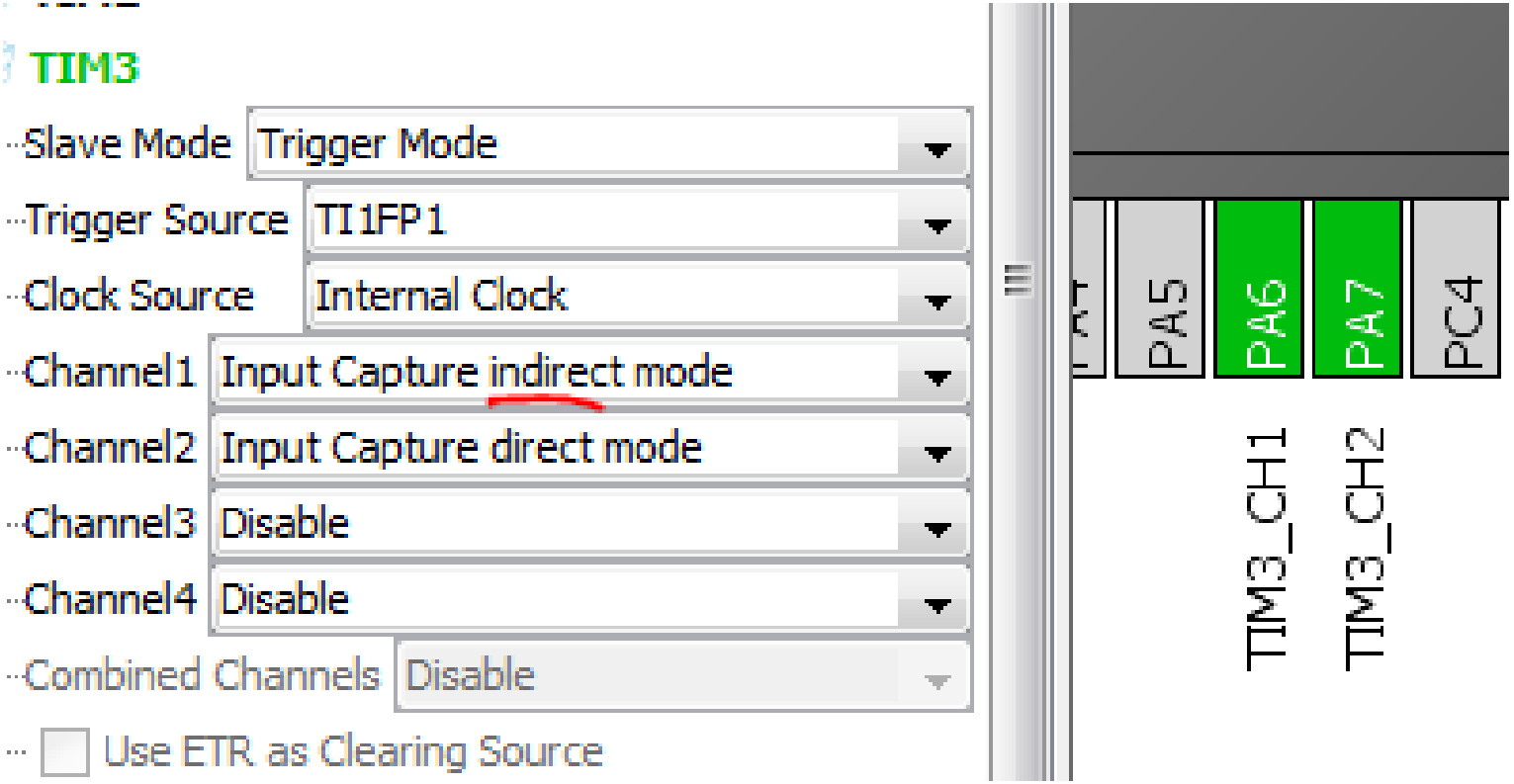
- 该说法存在于 输入通道1与通道2之间 或者通道3与通道4之间

- 比如：TI1FP2就是通道2的间接输入信号，TI3FP4就是通道4的间接输入信号。换言之，通道1的捕捉信号并非一定来自TI1，而可能来自TI2的信号。



基于不同时钟源应用的配置示例【2】

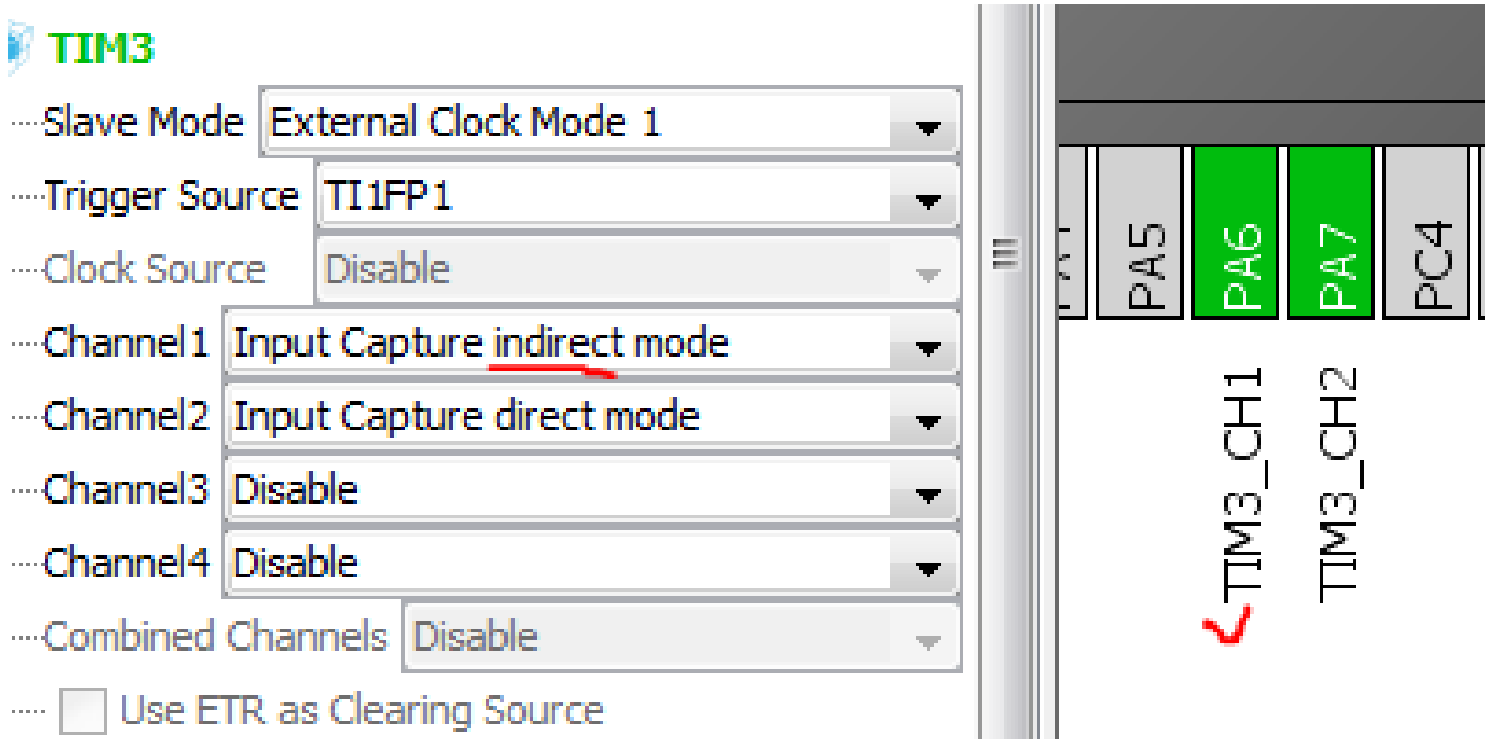
- TI2做输入捕捉信号、TI1FP1触发信号，计数器从模式不限
- 使用内部时钟源



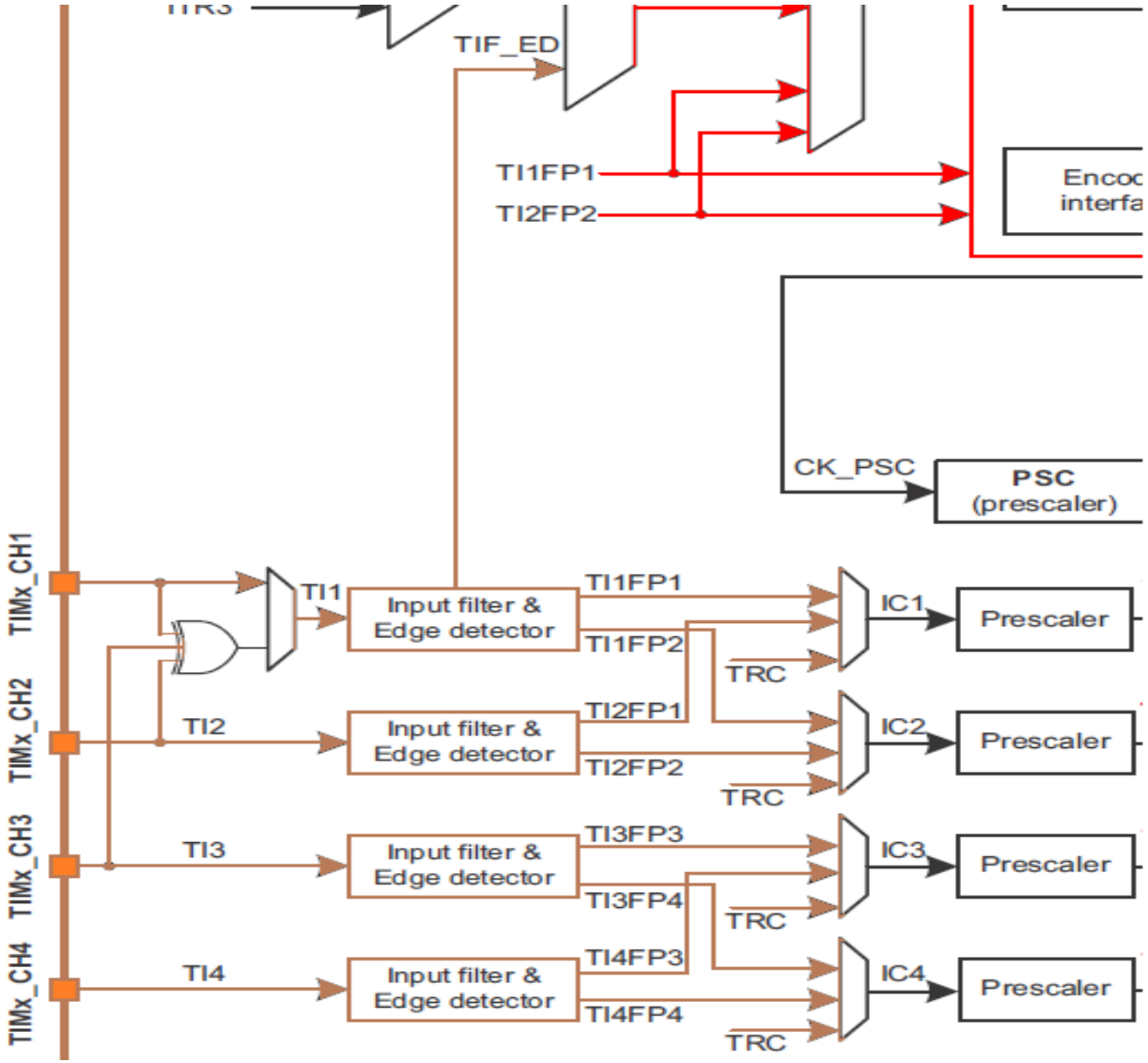
基于不同时钟源应用的配置示例【3】

- TI2做输入捕捉信号、TI1FP1做触发信号且做时钟，计数器工作在外部时钟模式1从模式。

- 看得懂否？信号怎么走的？输入信号、触发信号、捕捉信号、时钟信号各是谁？？



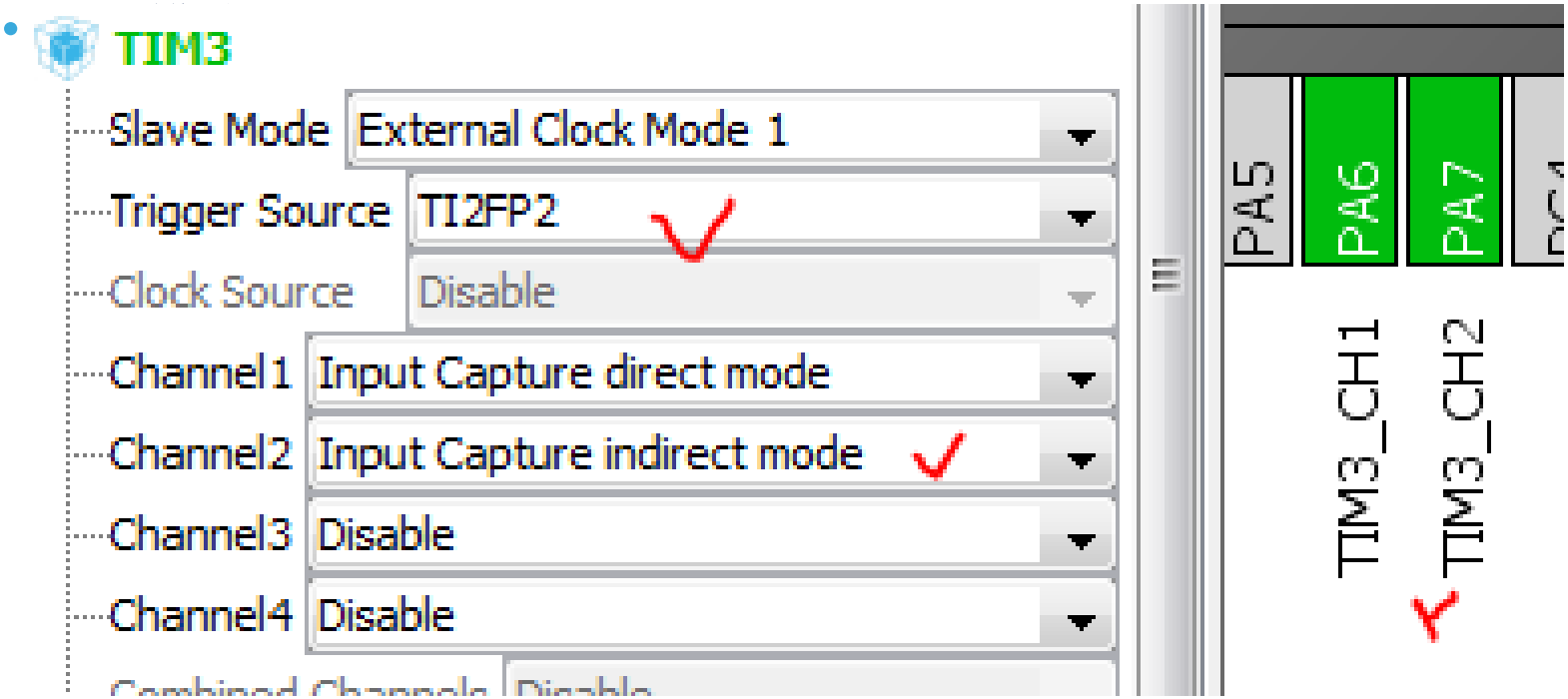
插播：输入捕捉信号与触发信号



life.augmented

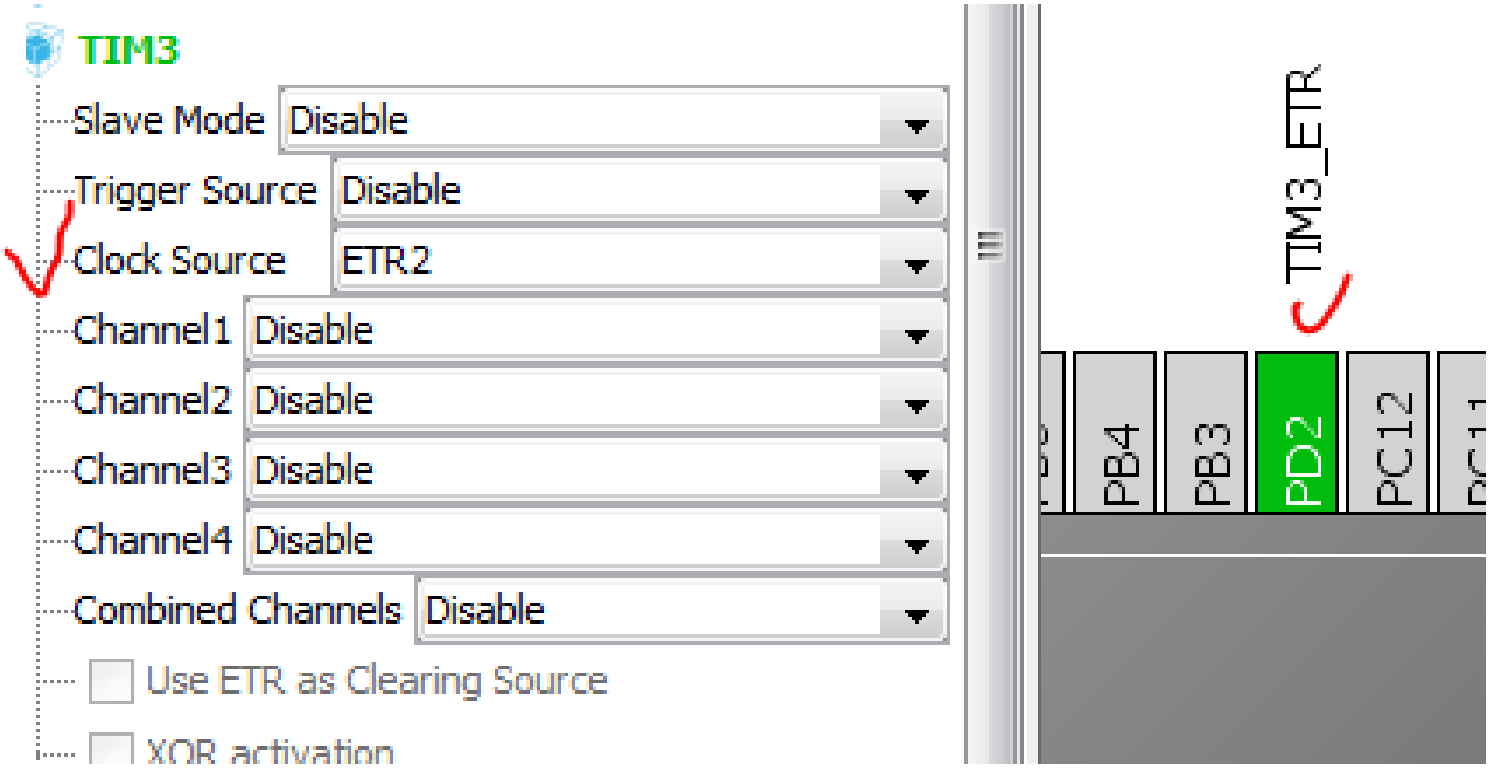
基于不同时钟源应用的配置示例【4】

- TI1 做输入捕捉信号、TI2FP2触发信号兼时钟，
- 计数器工作在外部模式1



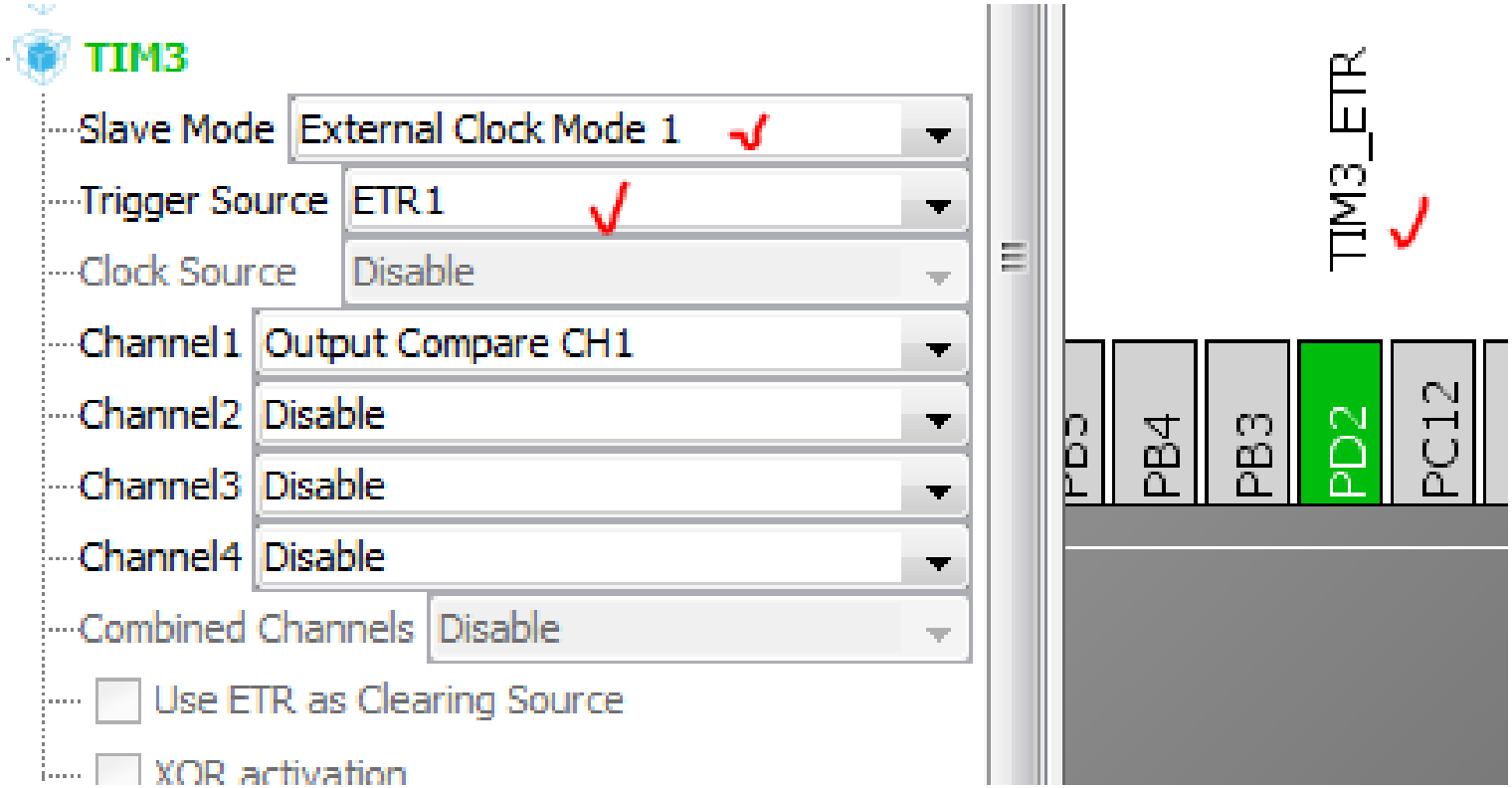
基于不同时钟源应用的配置示例【5】

- ETR提供时钟源，工作在主模式【前面提到过，再温习一遍】



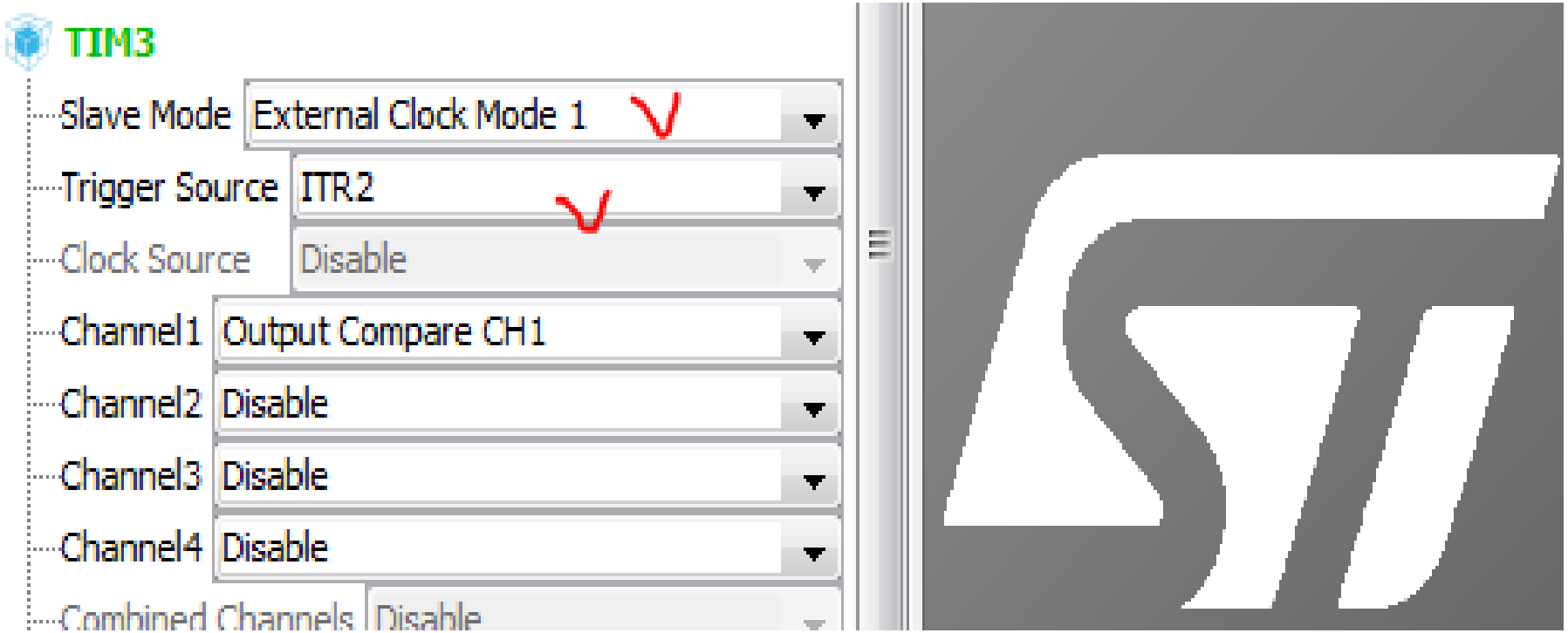
不同时钟源案例配置【6】

- ETR兼做时钟和触发信号，计数器工作在从模式---外部时钟模式1



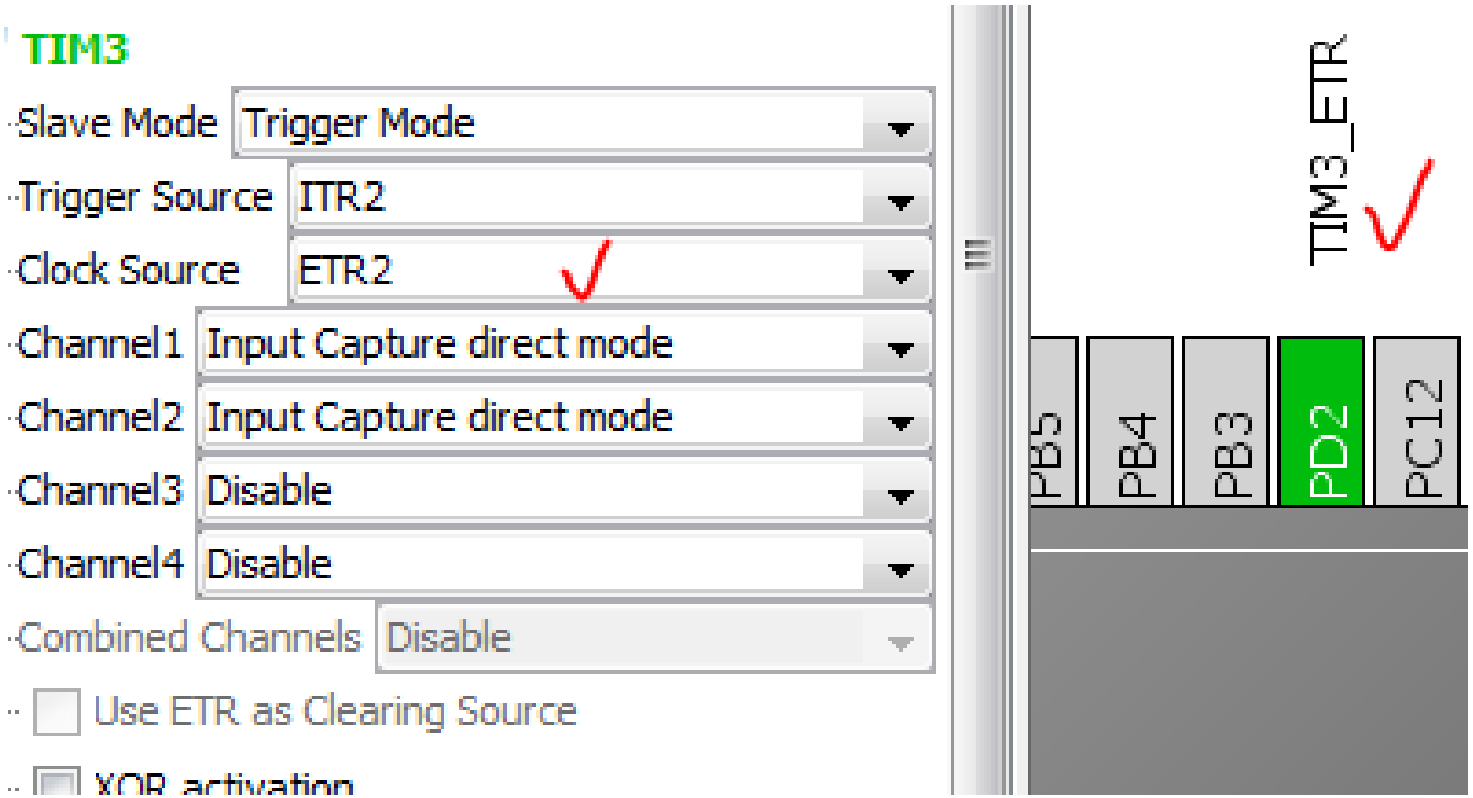
不同时钟源案例配置【7】

- 来自片内其它定时器触发输入信号作为时钟，工作在外部时钟模式1



不同时钟源案例配置【8】

- 触发信号来自其它定时器，ETR作为时钟
- 定时器模式可选外部时钟模式1以外的模式



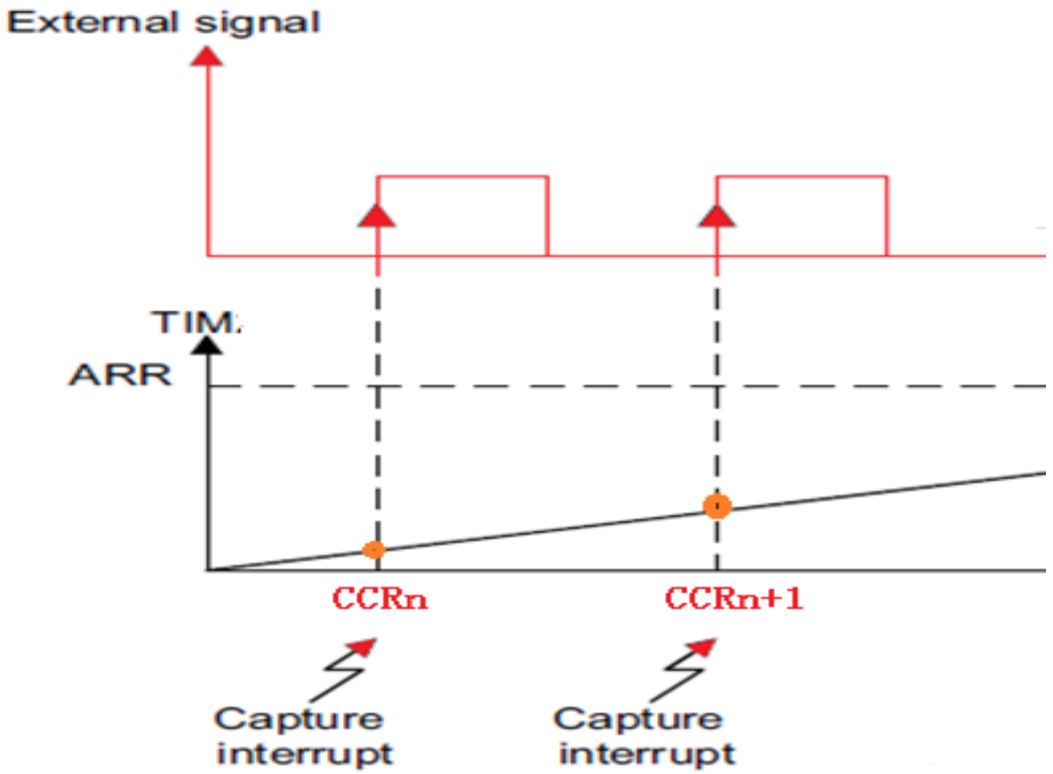
时钟源问题思考

- 1、几种时钟源？来自何处？
- 2、来自TI1fp1与TIF_ED信号差别是什么？
- 3、来自TI1/TI2的时钟是否可以直接像内部时钟一样被计数器用来计数？
- 4、根据时钟源的介绍，你了解到了几种触发输入信号？
- 5、TI3FP4可以作为通道2的输入捕捉信号吗？
- 6、TI4FP4可以作为触发信号吗？

三、定时器比较捕获【1】

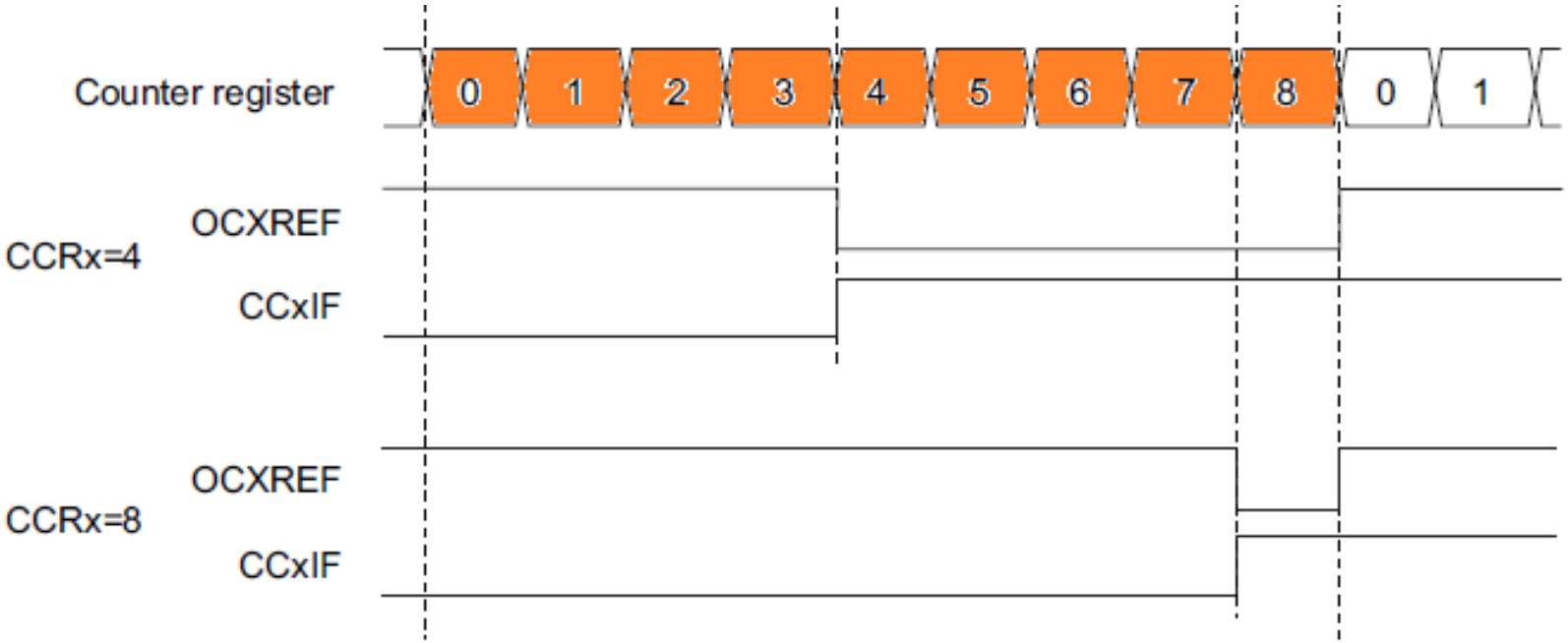
STM32定时器除了基本计数定时功能外，还对外拓展了输入、输出通道，从而实现输入捕捉、比较输出功能。

输入捕获【Input Capture】功能：捕捉外部触发信号、事件；可以触发中断或DMA请求，配合计数器对外来信号测量周期或频率，甚至可以用来做通信解码。

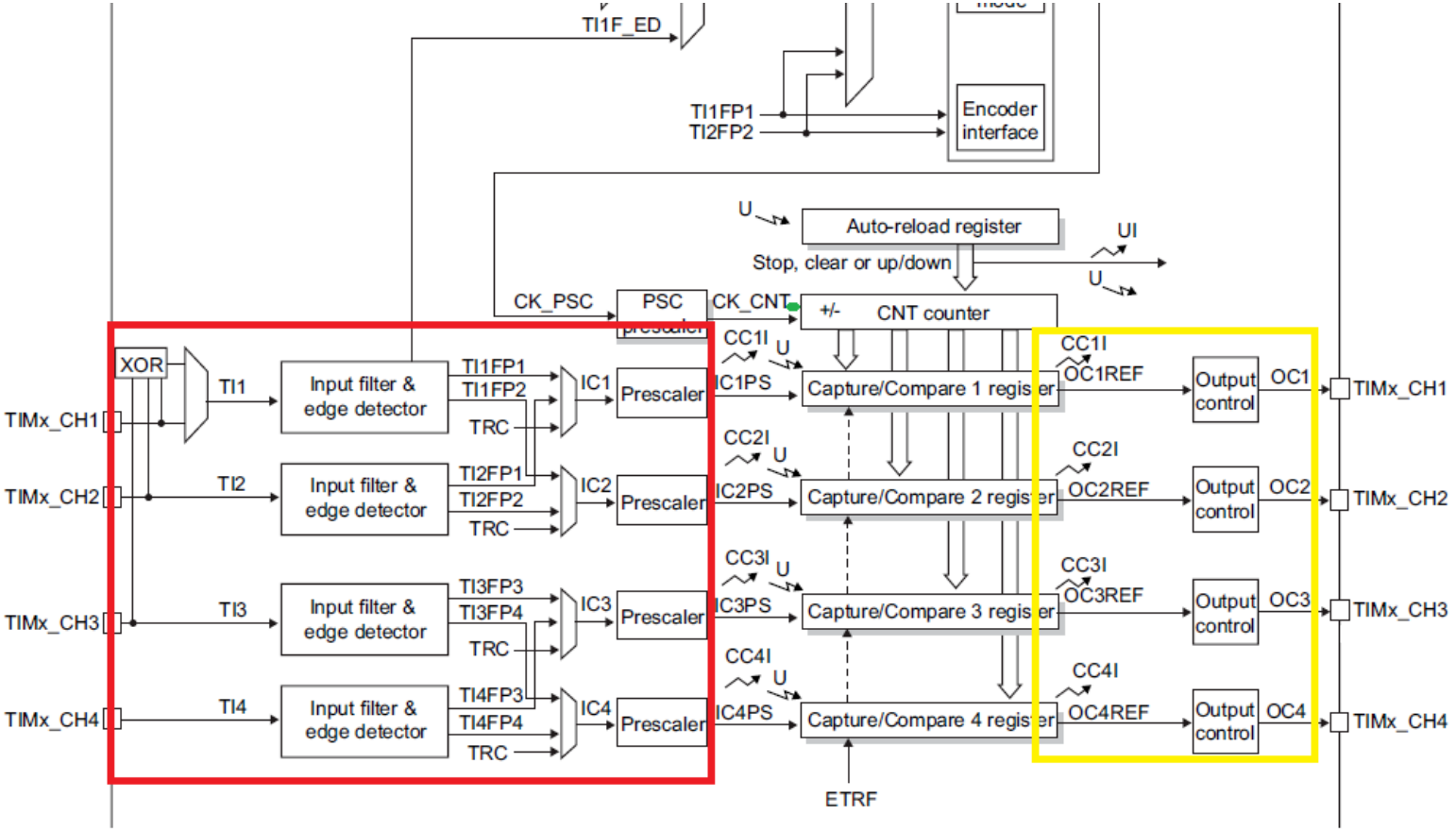


三、定时器比较捕获【2】

比较输出【Compare Output】功能：定时器通过对预定的比较值与计数器的值做匹配比较，实现各类输出。如PWM输出、电平翻转、单脉冲输出、强制输出等。一般来讲，STM32的通用定期和高级定时器都具有比较捕获功能，不同的定时器可能通道数有差异。



定时器捕获比较框图



- 注意：左右两边的重复编号通道只是逻辑上的，物理上只有1个通道与之对应。只是每个通道既可配置为输入捕获、也可配置为比较输出。

捕获/比较单元基本构成及功能

- 捕获/比较单元主要寄存器：捕捉比较寄存器**CCR**，它由其**影子寄存器/预装载寄存器**组成。用户访问时访问预装寄存器。预装载功能可开启或关闭。由OCxPE@TIMx_CCER控制。

- 输入捕获模式下：当捕获单元捕捉到外来有效信号边沿事件时，将此刻的计数器的值锁存到CCR影子寄存器并自动将CCR影子寄存器的值拷贝进CCR预装载寄存器，以供用户读取。

- 强调：此时CCR对用户是只读的，不可对其进行修改、赋值。】

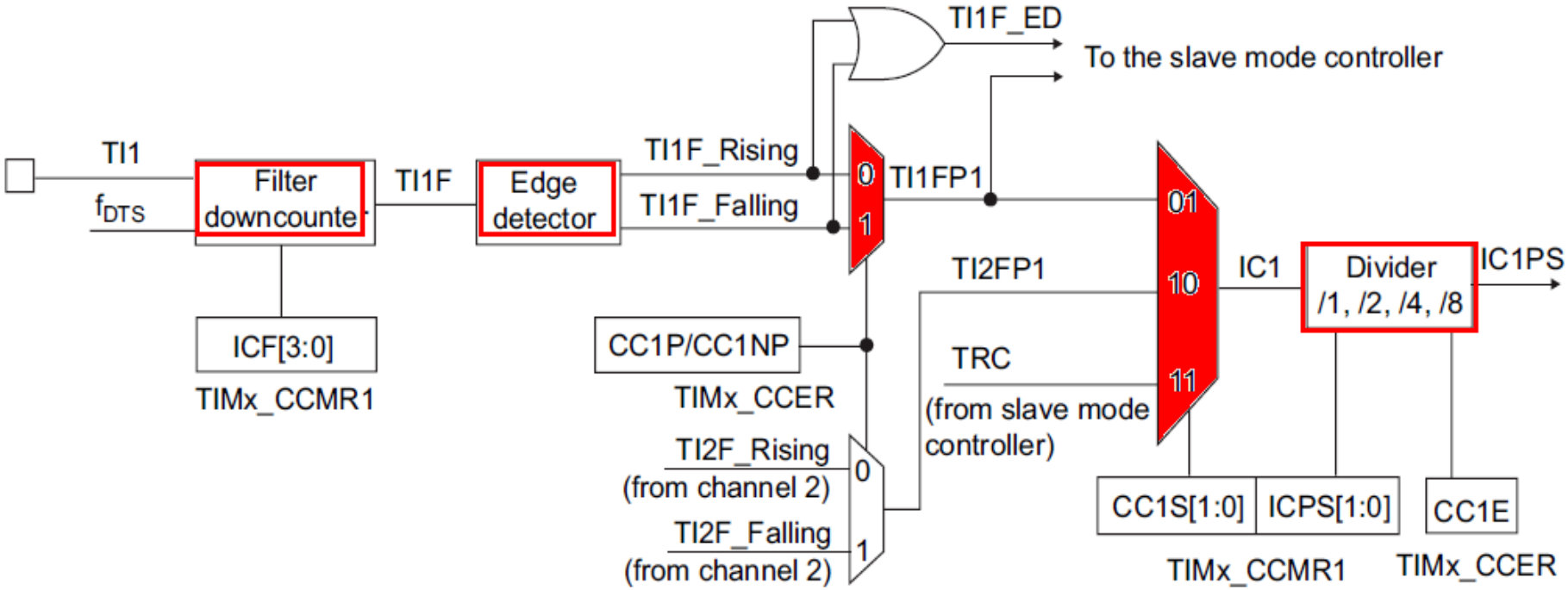
- 比较输出模式下：当捕获比较单元监测到计数器的值与CCR寄存器的数字匹配时，将根据相应的比较输出模式实现相应输出。

- 先看看输入捕获功能。

定时器输入捕获流程

外部输入信号一般经过滤波、边沿检测、极性选择、捕捉信号选择

、捕捉信号分频后进入捕捉单元



输入捕获相关事件

- 1、捕获时刻的计数器值的锁存到对应捕获通道的CCRx寄存器；
- 2、对应通道的捕获事件/中断标志会被置位；【CCxIF@TIMx_SR】
- 3、如果允许了 捕捉中断或DMA请求，可以触发中断或DMA请求。其它取决于CCxIE/CCxDE @TIMx_DIER的配置。
- 4、如果发生连续的捕获事件，前次CCxIF没有清零的话会置位CCxOF@TIMx_SR.
- 5、对于CCxIF标志可以软件清零或读取CCR来清零
- 6、可以通过置位CCxG@TIMx_EGR来软件触发输入捕获事件

与捕获比较直接相关的两个控制寄存器

- 1、捕获比较模式寄存器；

TIM1 and TIM8 capture/compare mode register 2 (TIMx_CCMR2)

Address offset: 0x1C

Reset value: 0x0000

Refer to the above CCMR1 register description.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OC4 CE	OC4M[2:0]			OC4 PE	OC4 FE	CC4S[1:0]		OC3 CE.	OC3M[2:0]			OC3 PE	OC3 FE	CC3S[1:0]	
IC4F[3:0]				IC4PSC[1:0]				IC3F[3:0]				IC3PSC[1:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

- 2、捕获比较使能寄存器

TIM1 and TIM8 capture/compare enable register (TIMx_CCER)

Address offset: 0x20

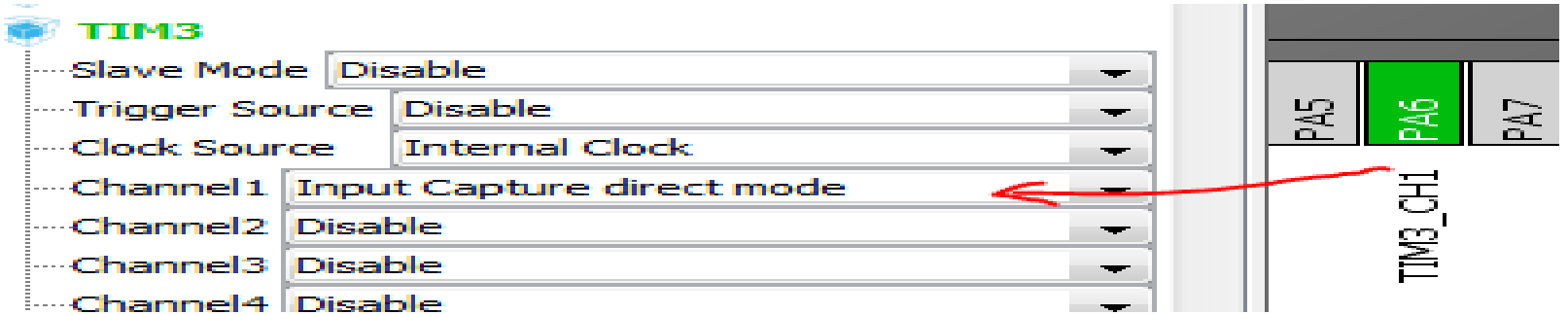
Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved		CC4P	CC4E	CC3NP	CC3NE	CC3P	CC3E	CC2NP	CC2NE	CC2P	CC2E	CC1NP	CC1NE	CC1P	CC1E
		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

通道1输入捕获示例

- 对TI1信号进行采样滤波，得到TI1F
  - IC1F@CCMR
  - $f_{SMAPLING} = f_{CK_INT}$或$f_{DTS}$分频因子、滤波N可配置
- 经过极性选择的边沿检测，得到TI1FP1
  - CC1P@CCER
  - 控制TI1FP1和TI2FP1的触发或捕获极性
    - 上升沿/下降沿/两个边沿
- 作为触发输入，送到从模式管理器
- 作为被捕获的信号IC1，经过分频后得到IC1PS
  - ICPS@CCMR1
  - IC1经过1、2、4、8分频得到最终的捕获信号IC1PS
- 捕获/比较使能控制
  - CC1E@CCER

通道1输入捕获示例



TIM3 Configuration

Parameter Settings | User Constants | NVIC Settings | DMA Settings | GPIO Settings

Configure the below parameters :

Search : Search (Crtl+F)

Counter Settings	
Prescaler (PSC - 16 bits value)	0
Counter Mode	Up
Counter Period (AutoReload Register - 16 bits value)	888
Internal Clock Division (CKD)	No Division
Trigger Output (TRGO) Parameters	
Master/Slave Mode (MSM bit)	Disable (Trigger input effect not delayed)
Trigger Event Selection	Reset (UG bit from TIMx_EGR)
Input Capture Channel 1	
Polarity Selection	Rising Edge
IC Selection	Direct
Prescaler Division Ratio	Division by 2
Input Filter (4 bits value)	4

通道1输入捕获示例

- 下面代码针对极性、通道选择、信号的分频、输入滤波进行设置。

```
sConfigIC.ICPolarity = TIM_INPUTCHANNELPOLARITY_RISING;
sConfigIC.ICSelection = TIM_ICSELECTION_DIRECTTI;
sConfigIC.ICPrescaler = TIM_ICPSC_DIV2;
sConfigIC.ICFilter = 4;
if (HAL_TIM_IC_ConfigChannel(&htim3, &sConfigIC, TIM_CHANNEL_1) != HAL_OK)
{
```

- 基于Cube Hal库，相关实现代码可以选择下面函数：
- HAL_TIM_IC_Start (TIM_HandleTypeDef *htim, uint32_t Channel)
- HAL_TIM_IC_Start_IT (TIM_HandleTypeDef *htim, uint32_t Channel)
- HAL_TIM_IC_Start_DMA(TIM_HandleTypeDef *htim, uint32_t Channel, uint32_t *pData, uint16_t Length)

通过输入捕获测量外来脉冲周期

- 在输入捕捉模式下，定时器可用于测量外部信号周期。根据定时器时钟、预分频器和定时器分辨率，可推导出最大测量周期。
- 相应的定时器配置包括：
- 1. 通过对CCMRx 寄存器中的 CCxS 位选择有效输入。
- 2. 根据需要，通过对 CCMRx 寄存器中的 IC1F[3:0] 位执行操作以编程滤波器参数，并通过对IC1PSC[1:0] 位执行写操作以编程预分频器。
- 3. 通过对 CCxNP/CCxP 位执行写操作选择极性，即选择上升沿触发、下降沿触发或边沿触发。

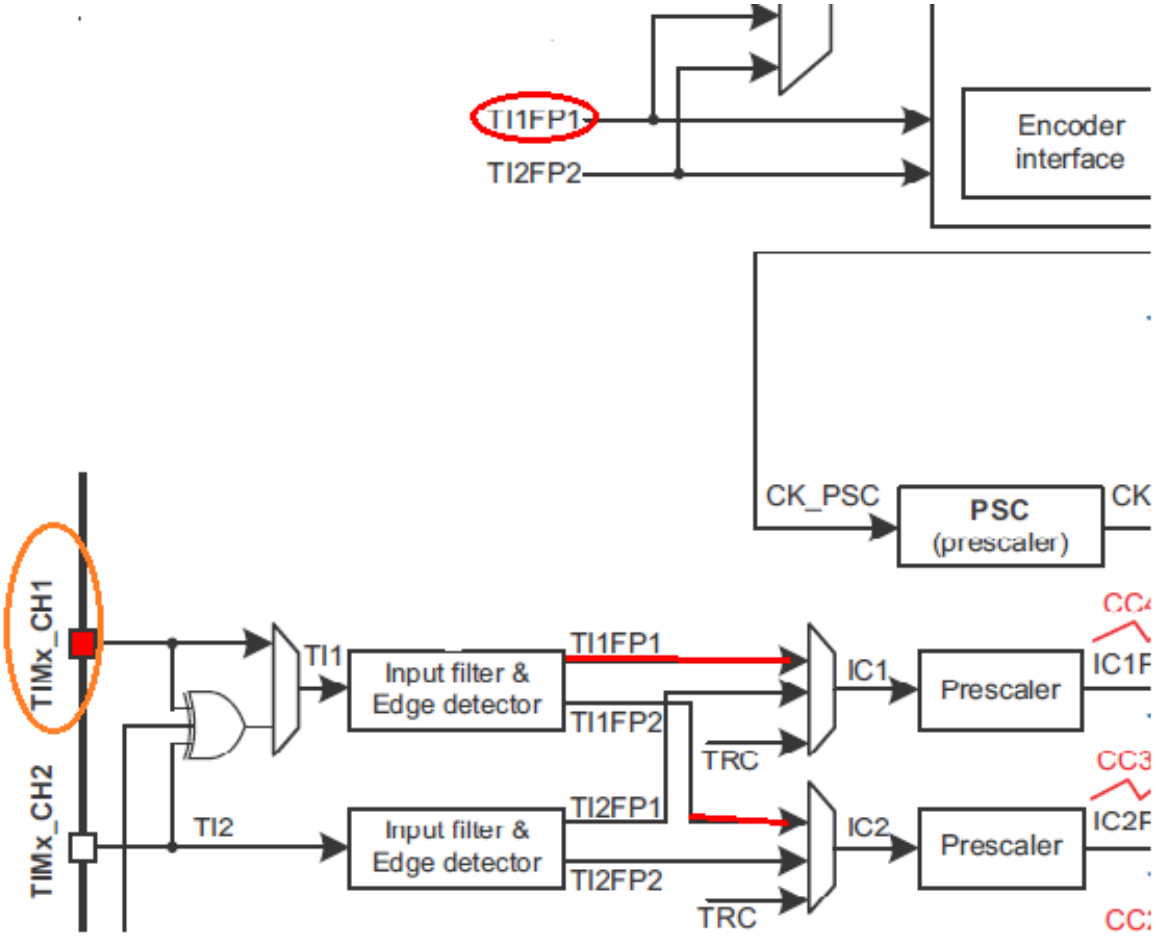
通过输入捕获测量外来脉冲周期（续）

- 当输入通道检测到相应的信号发生跳变时，可使用输入捕捉模块进行捕捉。要获取外部信号周期，需连续进行两次捕捉。通过将两次捕捉到的值相减可计算得到周期。
- 对于单沿捕捉，待测信号不长于计数器的计数周期时：
- 假设两次连续捕捉值为 CCRx_tn 和 CCRx_tn+1
- ● 如果 CCRx_tn < CCRx_tn+1：捕捉值= CCRx_tn+1 - CCRx_tn
- ● 如果 CCRx_tn > CCRx_tn+1：捕捉值= (ARR_max - CCRx_tn) + CCRx_tn+1
- 如果待测信号周期长于当前计数器周期的话，需要考虑溢出问题

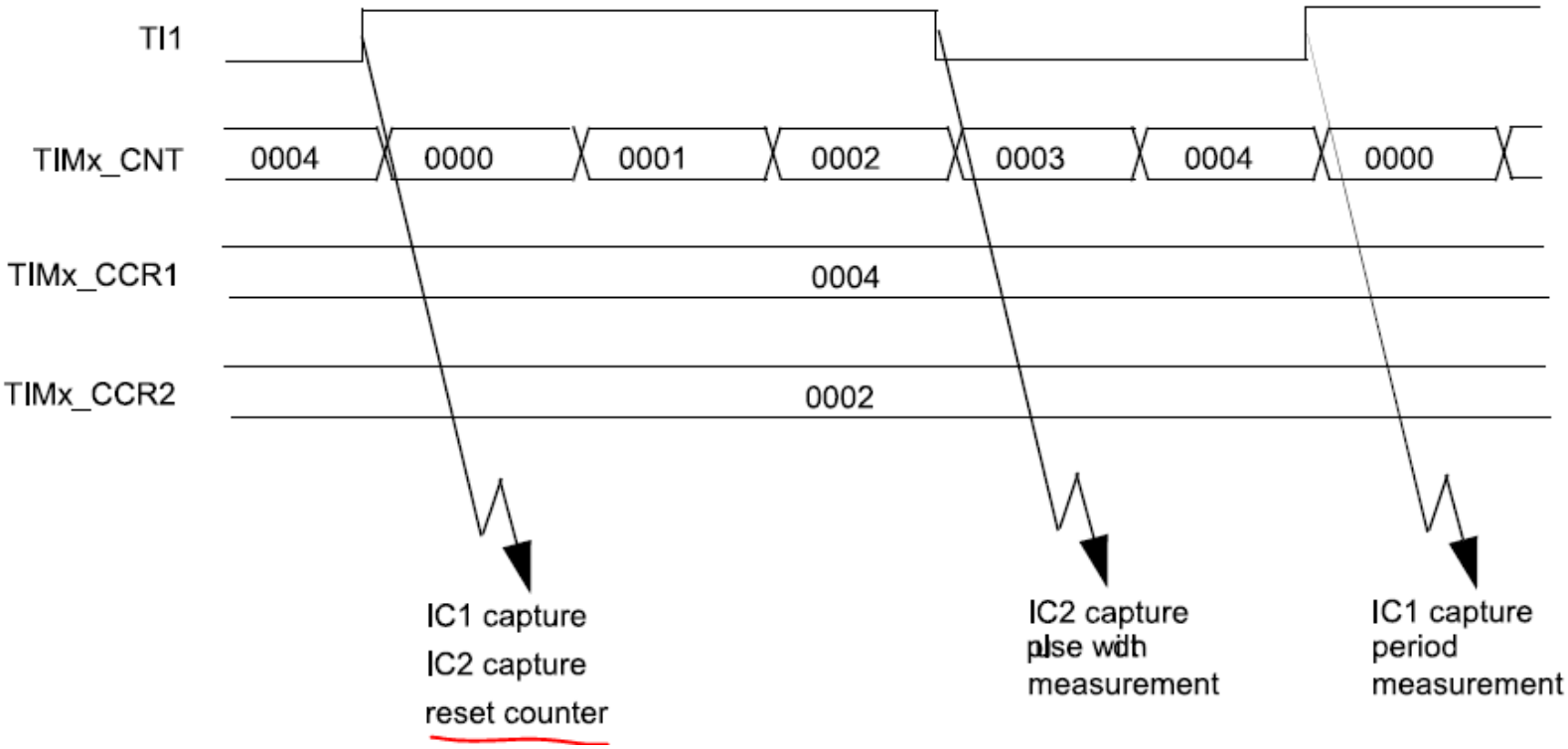
PWM输入模式工作原理【1】

- 是基于输入捕获与定时器从模式相结合的一个具体应用
- 同一个外部输入引脚【仅限于TI1/TI2的】的输入滤波信号【TIxFPx】映射到2个捕捉通道【仅限于IC1/IC2】，且配置为相反的捕捉极性，即一个通道捕捉上沿，另一个通道捕捉下沿。
- 定时器配置在复位从模式，TIxFPx作为触发信号。
- 下面示例中，待捕捉信号来自于定时器通道1，即TI1,其滤波信号TI1FP1作为触发信号。
- 经过两次连续的捕捉可测得信号的周期及占空比。
- 【前提：待测信号周期不长于当前定时器计数周期】

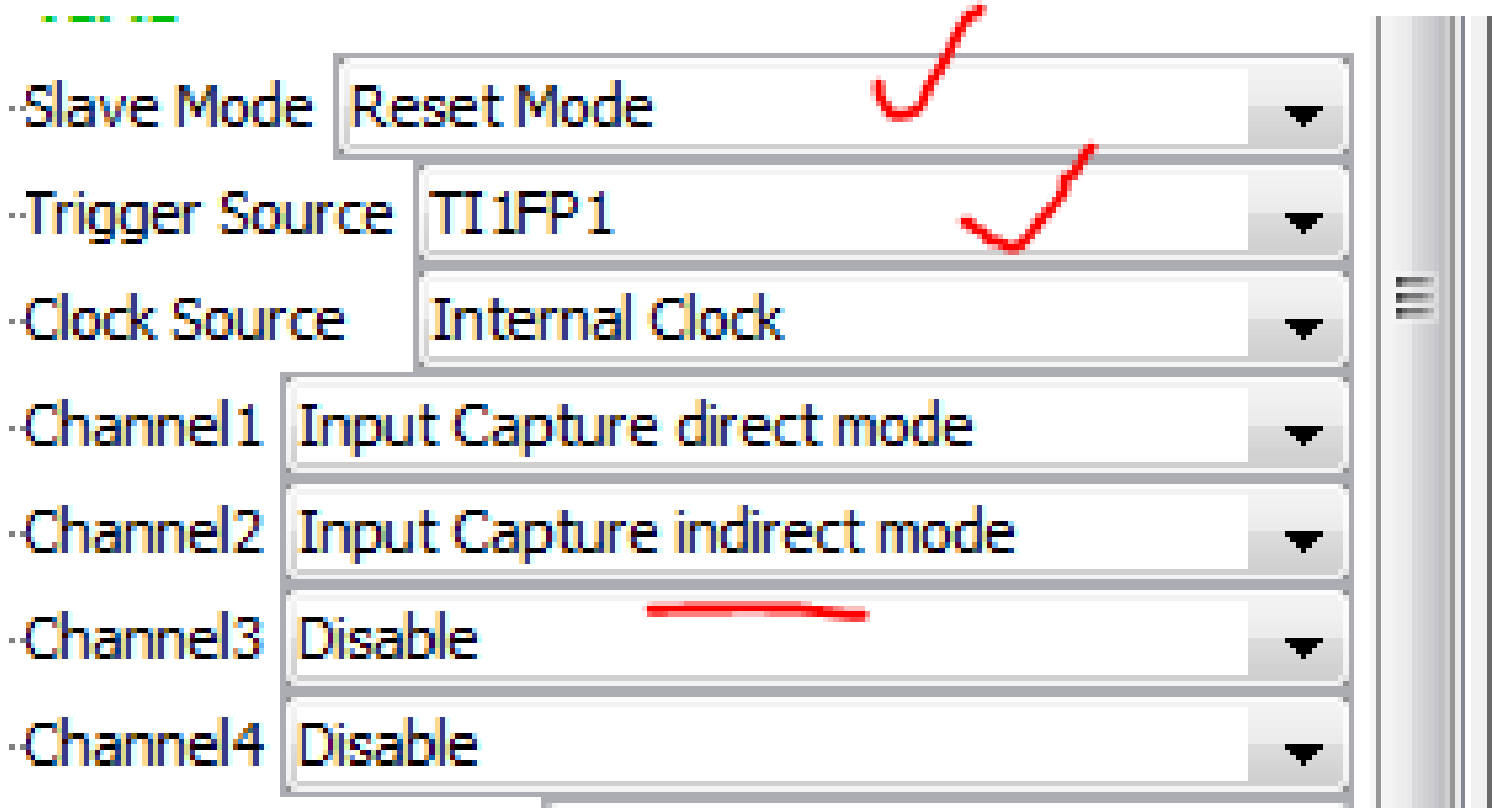
PWM输入模式工作原理【2】



PWM输入模式图例



PWM输入模式举例



PWM输入模式举例

Counter Settings	
Prescaler (PSC - 16 bits value)	0
Counter Mode	Up
Counter Period (AutoReload Register - 16 bits value)	888
Internal Clock Division (CKD)	No Division
Slave Mode Controller	Reset Mode
Trigger Output (TRGO) Parameters	
Master/Slave Mode (MSM bit)	Disable (Trigger input effect not delayed)
Trigger Event Selection	Reset (UG bit from TIMx_EGR)
Input Capture Channel 1	
Polarity Selection	Rising Edge
IC Selection	Direct
Prescaler Division Ratio	No division
Input Filter (4 bits value)	0
Input Capture Channel 2	
Polarity Selection	Falling Edge
IC Selection	Indirect
Prescaler Division Ratio	No division

关于输入捕获测量异常的提醒

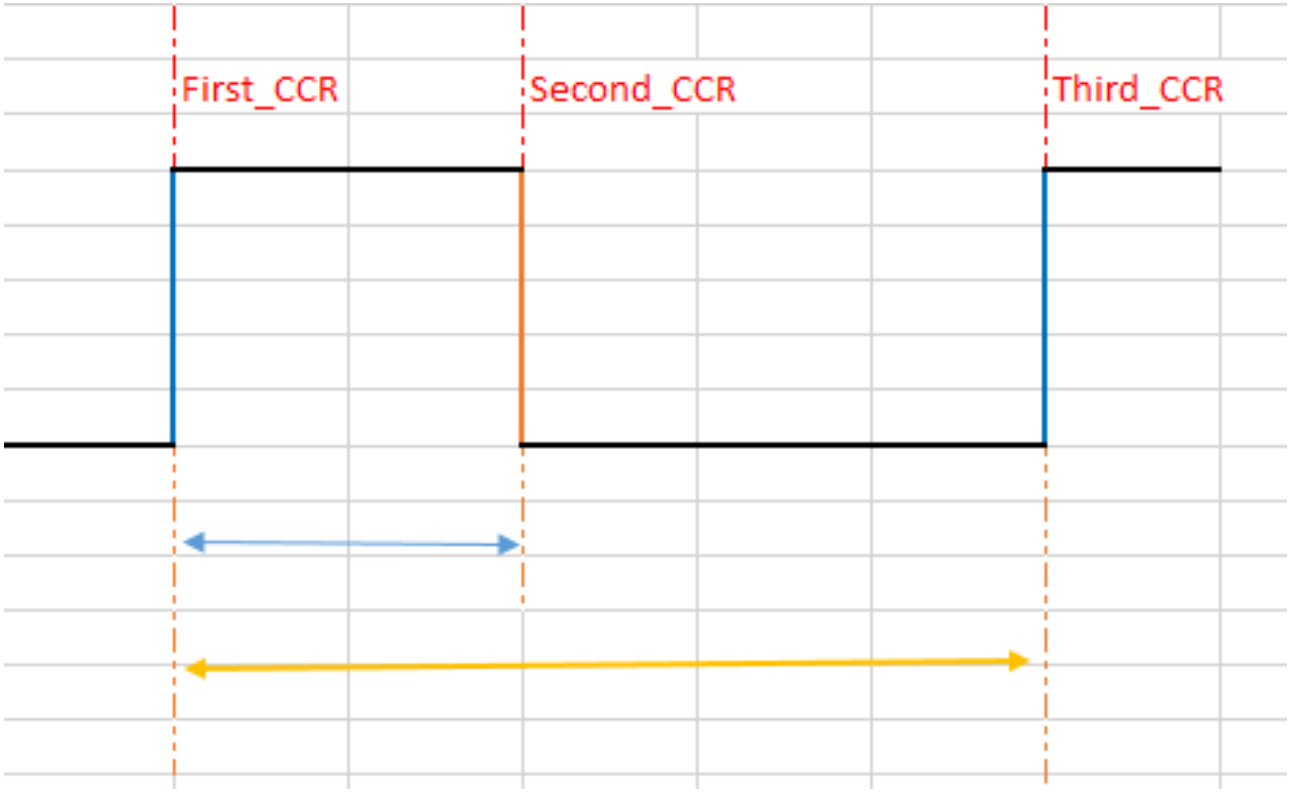
- 实际应用过程中使用TIMER进行输入捕获测量时间失败者较常见，常见的原因可能有如下两方面：
- 1、待测量的脉冲长于ARR所对应的时间，代码又没考虑更新事件；
- 2、使用PMW输入模式测量时选择了TI1/TI2以外的通道；
- 2个提醒：
- CCR寄存器整体上是可读可写的。但是，当相应通道配置为捕获输入时，CCR为只读寄存器，即用户不可修改或赋值！！！
- 在发生捕获事件时，CCR影子寄存器的内容由硬件自动将其拷贝到CCR预装寄存器，以供用户或DMA读取！！！！

关于输入捕获案例

- 平常ST官方库有输入捕获的例程，但有个前提待测信号周期不要长于1个定时器溢出周期。但实际应用中，待测信号可能长于1个定时器溢出周期。
- 一方面我们可以调整定时器溢出周期，以匹配待测信号周期；
- 另一方面，我们可以调整代码，计算定时器自身的更新次数来实现测量的目的。毕竟有时可能也不便于无限拉长定时器计数周期。
- 下面分享几个实例。

输入捕获案例【1-0】

一、单通道测量外来脉冲占空比和周期【考虑溢出更新】。



输入捕获案例【1-1】

- 使用1个输入通道TIM4_CH2测量外来脉冲的参考流程。
- 1、开启通道2的捕捉中断，初始化变量。Measure_Flag==0表示尚未开启测试。
- 2、对该脉冲捕捉3次，依次捕捉上沿、下沿、上沿。三次的捕捉值分别记为: First_CCR； Second_CCR； Third_CCR.在做第一次捕捉和第二次捕捉后做捕捉极性的切换。
- 3、开启对更新中断的计数，使用变量Cnt_Upevent记录溢出更新次数。
- 在第2次和第3次捕捉的时候，记录自开始测量以来发生的更新次数分别为： Width_Cnt_Upevent 和 Total_Cnt_Upevent。

输入捕获案例【1-2】

- 待测信号占空比
- (float)(Second_CCR+(Width_Cnt_Upevent*(TIM4_PERIOD+1))-First_CCR) *100u /(Third_CCR+(Total_Cnt_Upevent*(TIM4_PERIOD+1))-First_CCR);
- 待测信号频率：
- (float) (计数器频率) /(Third_CCR +(Total_Cnt_Upevent*(TIM4_PERIOD+1))-First_CCR);

输入捕获案例【1-3】


```
/* Input capture event */
if((htim->Instance->CCMR1 & TIM_CCMR1_CC2S) != 0x00U)
{

  if(Measure_Flag==0)
  {
  Measure_Flag = 0x01;      //////
  First_CCR = TIM4->CCR2;   /////
  Cnt_Upevent = 0 ;          /////
  TIM4->CCER|=0x0020;    //CHANGE POLARITY TO FALLING
  htim->Channel = HAL_TIM_ACTIVE_CHANNEL_CLEARED;
  return;
  }

  if(Measure_Flag == 0x01)
  {
  Second_CCR = TIM4->CCR2;
  Width_Cnt_Upevent = Cnt_Upevent;
  Measure_Flag = 0x02;
  TIM4->CCER &=0x3f5f;   ////CHANGE POLARITY TO RISING

  htim->Channel = HAL_TIM_ACTIVE_CHANNEL_CLEARED;
  return;
  }
```

输入捕获案例【1-4】

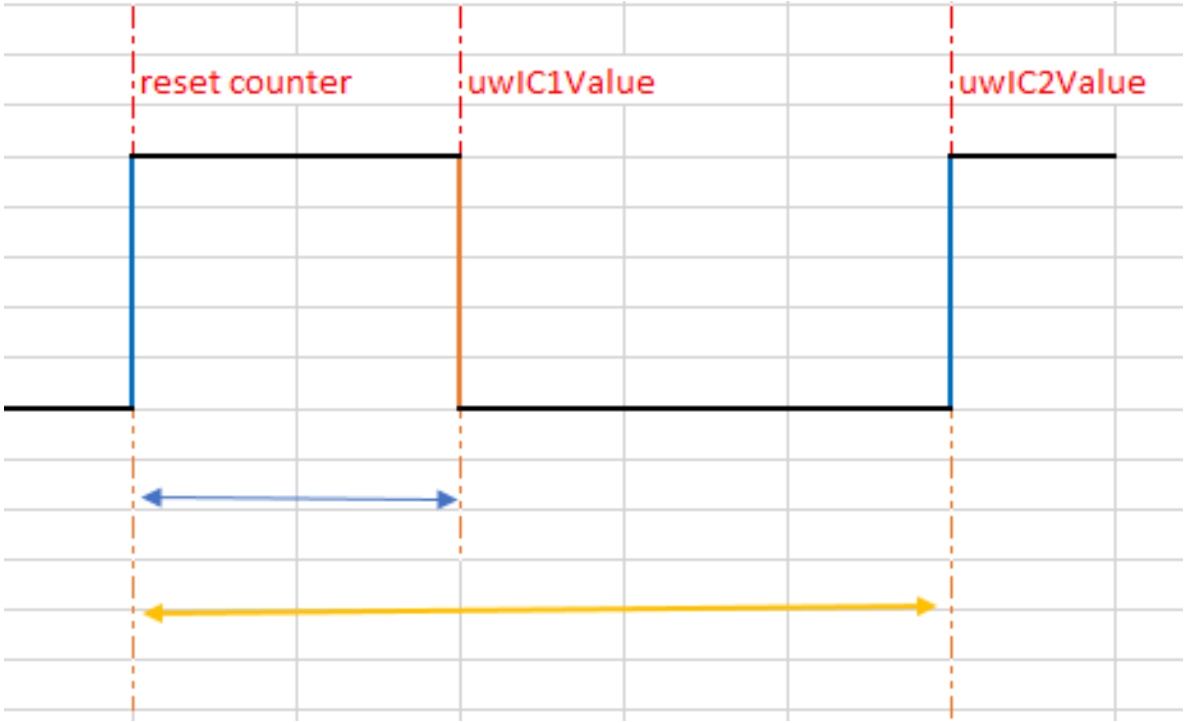

```
if(Measure_Flag == 0x02)
{

  Third_CCR = TIM4->CCR2;
   Total_Cnt_Upevent = Cnt_Upevent;
   HAL_TIM_IC_CaptureCallback(htim);//to calculating frequence/duty


/* 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);

  //  HAL_TIM_PeriodElapsedCallback(htim);

    if((Measure_Flag > 0) && (Measure_Flag < 3))
    {
    Cnt_Upevent++;    //Counting update only during this window
    }

  }
}
```

输入捕获案例【1-2】

- 待测信号占空比
- (float)(Second_CCR+(Width_Cnt_Upevent*(TIM4_ARR+1))-First_CCR) *100u /(Third_CCR+(Total_Cnt_Upevent*(TIM4_ARR+1))-First_CCR);
- 待测信号频率：
- (float) (计数器频率) /(Third_CCR +(Total_Cnt_Upevent*(TIM4_ARR+1))-First_CCR);

输入捕获案例【2-0】

- 1、使用PWM输入模式测量一外来脉冲，要考虑测量过程中的定时器溢出事件。定时器为向上计数模式。
- 2、测量的脉冲远长于定时器ARR所对应的时间，需考虑溢出；
- 3、待测信号从TI2进来，TI2FP2作为复位触发脉冲，TI2FP1间接连接到通道IC1.



输入捕获案例【2-1】

- 因为使用PWM输入模式，使用到两个输入通道做捕获，分别捕捉两个相反极性的边沿。IC1捕捉下降沿，IC2捕捉上升沿并做触发信号。

```
sSlaveConfig.SlaveMode = TIM_SLAVEMODE_RESET;
sSlaveConfig.InputTrigger = TIM_TS_TI2FP2;
sSlaveConfig.TriggerPolarity = TIM_INPUTCHANNELPOLARITY_RISING;
sSlaveConfig.TriggerFilter = 0;
if (HAL_TIM_SlaveConfigSynchronization(&htim4, &sSlaveConfig) != HAL_OK)
{

 sConfigIC.ICPolarity = TIM_INPUTCHANNELPOLARITY_FALLING;
 sConfigIC.ICSelection = TIM_ICSELECTION_INDIRECTTI;
 sConfigIC.ICPrescaler = TIM_ICPSC_DIV1;
 sConfigIC.ICFilter = 0;
 if (HAL_TIM_IC_ConfigChannel(&htim4, &sConfigIC, TIM_CHANNEL_1) !
 {

 sConfigIC.ICPolarity = TIM_INPUTCHANNELPOLARITY_RISING;
 sConfigIC.ICSelection = TIM_ICSELECTION_DIRECTTI;
 sConfigIC.ICFilter = 0;
 if (HAL_TIM_IC_ConfigChannel(&htim4, &sConfigIC, TIM_CHANNEL_2) != HAL_OK)
 {
```

输入捕获案例【2-2】

- 因为计数器被配置在复位模式，所以当TI2FP2过来上沿触发信号时，会同时触发2个事件，触发事件和更新事件。置位TIF/UIF@TIMx_SR

- 可以考虑使用TIF中断来作为测量开始，在TIF触发的中断里设置测试开启标志并对更新事件计数器Cnt_Upevent清零。同时也清除更新中断标志。分别在基于通道1、通道2的捕捉中断里获取捕捉值和当时的更新次数。

```
/* Input capture event */
if((htim->Instance->CCMR1 & TIM_CCMR1_CC1S) != 0x00U)
{
 // HAL_TIM_IC_CaptureCallback(htim);
   if (Measure_Flag != 0)
   {
    Front_Cnt_Upevent = Cnt_Upevent;
   }
}
```

输入捕获案例【2-3】


```
/* Input capture event */
if((htim->Instance->CCMR1 & TIM_CCMR1_CC2S) != 0x00U)
{
  if(Measure_Flag != 0)
  {
  Total_Cnt_Upevent = Cnt_Upevent;
  HAL_TIM_IC_CaptureCallback(htim);//to calculating freq/duty
  }
}
```

两个通道的捕捉值分别是 uwIC1Value，uwIC2Value

两次捕捉时读到的更新次数分别为：Front_Cnt_Upevent和

Total_Cnt_Upevent

输入捕获案例【2-4】

- 更新次数统计参考代码

```
if(__HAL_TIM_GET_FLAG(htim, TIM_FLAG_UPDATE) != RESET)
{
    __HAL_TIM_CLEAR_IT(htim, TIM_IT_UPDATE);

  if(__HAL_TIM_GET_FLAG(htim, TIM_FLAG_TRIGGER) != RESET)
    {
        __HAL_TIM_CLEAR_IT(htim, TIM_IT_TRIGGER);

        if (Measure_Flag == 0)

        {
          Cnt_Upevent = 0;  //clear it to count update event
          Measure_Flag = 0xff; //start measure
        }
     }

  else
  {
    if (Measure_Flag != 0)
    {
     Cnt_Upevent++; //counting update times
    }
  }
}
```

输入捕获案例【2-5】

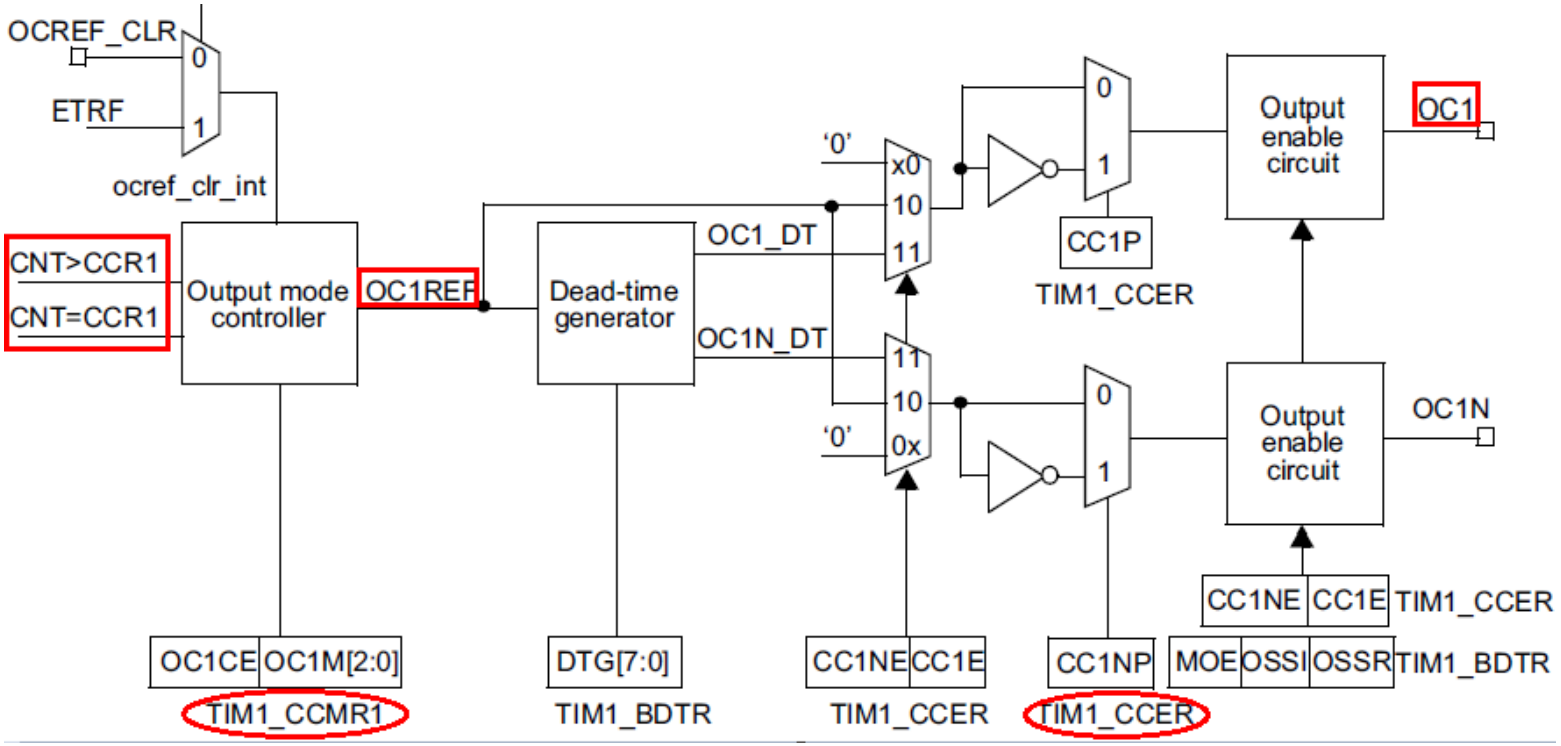
- 信号占空比：
- (float) ((uwIC1Value+(Front_Cnt_Upevent*(TIM4_ARR+1))) * 100) / (uwIC2Value+(Total_Cnt_Upevent*(TIM4_ARR+1)));
- 信号频率：
- uwFrequency =(float) (计数器时钟频率)/ (uwIC2Value+(Total_Cnt_Upevent*(TIM4_ARR+1)));

输入捕获小结

- 来自外部的输入信号等同于捕捉信号吗？
- 来自通道2的输入信号TI2可以作为通道1的捕捉信号吗？
- 来自通道3的输入信号TI3可以作为通道2的捕捉信号吗？
- 当某通道配置为输入时，对于通道的CCR寄存器可以对其赋值吗？
- PWM输入测量模式，用到了哪一种从模式？该模式有什么特点？

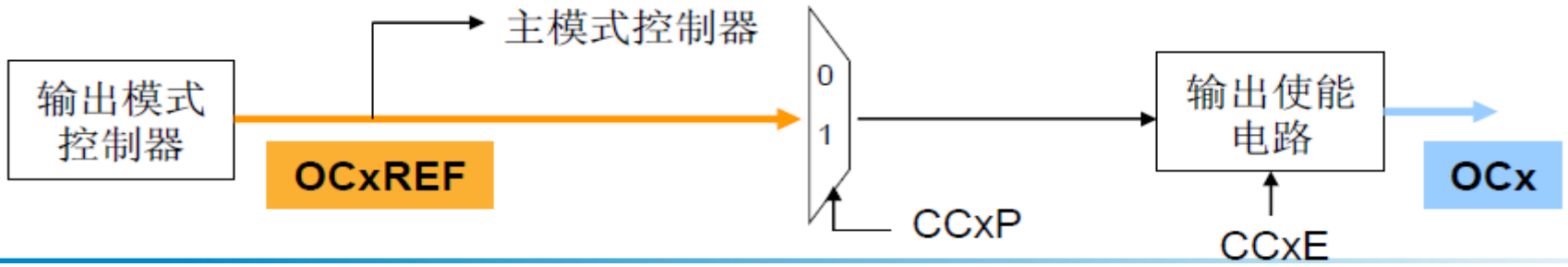
五、比较输出功能

- 基本原理：根据计数器与CCRx的比较结果，结合不同的输出控制模式
- 而输出相应的结果。【x代表不同的通道号，不同通道独立配置】



OCxREF 、OCx 、输出极性

- OCxREF源于输出模式控制器，并规定高电平为有效信号。它经过极性选择后，再经过输出控制电路输出。当CCxP=0时，高电平作为Ocx的有效输出信号，当CCxP=1时，低电平作为Ocx的有效输出。
- 输出端的active state【有效状态】/inactive state【无效状态】，取决于极性选择CCxP/CCxNP
- 对于非互补输出时 Ocx =OCxREF + 极性
- 对于互补输出时 Ocx =OCxREF + 极性 + 死区



输出比较模式概览

	输出模式 OCxM	描述	比较输出选择 CCxS
强制输出	100	OCxREF 强制输出低电平	通道配置输出 00
	101	OCxREF 强制输出高电平	
比较输出	001	比较匹配时，OCxREF 输出高电平	
	010	比较匹配时，OCxREF 输出 低电平	
	011	比较匹配时，OCxREF 电平翻转	
PWM 输出	110	向上计数，CNT<CCR,OCxREF 输出有效，否则无效输出电平。【先有效，后无效】	
		向下计数，CNT<CCR,OCxREF 输出有效，否则无效输出电平。【先无效，后有效】	
	111	向上计数，CNT<CCR,OCxREF 输出无效，否则输出有效电平。【先无效，后有效】	
		向下计数，CNT<CCR,OCxREF 输出无效，否则输出有效电平。【先有效，后无效】	
单脉冲模式	特殊的 PWM 输出模式		

比较输出相关事件

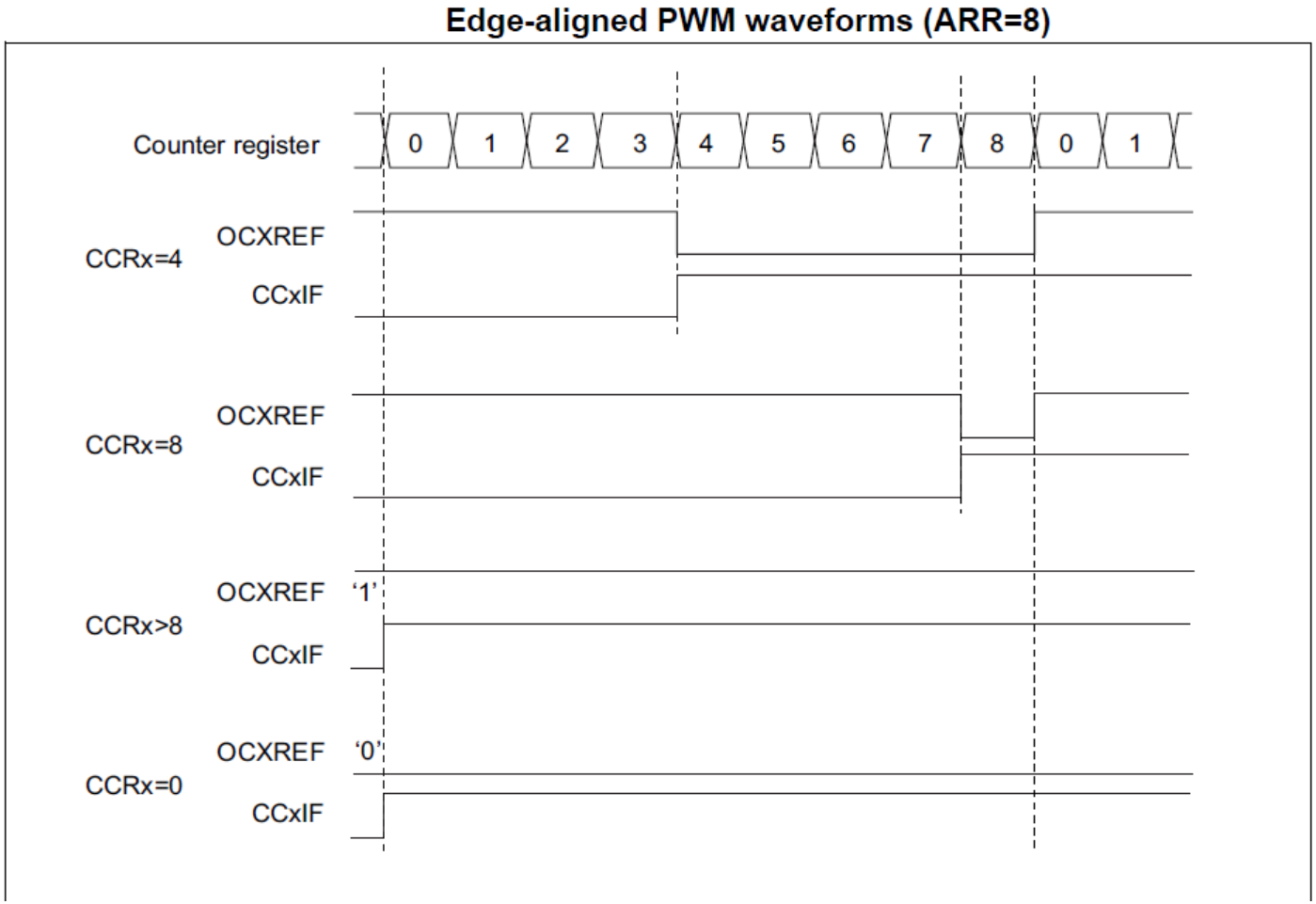
- 当核心计数器的值与比较寄存器的数值匹配时发生比较输出事件
- CCxIF@TIMx_SR被置位
- 触发比较中断 （如果CCxIE@TIMx_DIER被置位使能）
- 触发DMA请求 （CCxDE@TIMx_DIER CCDS@TIMx_CR2 使能允许）
- CCRx 影子寄存器的预装载功能可以软件开启或关闭 (OCxPE@TIMx_CCMR)

PWM输出频率的计算

- 定时器比较输出最常见的应用就是PWM输出，即脉宽调制输出。这里简单介绍下PWM波形参数的计算。以计数器向上计数、PWM1为例：
- PWM输出方波信号，信号的频率是由TIMx的时钟频率和TIMx_ARR这个寄存器所决定。输出信号的占空比由TIMx_CRRx寄存器确定。
- **占空比=(TIMx_CRRx/（TIMx_ARR+1）)*100%**
- F_pwm = CK_PSC/((PSC + 1)*(ARR + 1))
- 以中央对齐计数、PWM1为例：
- **占空比=(TIMx_CRRx/（TIMx_ARR）)*100%**
- F_pwm = CK_PSC/((PSC + 1)*ARR)

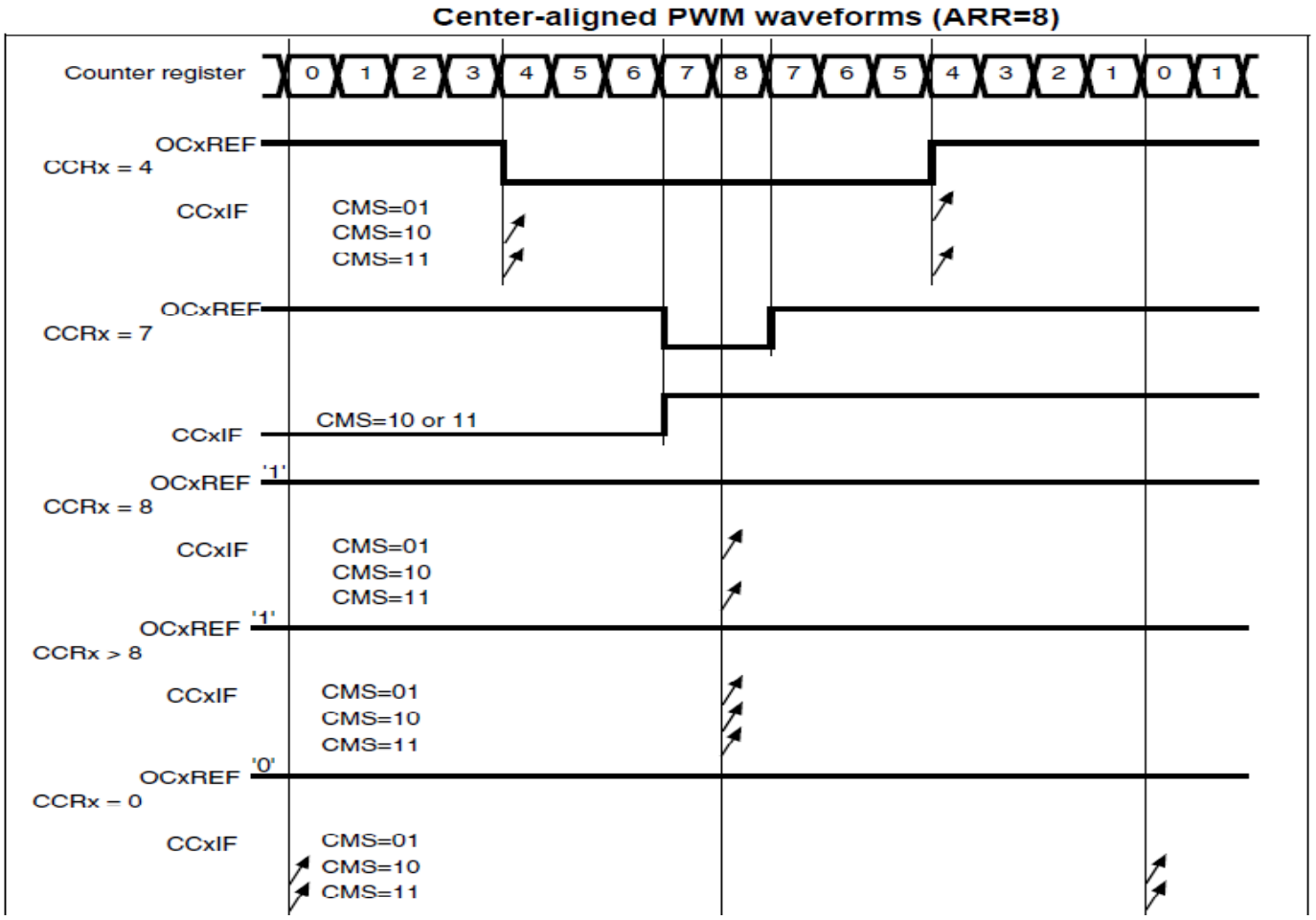
边沿对齐PWM输出波形示例

- Up Counting + PWM mode1



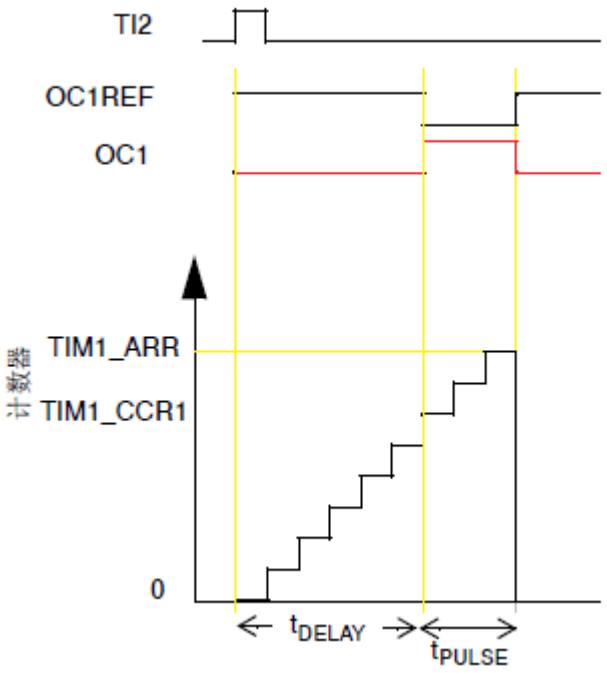
中心对齐PWM输出波形示例

- Center aligned Mode + PWM1



单脉冲模式

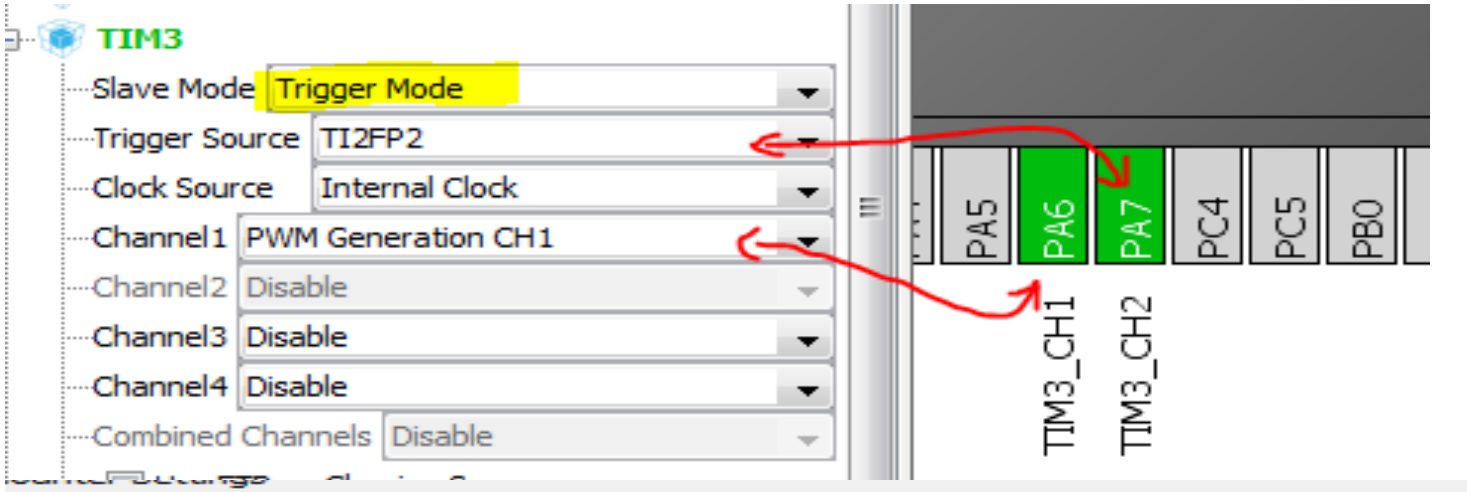
- 是PWM输出模式的一种特例。
- 原理：计数器启动后，在下一个更新事件来临之前的时间段内实现固定个数的脉冲输出。当下一个更新事件来临时计数器停止计数。脉冲个数可以一个或几个。如果是通用计数器就是1个，如果是高级定时器，脉冲个数与RCR数值及计数模式有关。
- 寄存器位OPM@TIMx_CR1置1
- 实现方式： 使用OC比较输出或PWM输出模式
- 计数器的启动可以通过自身软件使能启动也
- 可以将定时器配置在触发从模式经触发启动



单脉冲模式应用示例

- 比方在某通用定时器的TI2输入信号的上沿触发下，经过一段延时Tdelay,然后OC1输出一个宽带为Tpulse的单脉冲信号。
  - 通过输入捕获模块，配置对计数器的触发
    - 把IC2映射到TI2：IC2S=01@TIMx_CCMR1
    - 检测TI2的上升沿：CC2NP/CC2P=00@TIMx_CCER
    - 把TI2FP2导向从模式控制器作为TRGI的触发源：TS=110
    - 该定时器工作在触发从模式下：SMS=110@SMCR
      - TGRI的上升沿启动计数器
  - 通过比较输出模块，配置波形输出
    - 选择PWM2模式：OC1M=111@TIMx_CCMR1
    - 设置OC1的输出高电平有效：CC1P=0@TIMx_ECR
      - 在向上计数模式下，先是inactive，即低电平
    - 分别在TIMx_CCR1和TIMx_ARR中指定t_{DELAY}和t_{PULSE}
  - 置位OPM来在下一个UEV到来时关闭计数器
    - 使得只产生一个脉冲信号

单脉冲的配置



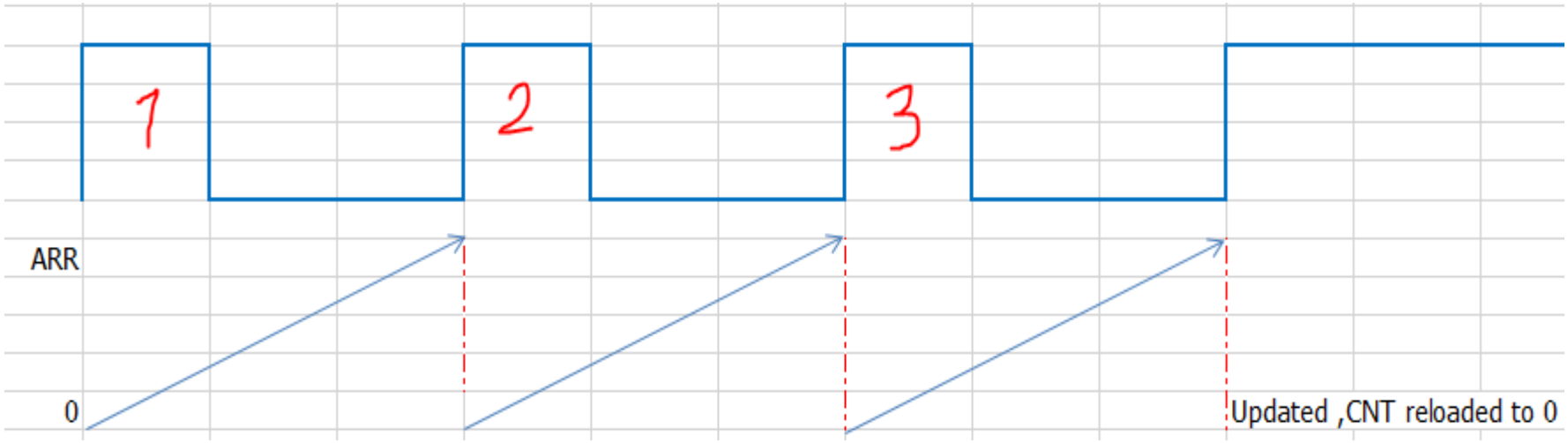
Prescaler (PSC - 16 bits value)	0
Counter Mode	Up
Counter Period (AutoReload Register - 16 bits value)	500
Internal Clock Division (CKD)	No Division
Slave Mode Controller	Trigger Mode
rigger Output (TRGO) Parameters	
Master/Slave Mode (MSM bit)	Disable (Trigger input effe
Trigger Event Selection	Reset (UG bit from TIMx_E
rigger	
Trigger Polarity	Rising Edge
Trigger Filter (4 bits value)	0
WM Generation Channel 1	
Mode	PWM mode 2
Pulse (16 bits value)	300
Fast Mode	Disable
CH Polarity	High

六、几个比较输出应用案例

- 指定个数脉冲的输出
- 灵活使用OC比较切换功能实现多种波形

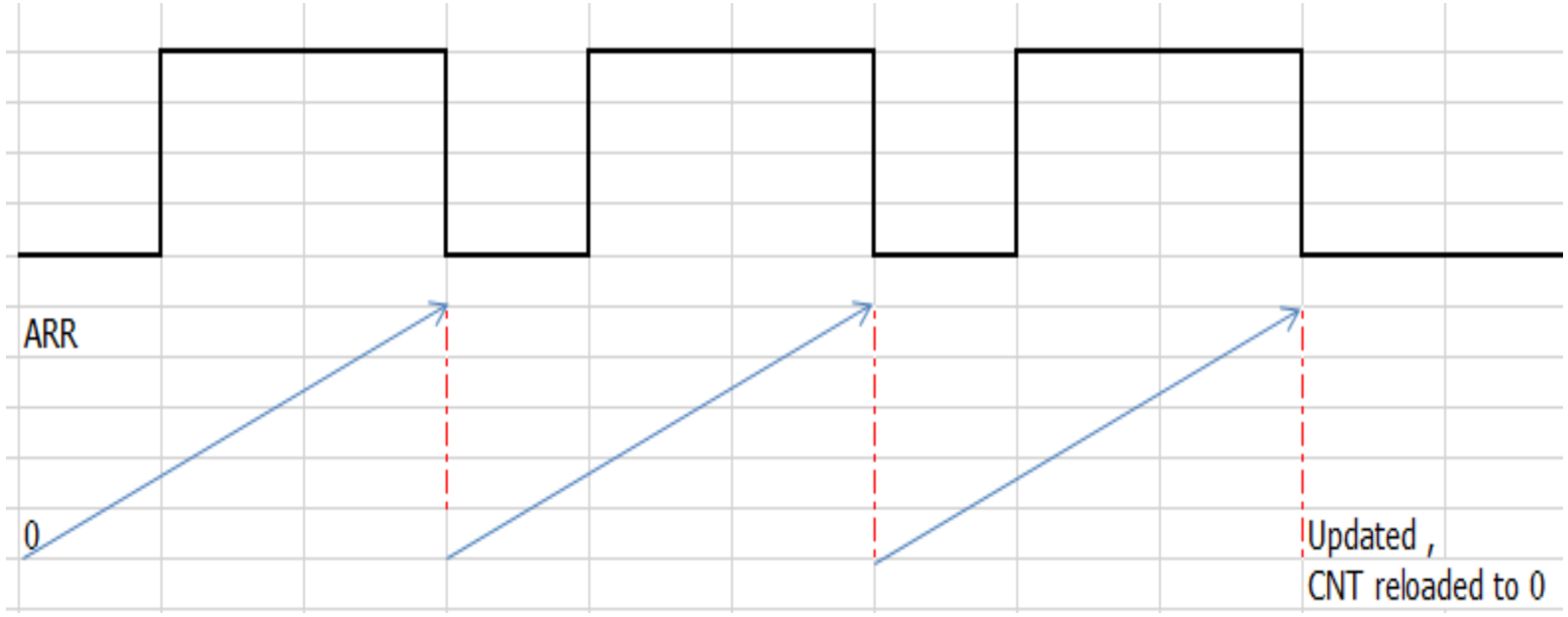
关于指定个数脉冲的实现【1】

- 一、通过高级定时器，使用单脉冲模式，借助RCR重复计数器实现
- 我们要选择合适的计数模式与PWM模式来满足需求
- 示例：UP counting + PWM1 RCR=2; 极性选择高有效



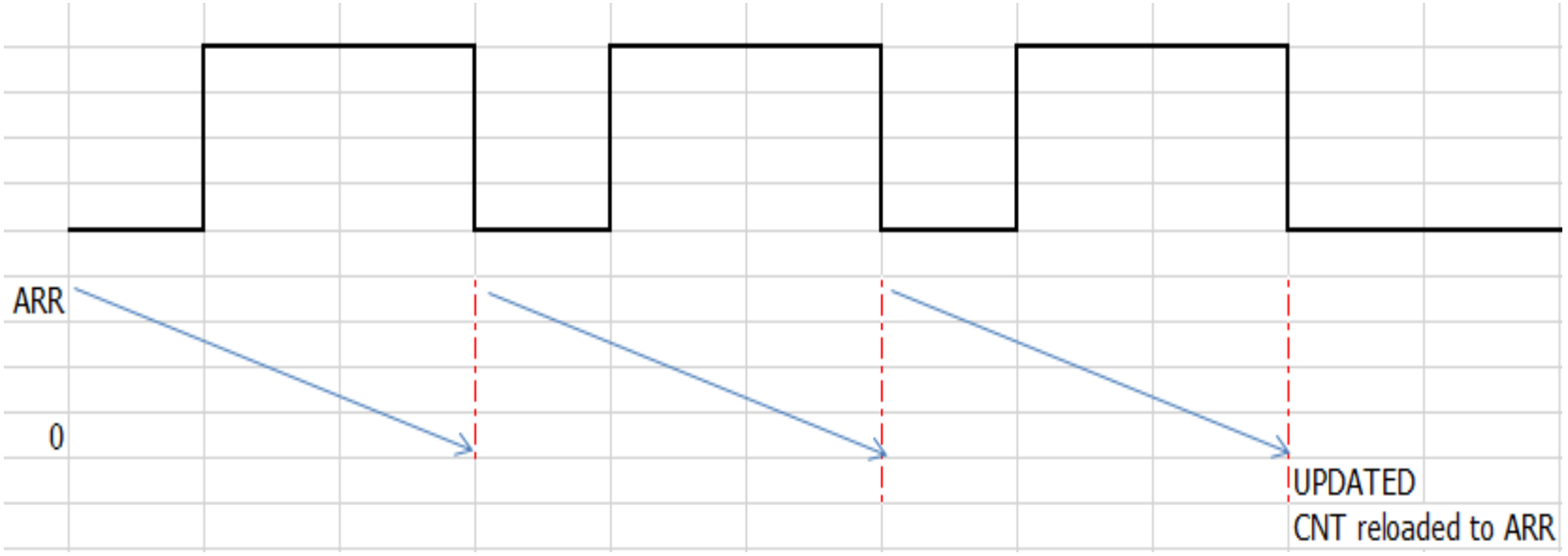
关于指定个数脉冲的实现【2】

- 示例：UP counting + PWM2 RCR=2; 极性选择高有效



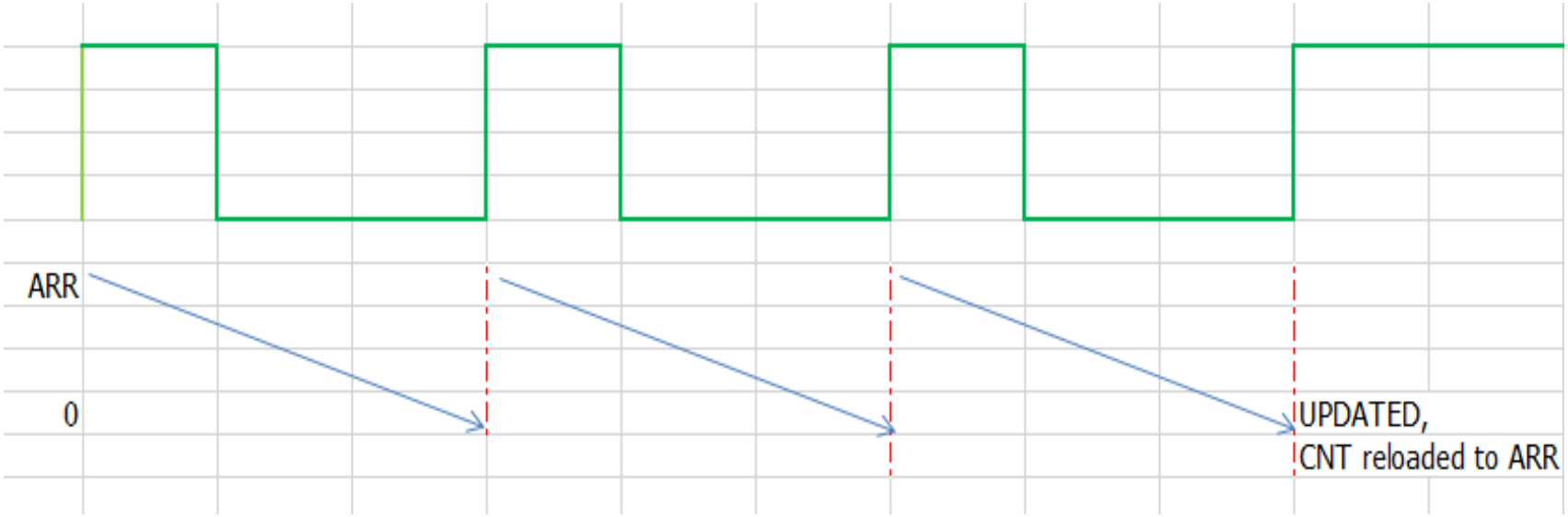
关于指定个数脉冲的实现【3】

- Down counting + PWM1 RCR=2; 极性选择高有效



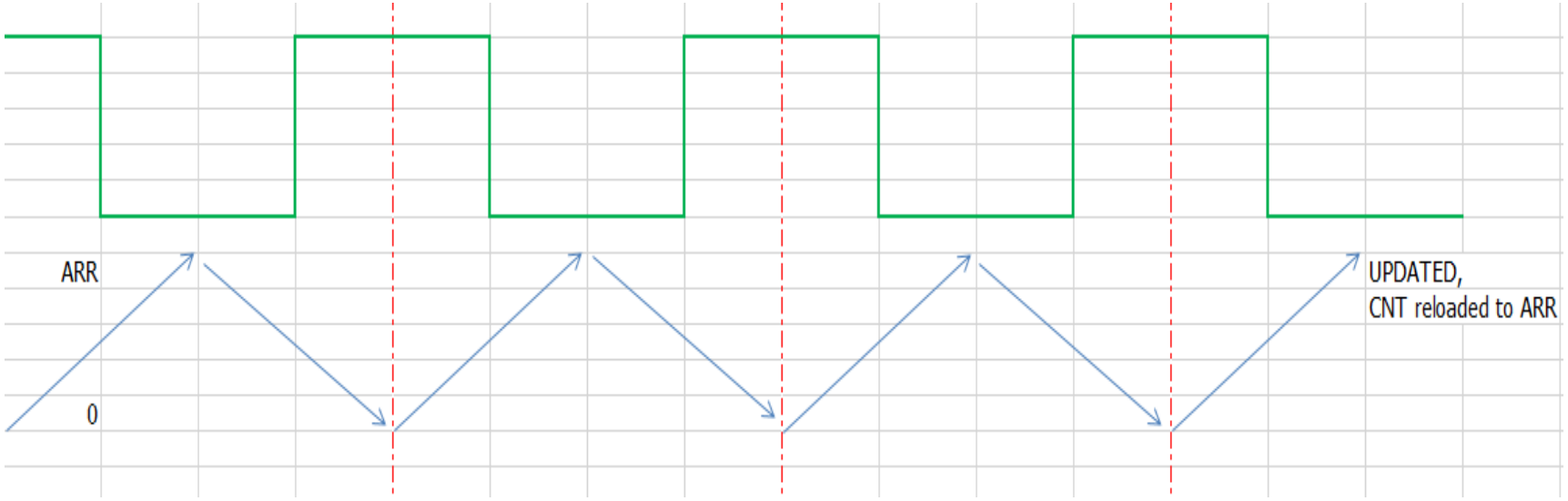
关于指定个数脉冲的实现【4】

- Down counting + PWM2, RCR=2; 极性选择高有效



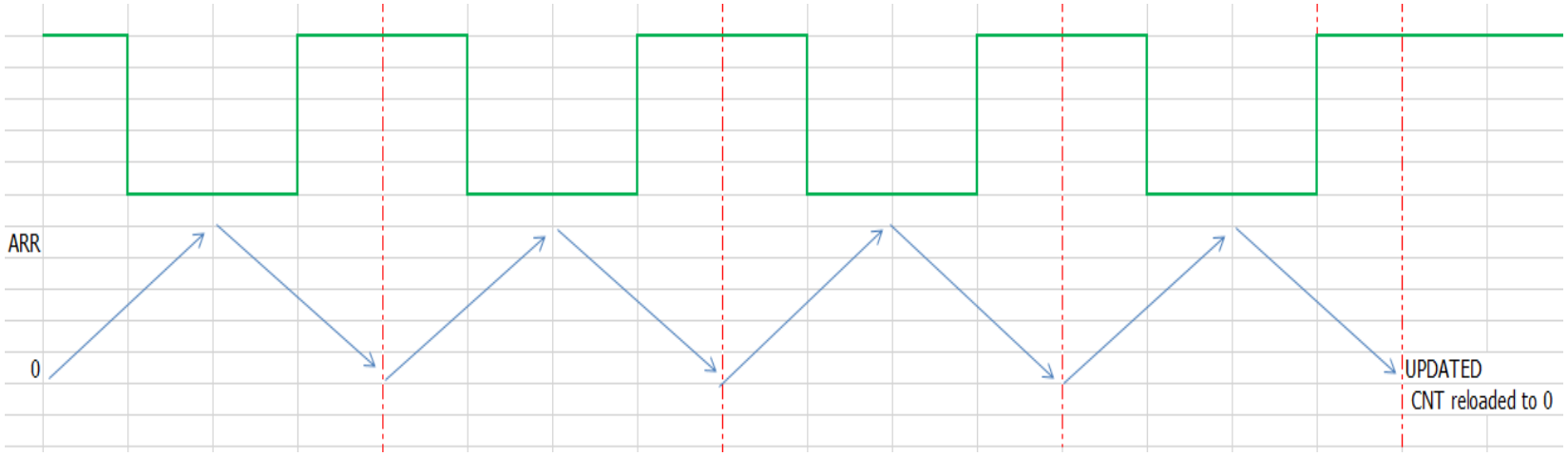
关于指定个数脉冲的实现【7】

- Center counting / pwm1 / RCR=6　极性选择：高有效



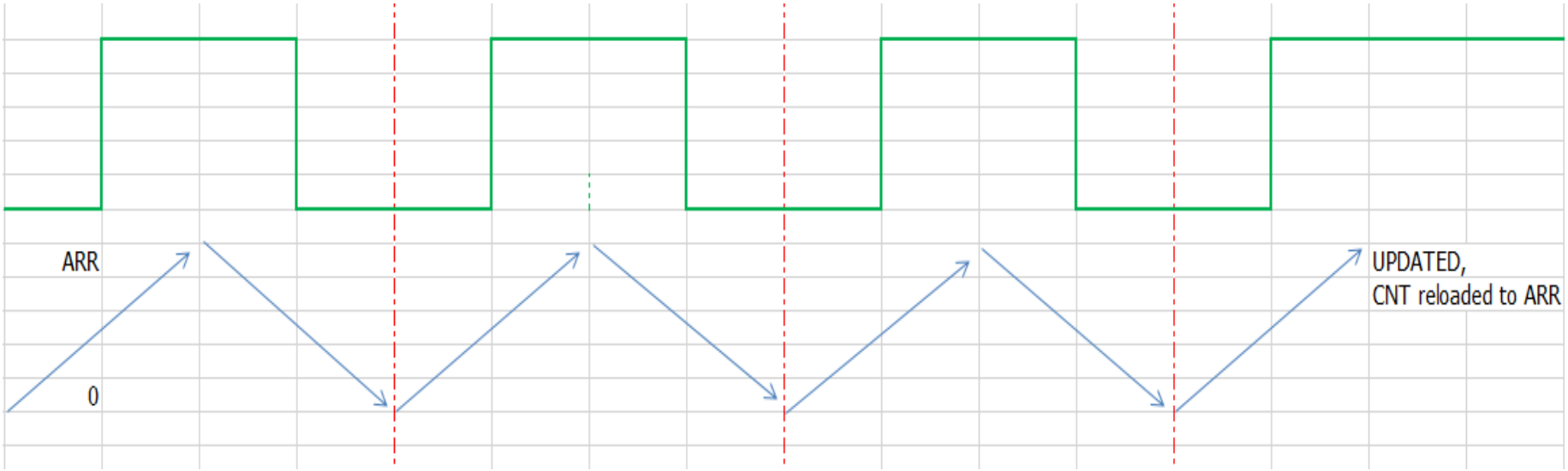
关于指定个数脉冲的实现【8】

- Center counting / pwm1 / RCR=7　极性选择：高有效



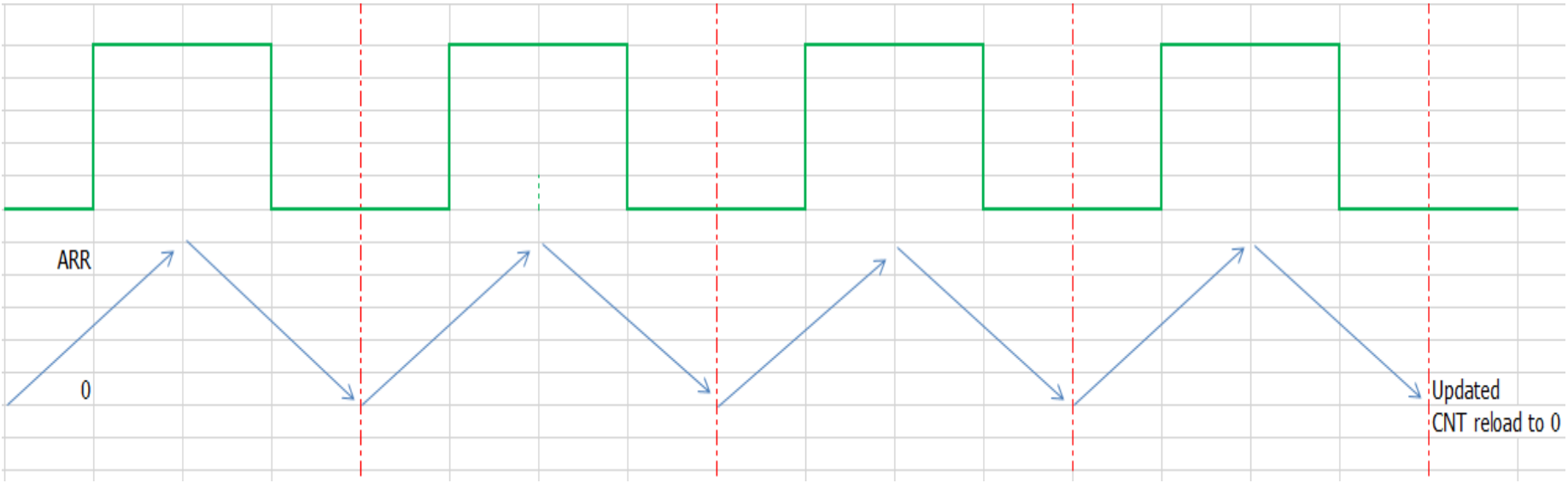
关于指定个数脉冲的实现【9】

- Center counting / pwm2/ RCR=6 极性选择：高有效



关于指定个数脉冲的实现【10】

- Center counting / pwm2 / RCR=7　极性选择：高有效



- 总之，通过RCR实现N个脉冲很方便。但不能修改脉冲占空比或频率。

基于通用定时器实现N个PWM输出【1】

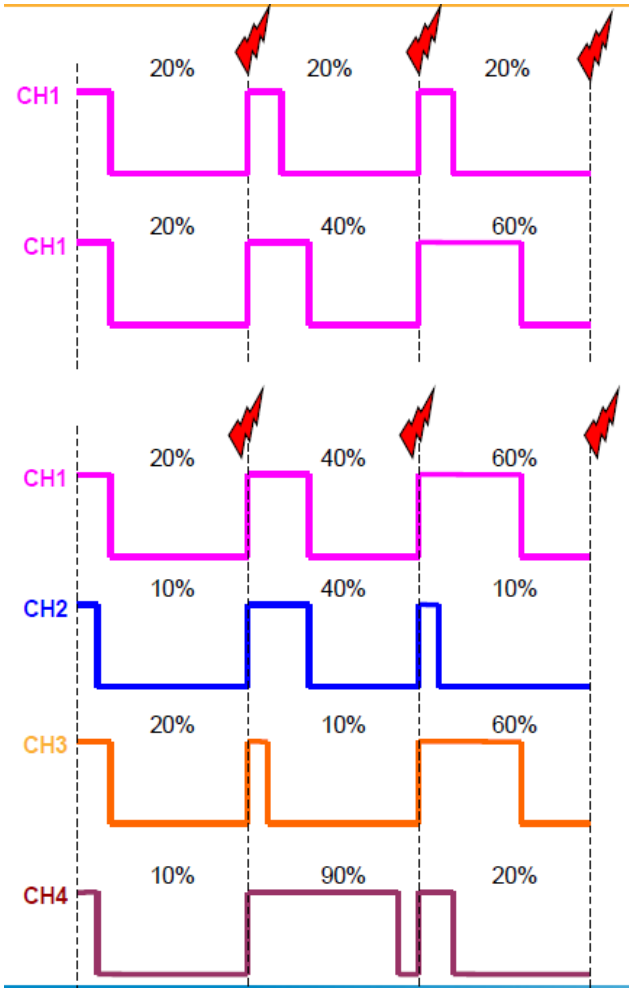
- 中断方式
- 我们可以在比较或更新中断里做脉冲个数计数，当到达指定个数时将ocx的电平固定在期望为电平。
- 例：pwm1模式,高有效；输出5个脉冲,脉冲个数计数器初始值为0
- 开启定时器比较中断或更新中断
- 当脉冲计数器计到达指定个数时，根据实际需求修改CCR的值：
  - 修改CCR=0，　　令输出为低电平；
  - 修改CCR=ARR+n [n不小于1】　　令输出为高电平
- 可以拓展功能，使用不同数据来修改脉冲的占空比或频率【修改ARR的值】

基于通用定时器实现N个PWM输出【2】

- DMA方式 来实现上述目的。pwm1,高有效；输出5个脉冲。
- 开启定时器比较事件的DMA请求【更新事件也可以】
- DMA每次传输CCR的数据过来，根据所需脉冲数设置末
- 尾CCR的值【如果修改CCR值，同时达到修改占空比目的】：
  - 修改CCR=0，　　令输出为低电平；
  - 修改CCR=ARR+n [n不小于1】　　令输出为高电平
- 同样，也可以拓展功能，修改脉冲的占空比或频率【修改ARR】
- 显然，相比RCR的实现更为灵活，但如果希望同时修改多个通道的寄存器参数，上面的方法实现起来还是有点力不从心。不妨看看定时器的批量传输。

七、定时器的批量传输

- 很多STM32定时器模块中有个专门针对定时器批量访问的传输方式。



使用定时器的PWM输出模式，设置ARR和CC1R来确定PWM波形的频率和占空比

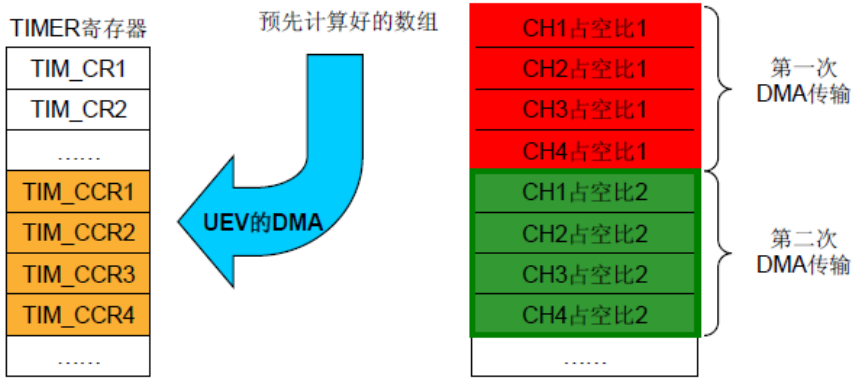
CC1R / ARR = 20%

占空比不断变化，可以在UEV事件触发的DMA中修改 CC1R数值，把预先计算并存储在SRAM数组中的占空比计算值传输到CC1R，进行更新

CC1R / ARR = 20%，40%，60% …

每个通道的占空比都在不断变换，如何在**UEV**事件触发的**DMA**中，同时修改多个比较输出寄存器的值**CC1R/CC2R/CC3R/CC4R**？

→ 定时器的**DMA**批量传输特性



定时器DMA批量传输寄存器DMAR、DCR

- 相比基于定时器计数器的非DMA批量传输，其配置并无太多差别。
- 这里涉及到2个寄存器TIMx_DMAR和TIMx_DCR。
- DMAR寄存器作为DMA批量传输时外设的目标或源地址。配置时将其自身地址给DMA控制器就好，无须用户对它做额外配置。【它专用于定时器的批量传输，DMA访问它时就知道要做定时器批量传输了】*
- 通过对DCR寄存器配置批量寄存器组连续访问时的第一个寄存器相对于定时器地址映射表中的初始寄存器的地址偏移量 以及每次批量访问的寄存器个数。 即DCR中需要2个数据，批量寄存器个数，首寄存器在定时器寄存器地址表中的偏移量。不妨看个实例：
- 假设我们要对TIM1的CCR1~CCR4四个寄存器做DMA批量访问，看看DCR怎么配置。

定时器地址映射表局部图

Table 96. TIM1 and TIM8 register ma

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0x00	TIMx_CR1																
0	Reset value																
0x04	TIMx_CR2																
1	Reset value																
0x08	TIMx_SMCR																
2	Reset value																
0x0C	TIMx_DIER																
3	Reset value																
0x10	TIMx_SR																
4	Reset value																
0x14	TIMx_EGR																
5	Reset value																
	TIMx_CCMR1																

Offset	Register	31	30	29	28	27	26	25
0x2C	TIMx_ARR							
11	Reset value							
0x30	TIMx_RCR							
12	Reset value							
0x34	TIMx_CCR1							
13	Reset value							
0x38	TIMx_CCR2							
14	Reset value							
0x3C	TIMx_CCR3							
15	Reset value							
0x40	TIMx_CCR4							
16	Reset value							

关于批量传输寄存器DCR的配置

TIM1 and TIM8 DMA control register (TIMx_DCR)

Address offset: 0x48

Reset value: 0x0000

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

- DBA：被访问的第一个寄存器的地址偏移量【位于定时器地址映射表】；DBL：批量访问的寄存器个数。【从0开始算】
- 结合上图可得知TIM_CCR1位于寄存器地址映射表中的第14号位置，则DBA=14-1； 另外，用于批量访问的寄存器数为4个，则DBL=4-1 ；那么DMA访问DMAR寄存器就是按照下面地址分次访问：
- (TIMx_CR1 address) + (DBA + DMA index) x 4
- 【Index是DMA批量访问时硬件自动生成的动态索引号，按0~DBL依次访问实现批量访问传输】
- 简而言之，我们用户要配置就是DBA/DBL两个数据。数数表格即可完成。

基于Cube库的定时器BURST传输函数

- 基于上述例程，相关用户调用及配置如下：【非常直观简洁】
- HAL_TIM_DMABurst_WriteStart(&htim1, TIM_DMABASE_CCR1, TIM_DMA_UPDATE,(uint32_t*)aSRC_Buffer, TIM_DMABURSTLENGTH_4TRANSFERS);
-

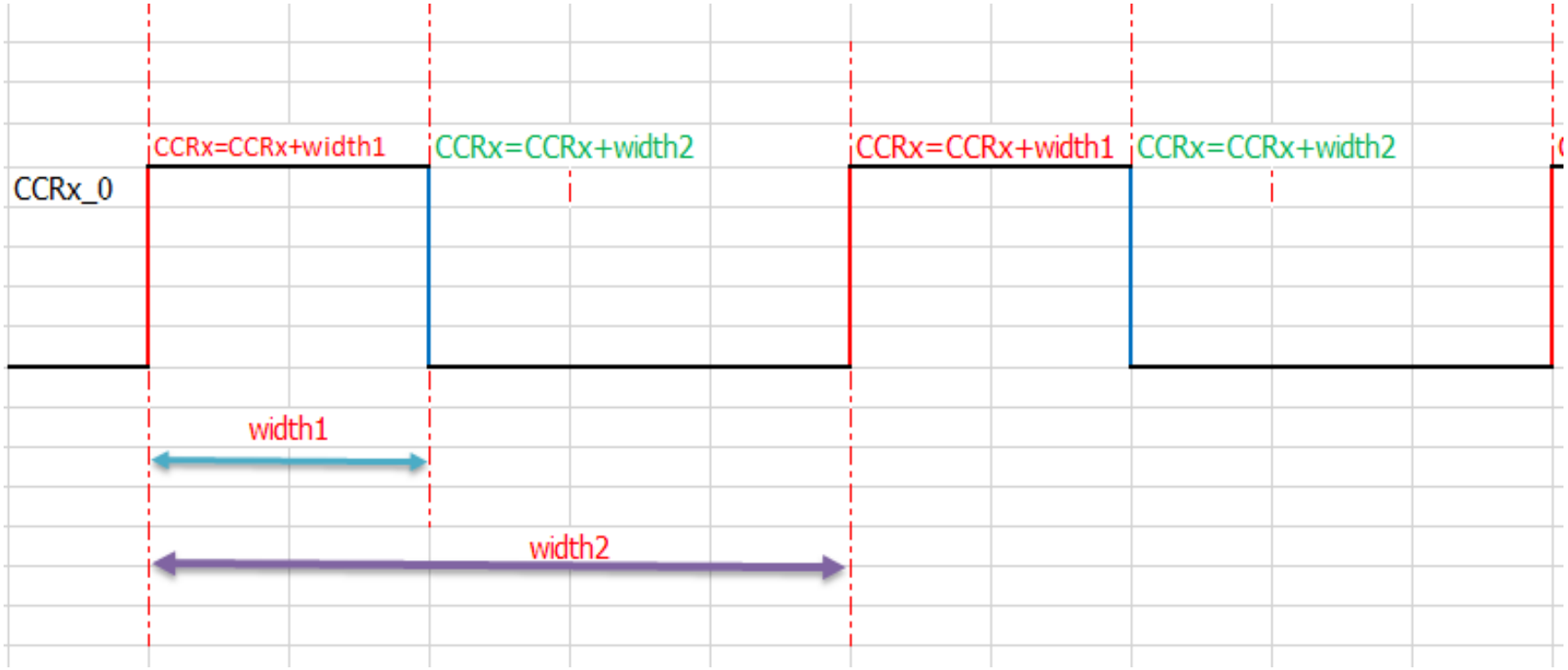
```
/* TIM1 DMA Init */
/* TIM1_CH2 Init */
hdma_tim1_ch2.Instance = DMA2_Stream2;
hdma_tim1_ch2.Init.Channel = DMA_CHANNEL_6;
hdma_tim1_ch2.Init.Direction = DMA_MEMORY_TO_PERIPH;
hdma_tim1_ch2.Init.PeriphInc = DMA_PINC_DISABLE;
hdma_tim1_ch2.Init.MemInc = DMA_MINC_ENABLE;
hdma_tim1_ch2.Init.PeriphDataAlignment = DMA_PDATAALIGN_WORD;
hdma_tim1_ch2.Init.MemDataAlignment = DMA_MDATAALIGN_WORD;
hdma_tim1_ch2.Init.Mode = DMA_NORMAL;
hdma_tim1_ch2.Init.Priority = DMA_PRIORITY_LOW;
hdma_tim1_ch2.Init.FIFOMode = DMA_FIFOMODE_DISABLE;
if (HAL_DMA_Init(&hdma_tim1_ch2) != HAL_OK)
{
  _Error_Handler(__FILE__, __LINE__);
}
__HAL_LINKDMA(htim_base,hdma[TIM_DMA_ID_CC2],hdma_tim1_ch2);
```

定时器BURST传输的小结

- 1、定时器BURST传输不是所有定时器支持；
- 2、相比基于定时器寄存器的非批量传输，设置上仅多了个基于位置偏移量和寄存器个数的配置的DCR寄存器，DMAR寄存器完全不用配置；
- 3、定时器BURST传输限于同一定时器内的寄存器、且只能做地址连续的寄存器访问，不可跳跃；

输出比较切换模式的应用案例【1】

- 比较输出模式中有个比较切换模式，即当CCR值与CNT值相等时做输出的电平翻转切换。 基于这个原理，我们可以利用该模式灵活输出多种格式的信号。**基本原理 CCR_next=CCR_current+Width （*）**

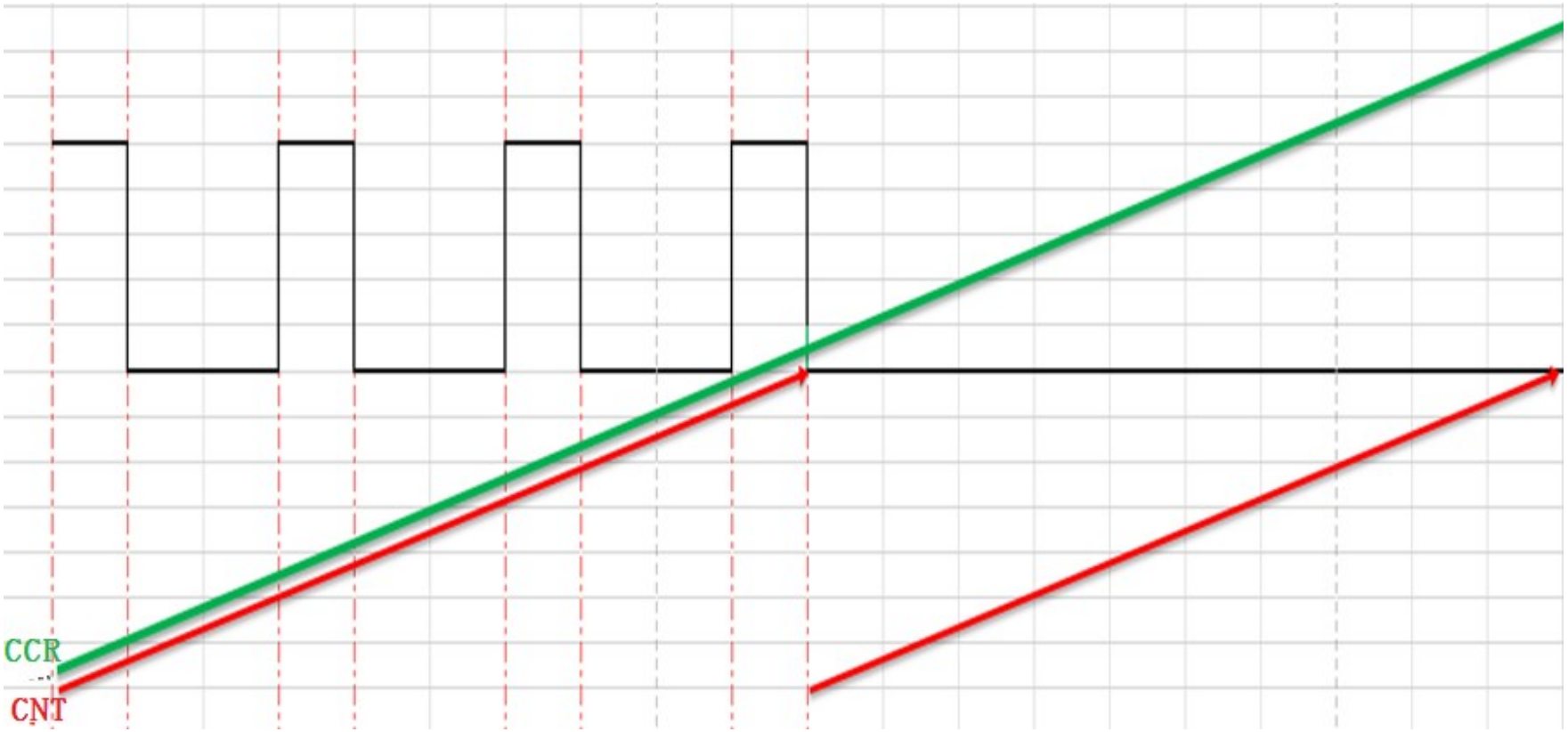


输出比较切换模式的应用案例【2】

- 利用比较切换输出模式来实现PWM输出时，要注意几点：
- 1、关闭CCR寄存器的预装载功能；即OCxPE=0;
- 2、我们一般使用单向向上计数模式；【主要是为了方便】
- 2、关于定时器的时基参数ARR的值建议使用满量程值，因为我们目前
- 每次读取计数器的值，加上相应宽度数据后赋值给CCR寄存器，然后让CCR跟后续的计数器数字做比较。如果ARR不使用计数器的满量程值，意味着它不许记到计数器自身宽度满量程时会发生溢出重装，由于CCR的值在不停增加，那当CCR的赋值超过ARR的值而又CCR又没有发生溢出的那一段时间内，是不会发生比较切换输出的。因为那段时间内CCR值就不可能会等于计数器的值。
- 不妨来个图示意下。假设16位定时器TIM3。设定ARR=0xfff 【3个F】

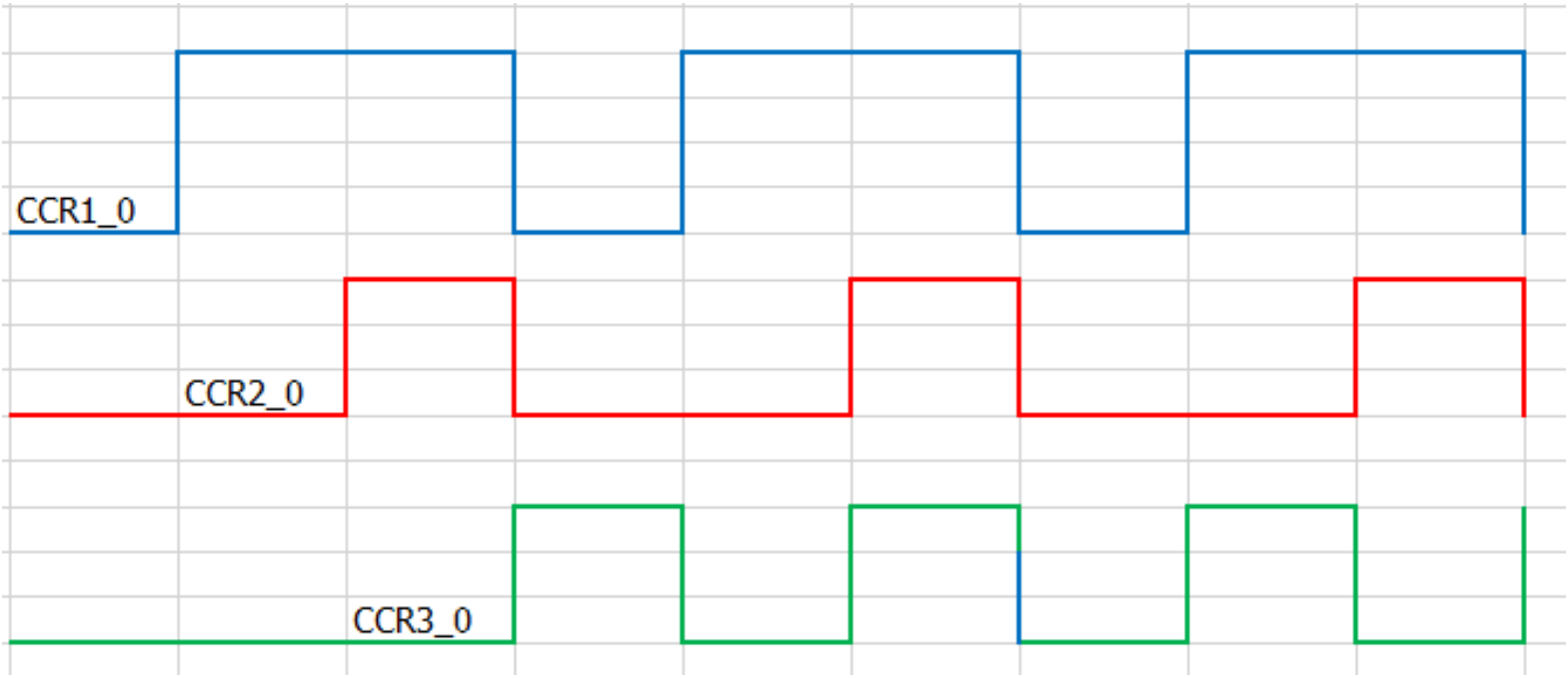
输出比较切换模式的应用案例【3】

- 因为ARR=0xfff,而 CCR最大可以累积到0xffff,当CCR的值超出ARR的值时，它还可以继续往前累加，但计数器的值只能在0~ARR之间变动，在CCR的值发生溢出前，二者是不可能有相等的机会的。



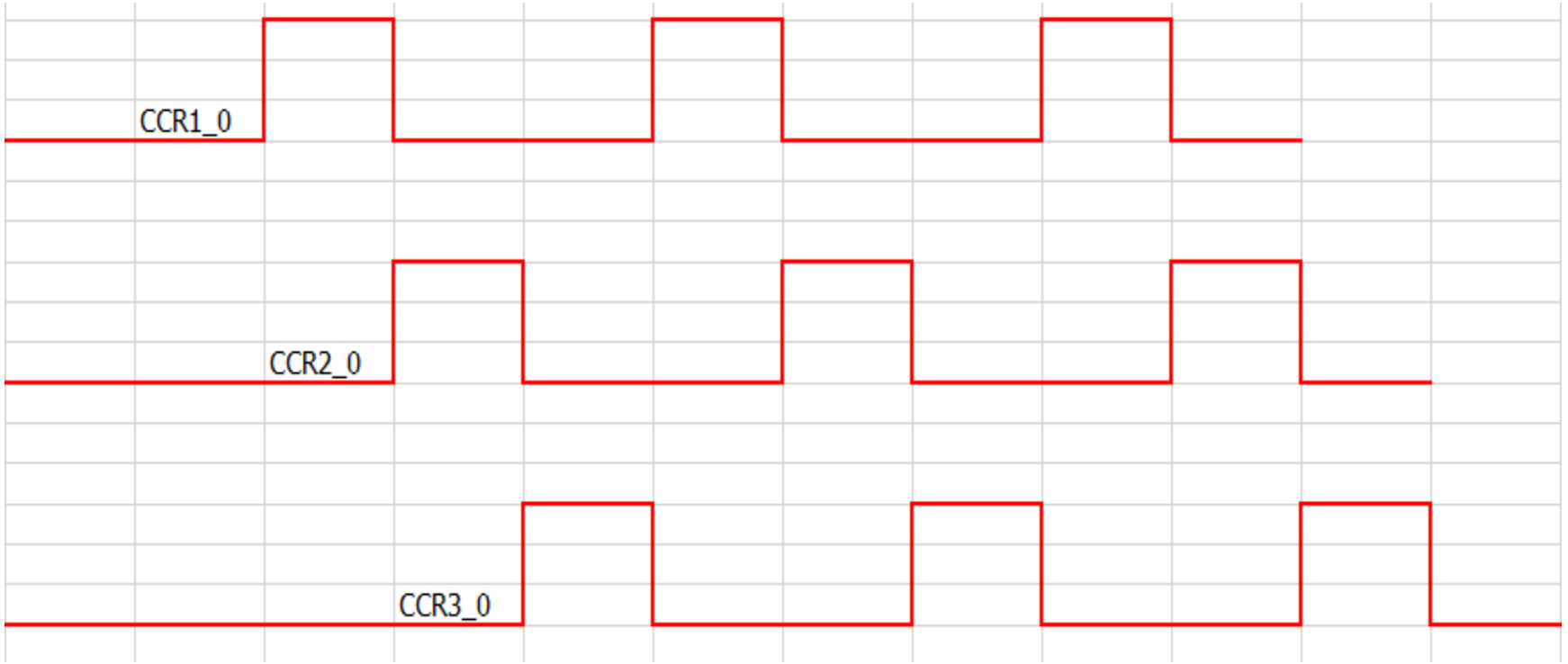
输出比较切换模式的应用案例【4】

- 实现多路不同频率与不同占空比的波形



输出比较切换模式的应用案例【3】

- 实现多路固定相位差的相同波形

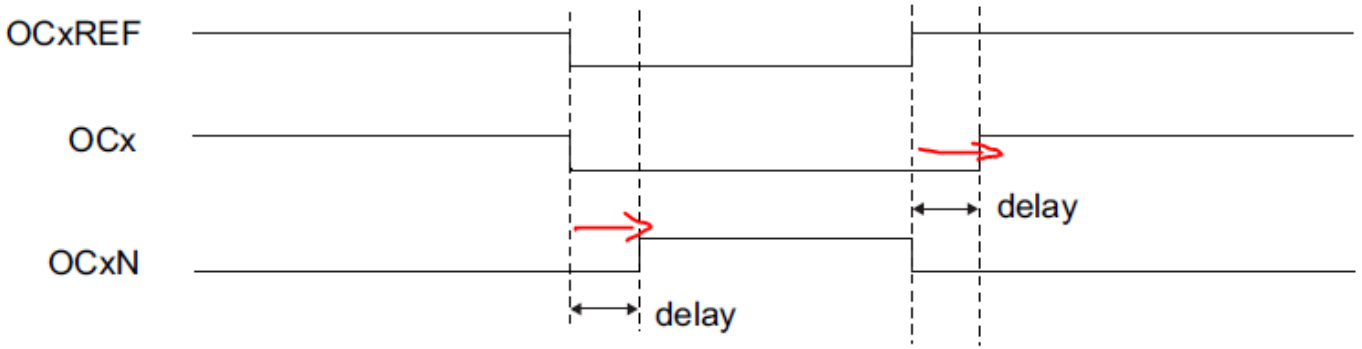


输出比较切换模式的应用案例【4】

- 实现带死区的互补波形

比较输出应用时的几个注意点（1）

- 1、高级定时器相比通用定时器，它增加了主输出使能控制位，【MOE@timx_bdtr】,如果该位置零的话，此时OCx端没有波形输出；
- 2、高级定时器相比通用定时器，它增加了主输刹车控制机制，BKE@timx_bdtr】,如果使能了刹车控制并触发有效刹车电平，此时OCx端没有波形输出；
- 3、高级定时器相比通用定时器，使用互补输出时增加了死区插入机制，【DTG@timx_bdtr】,若插入的死区时间过大会导致OCx/OCxN一端或两端没有波形输出；



比较输出应用时的几个注意点（2）

- 4、高级定时器的互补输出在OCxREF出来后、在极性选择之前，二者是互补的，经过极性选择后是否互补取决于两互补通道的极性选择。极性的选择最终由实际驱动电路需求决定的。

- 5、定时器的所有输出通道可以独立自由设置，如禁用/开启、输出模式、极性安排等。但高级定时器的做互补输出时，他们共用相同输出模式和CCR值。

- 6、对于通用定时器的Ocx通道，当使能该通道时，Ocx输出=OCxREF+极性；当禁用该通道时，Ocx的输出=0。

- 7、对于高级定时器Ocx/OCxN互补通道，没法实现同时实现有效电平的输出。区分有效电平、无效电平与最终输出端的高低电平。

- 8、对于高级定时器来说，Ocx/OCxN的输出除了跟输出使能位有关外，跟多个控制位【MOE/OSSI/OSSR/CCxP/CCxNP】有关。手册中有个Ocx/OcxN互补通道输出控制表格可以查看。

高级定时器比较输出时的几种状态

- 1、 output disable【输出禁止】:OCx或OCxN输出为0，此时

  对应的输出使能【CCxE/CCxNE】被禁止，输出不受定时器控制；

- 2、off-state 【关闭状态】:OCx或OCxN输出无效状态或电平。

- 3、idle state【空闲状态】/run state 【运行状态】：

  MOE=0 ==》空闲状态 / MOE=1 ==》运行状态

4、idle电平：在Idle状态下，由OISx位决定Ocx/OCxN的电平。

案例：停止PWM输出时的管脚电平问题

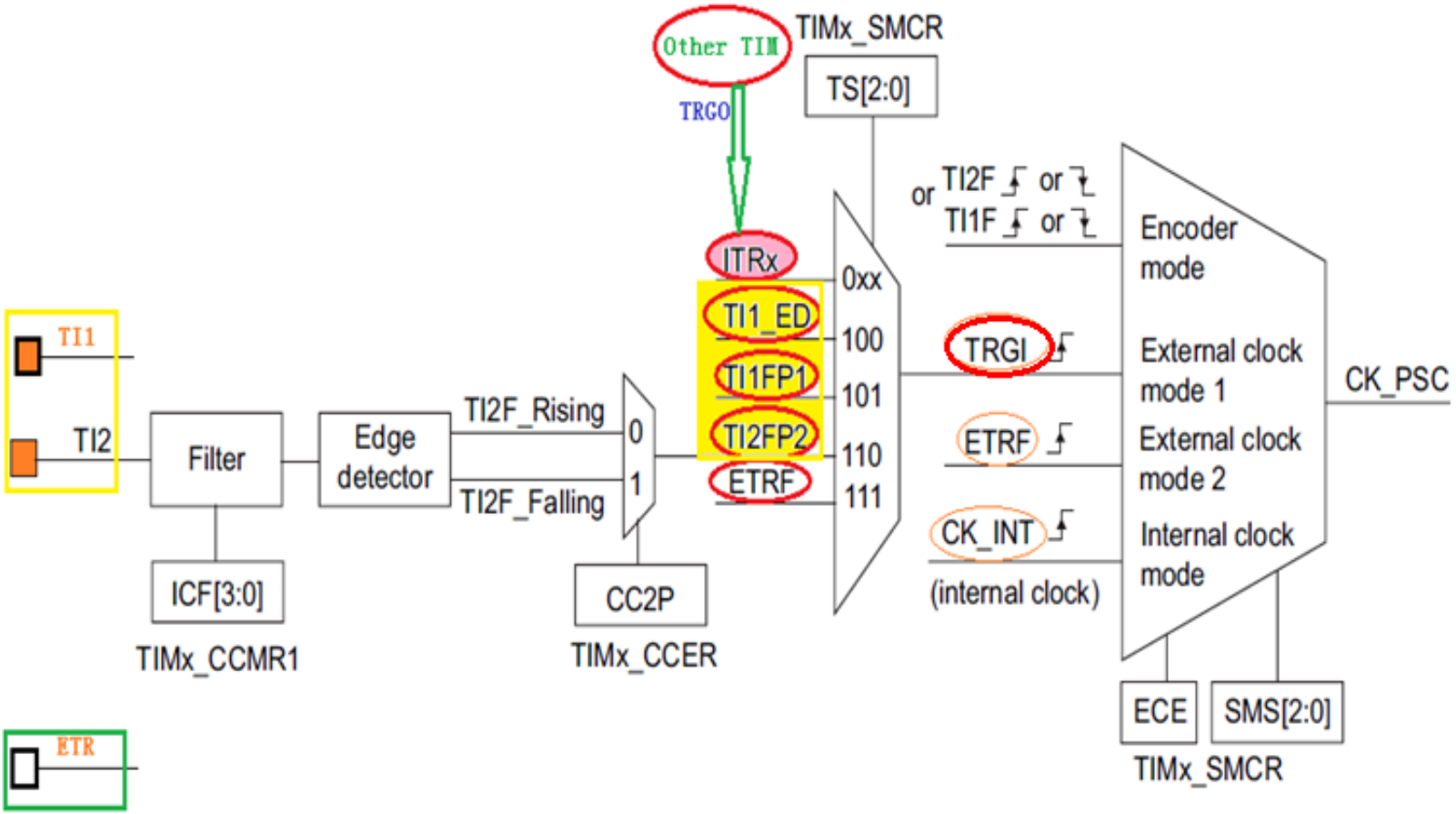
- 现象：经常有人在使用TIMER做PWM输出时，发现关闭定时器停止PWM输出时OCx端电平不定，觉得很困扰。
- 原因：暂停PWM时的时机不定，导致CCR与CNT的比较结果不定进而使得停止PWM时刻的电平不定，时高时低。
- 两个参考办法：
- 1、通过修改CCR=0或大于ARR值来实现输出固定电平。
- 这个过程中要综合考虑PWM模式、极性选择及你期望的电平。
- 2、修改OC模式，强制输出有效或无效Ocxref电平，配合极性选择
- 实现你的目的。至于是否开启ARR/CCR的预装载，做具体分析】

八、定时器主从触发与同步

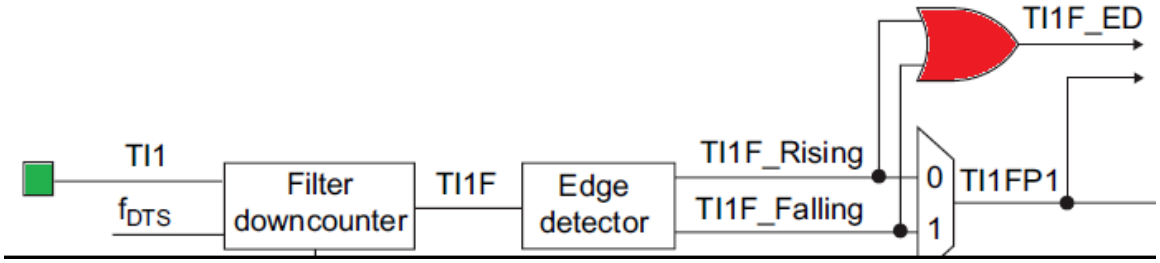
- 定时器的同步触发与级联包含两个层面的内容：
- 1、定时器与触发信号的触发同步
- 此时定时器被触发输入信号【TRGI】触发并工作在从模式，有复位、触发、门控三种基本的从模式。当触发信号作为从模式定时器的时钟源时，定时器工作在外部时钟1从模式。
- 2、定时器之间的同步触发与级联
- 定时器之间通过触发信号[TRGO/TRGI]进行同步触发或分频级联。【实质是定时器从模式的具体应用】
- 定时器间形成主从关系，主定时器可以对从模式定时器进行复位、开启、停止或提供时钟。
- 3、要弄清定时器从模式与主从触发关系，首先得知晓有哪些触发信号，TRGI/TRGO, 从哪里来？去到哪里？

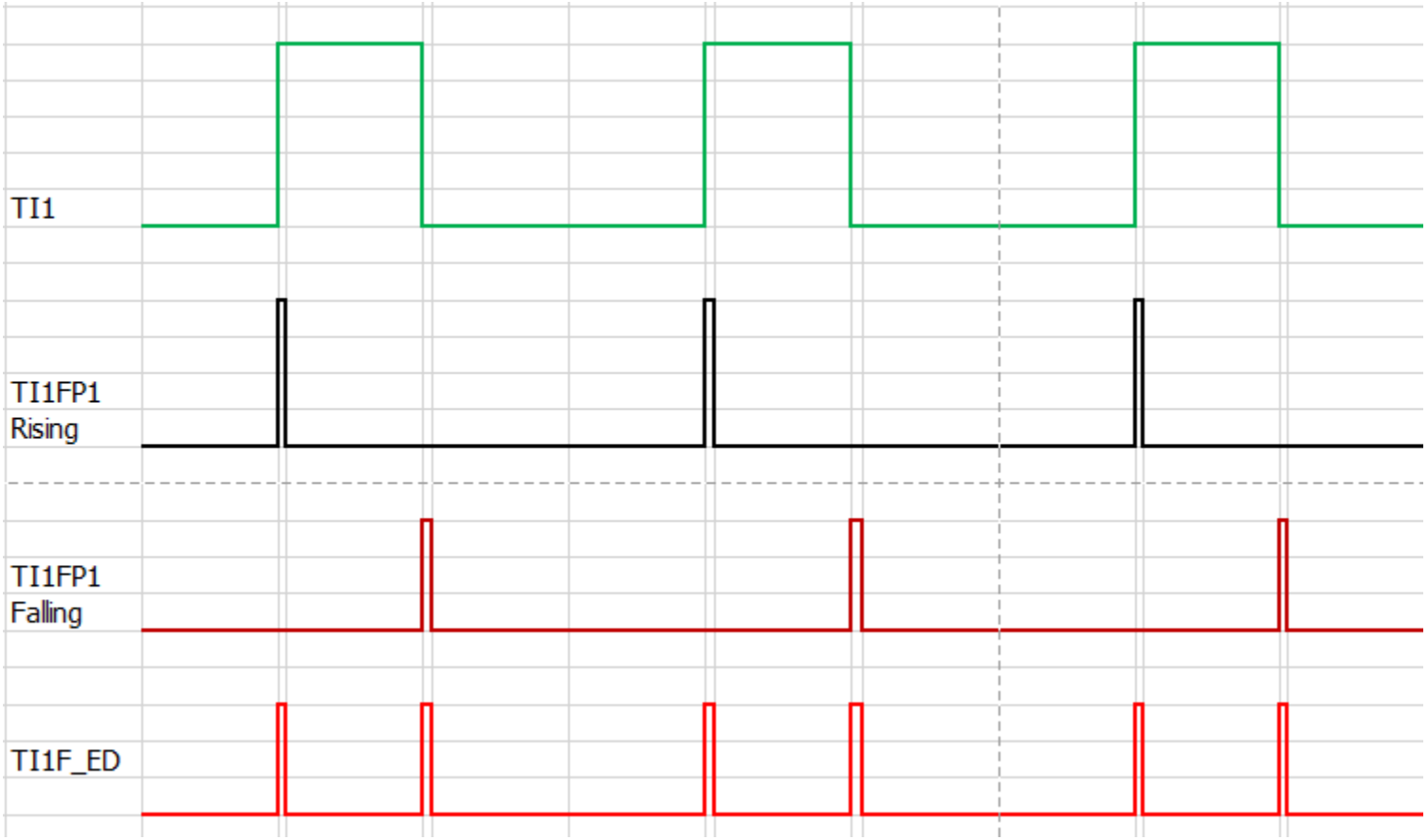
定时器内部触发信号TRGI有哪些

- 三类：来自TI1/TI2的信号 来自ETR脚的信号 来自其它TIMER的输出



TI1、TI1FP1、TI1FP_ED关系示意图





life.augmented

从模式定时器的触发信号选择

- 作为从定时器的触发输入有多种。内、外部信号的有效边沿连到TRGI可以作为定时器的同步信号或时钟。

TS @ TIMx_SMCR 触发选择信号	000~011	内部触发信号0/1/2/3：ITR0/1/2/3
	100	TI1边沿检测信号：TI1F_ED
	101	定时器输入1的滤波信号：TI1FP1
	110	定时器输入2的滤波信号：TI2FP2
	111	外部触发输入：ETRF

定时器TRGO 及产生

- 主定时器通过如下方式产生TRGO信号给到其它从定时器或其它外设：
- **1、定时器复位：置位UG@TIMx_EGR**
- **2、使能计数器。置位CEN@TIMx_CR1**
- **3、定时器更新事件**
- **4、定时器捕获、比较事件**
- **5、各输出通道中间参考信号 OCxREF**

TRGO

- 这些TRGO将通过内部线路连接到其它定时器作为从定时器的触发输入信号TRGI,或者作为其它外设控制触发信号，如触发ADC/DAC等。

定时器主模式的配置

- 当选择定时器作为主定时器时，其相应的触发输出信号可用作从定时器内部触发信号或时钟。
- 配置为主模式定时器的几个基本流程：
- 1、定时器基本时基参数配置。【ARR/PSC/…】
- 2、通过 CR2 寄存器中的MMS【2:0】（主模式选择）位来选择要使用的触发输出TRGO。
- 3、使能 SMCR 寄存器中的 MSM（主/ 从模式）位以实现当前定时器与其它从定时器的完美同步（通过 TRGO 实现）。

定时器从模式选择配置

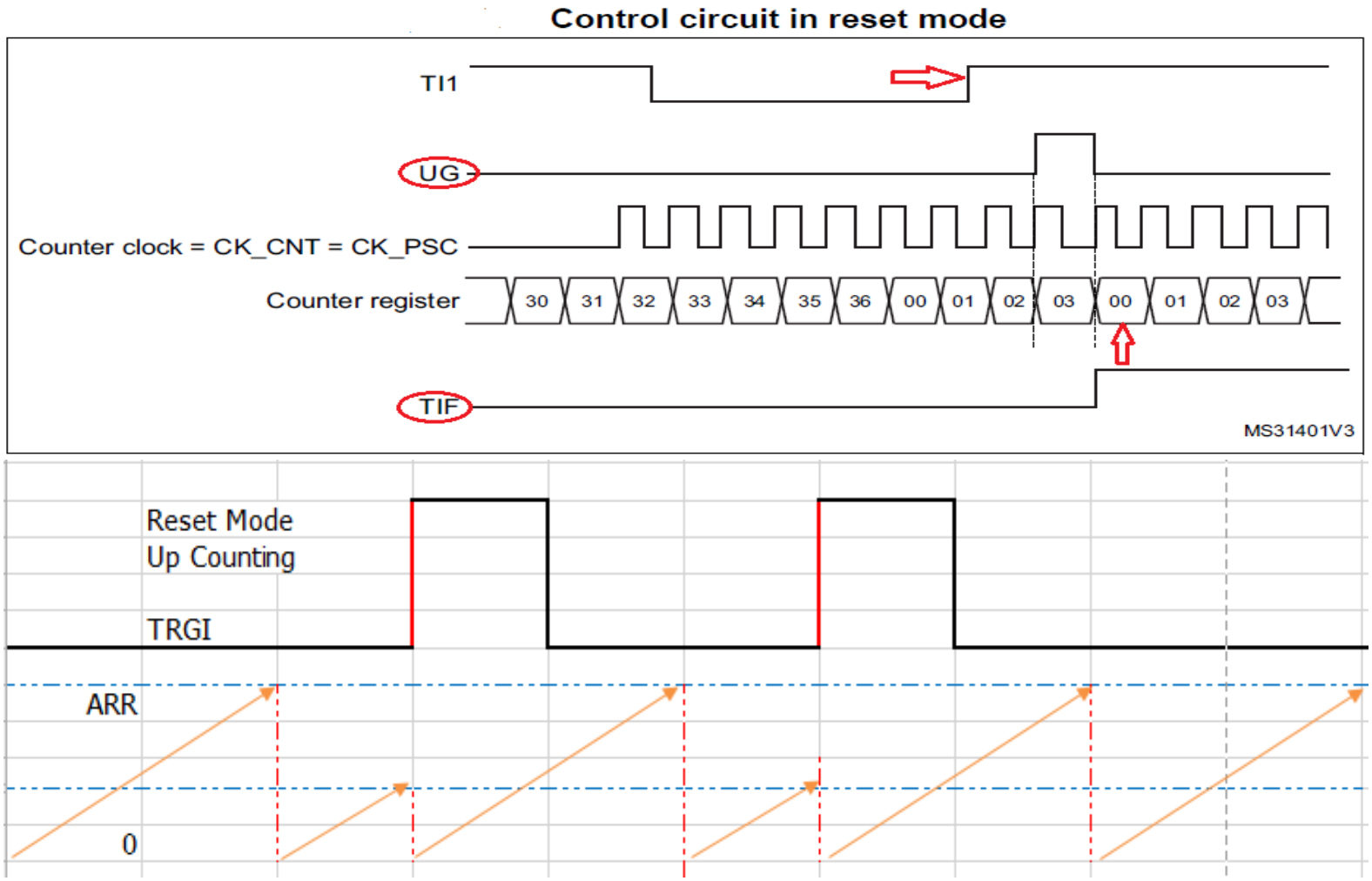
- 定时器收到TRGI信号时，可以工作在不同从模式下：

SMS @ TIMx_SMCR 从模式选择	000	从模式关闭：只要计数器使能(CEN=1)，内部时钟的分频信号给计数器提供时钟
	001~011	编码模式1/2/3
	100	复位模式：TRGI上升沿复位计数器，并产生UEV（若URS复位）
	101	门控模式：TRGI高电平计数器时钟运行；TRGI低电平计数器时钟停止，但不复位
	110	触发模式：TRGI上升沿启动计数器的运行，并不复位它
	111	外部时钟模式1：TRGI上升沿作为计数器的时钟

- 从定时器配置必要流程：

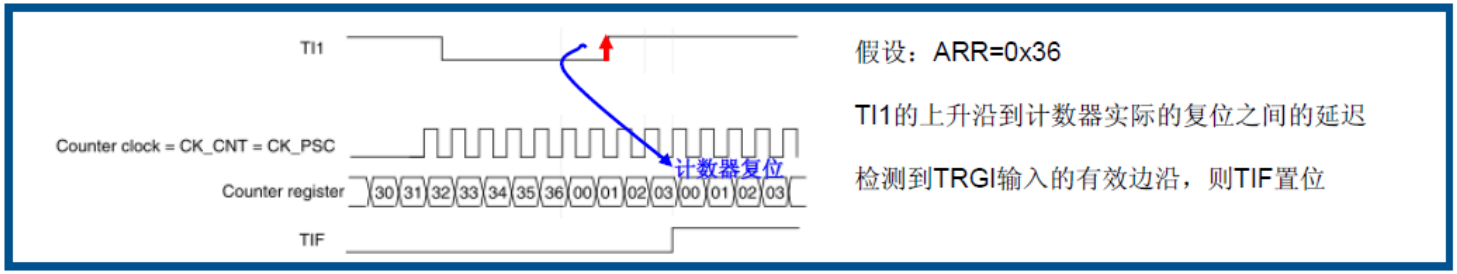
1、操作SMCR 寄存器中的 SMS（从模式选择）位以选择要使用的从模式。
2、操作SMCR 寄存器中的 TS（触发选择）位以选择要使用的内部触发。

从模式之一：复位模式【1】

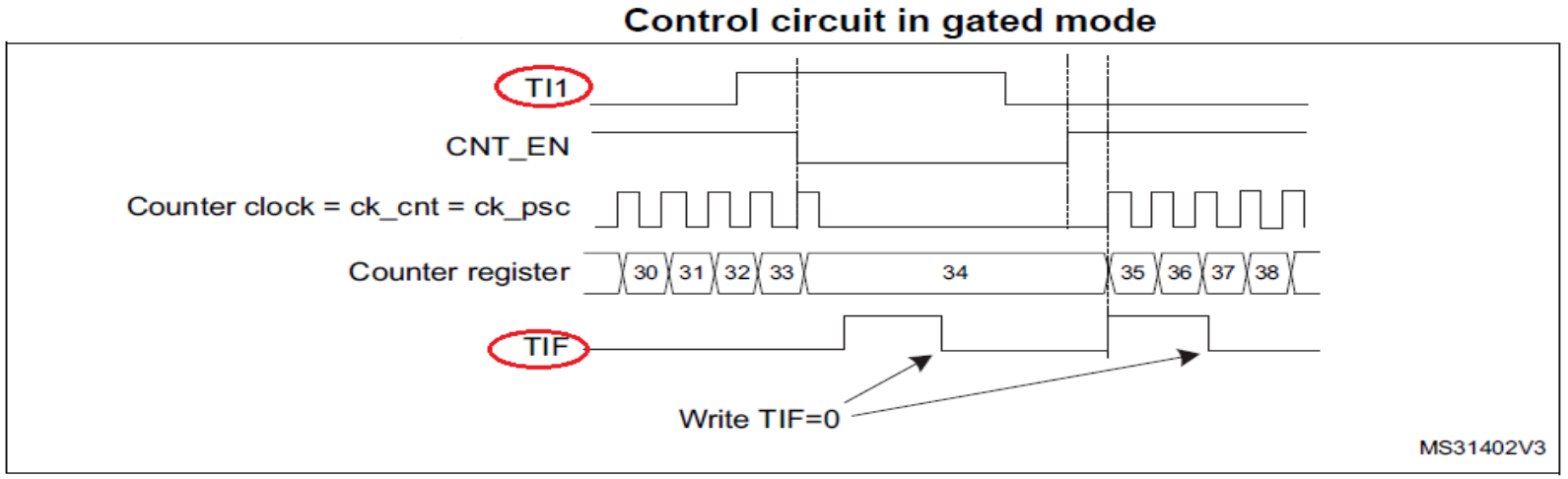


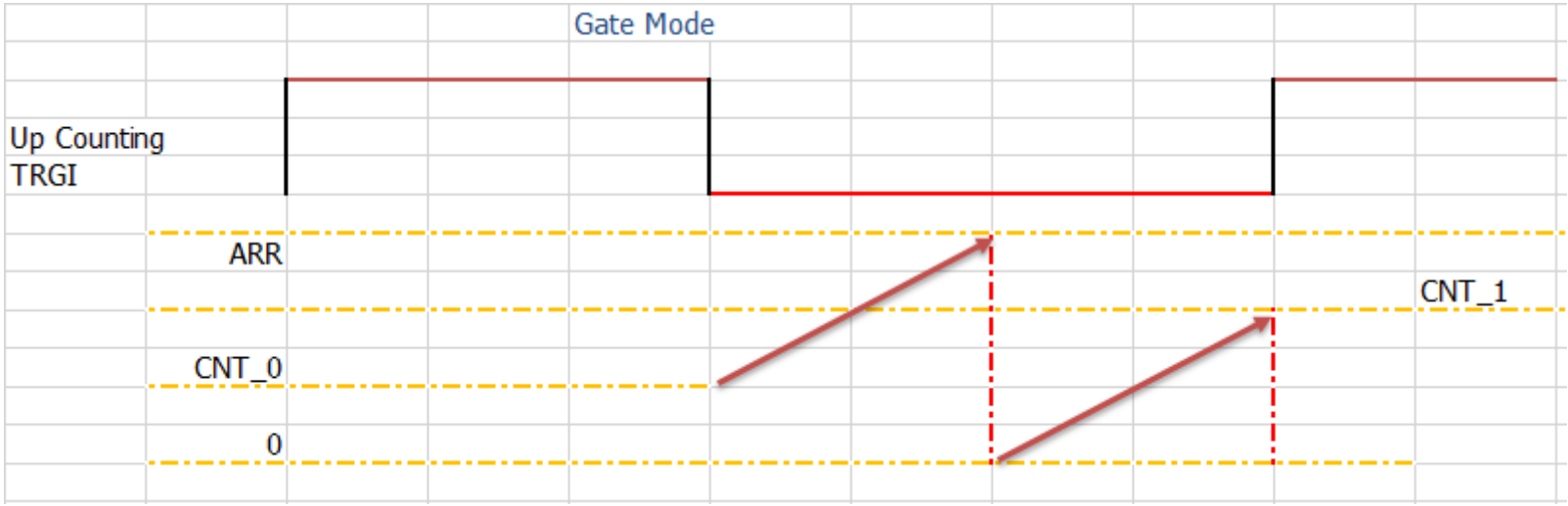
从模式之一：复位模式【2】

- 工作原理：TRGI上升沿复位计数器和分频器
  - 如果URS复位，则还会产生UEV
- 举例：TI1上升沿复位计数器
  - 检测TI1的上升沿
    - CC1S=01@TIMx_CCMR1 CC1P=CC1NP=0@TIMx_CCER
  - 定时器工作在复位模式，配置TRGI触发源
    - TS=101(TI1FP1) SMS=100@TIMx_SMCR
  - 使能计数器
    - CEN=1@TIMx_CR1



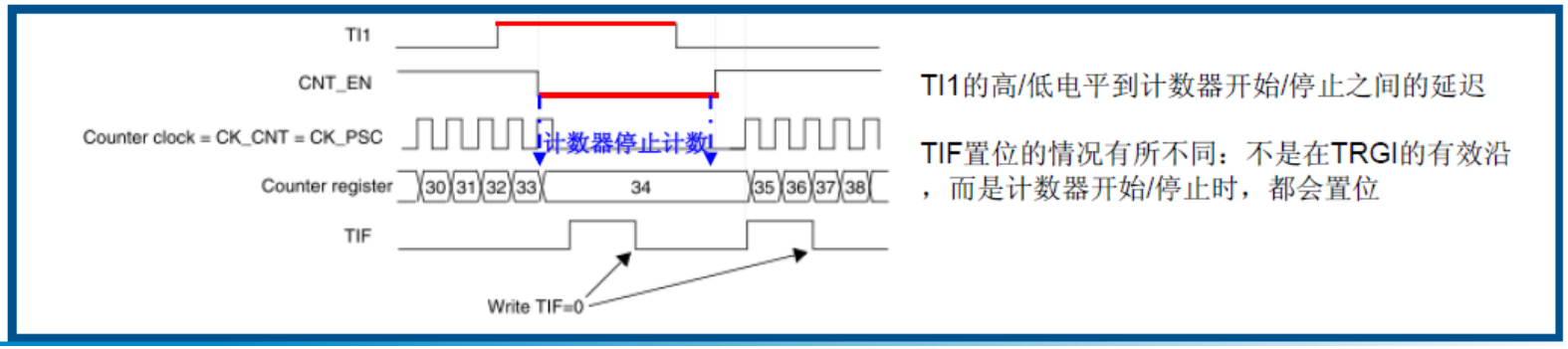
从模式之二：门控模式【1】





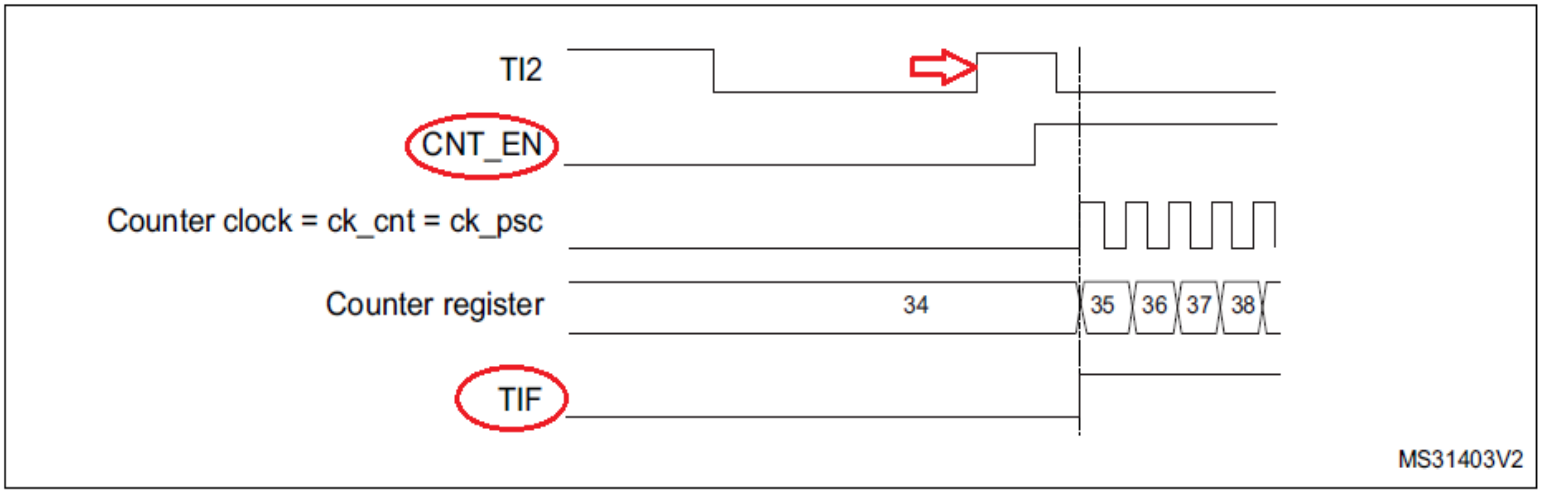
从模式之二：门控模式【2】

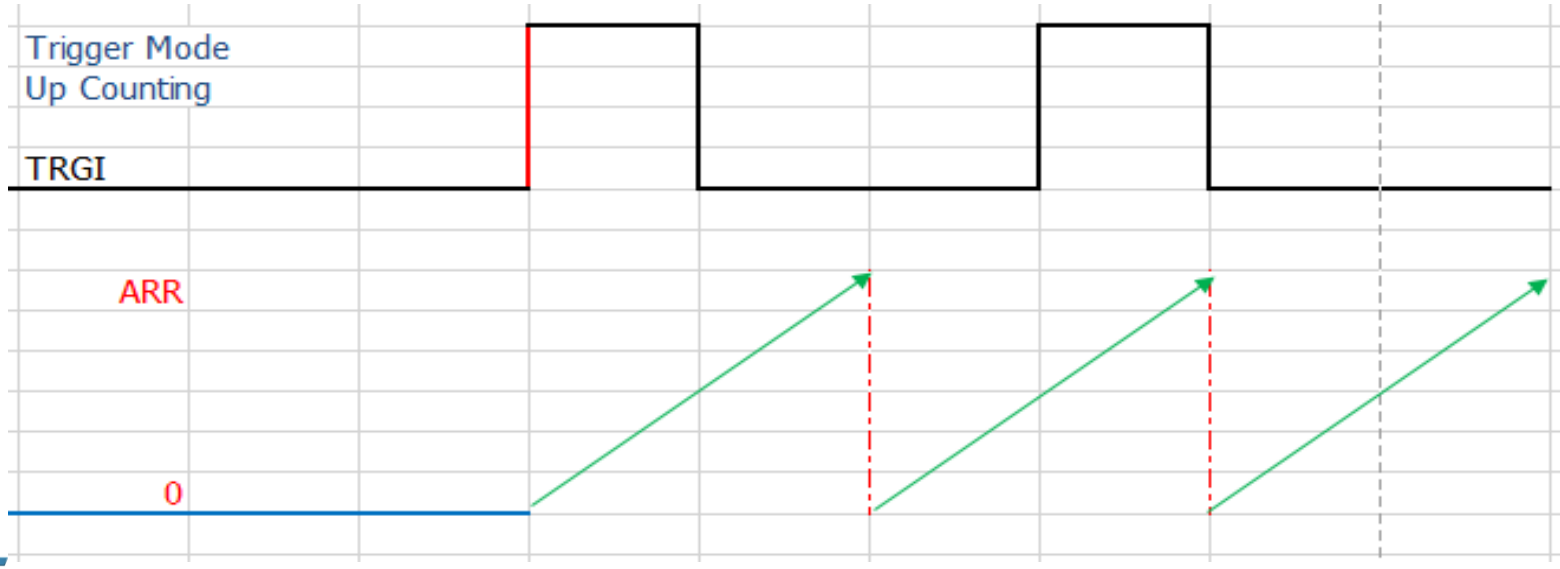
- 工作原理：根据TRGI信号电平启动或停止计数器
- 举例：TI1输入引脚低电平时使能计数器
  - 检测TI1的低电平
    - CC1S=01@TIMx_CCMR1 CC1NP/CC1P=01@TIMx_CCER
  - 定时器工作在门控模式，配置TRGI触发源
    - TS=101(TI1FP1) SMS=101@TIMx_SMCR
  - 使能计数器
    - CEN=1@TIMx_CR1(否则无论TRGI电平如何计数器都不工作)



从模式之三：触发模式【1】

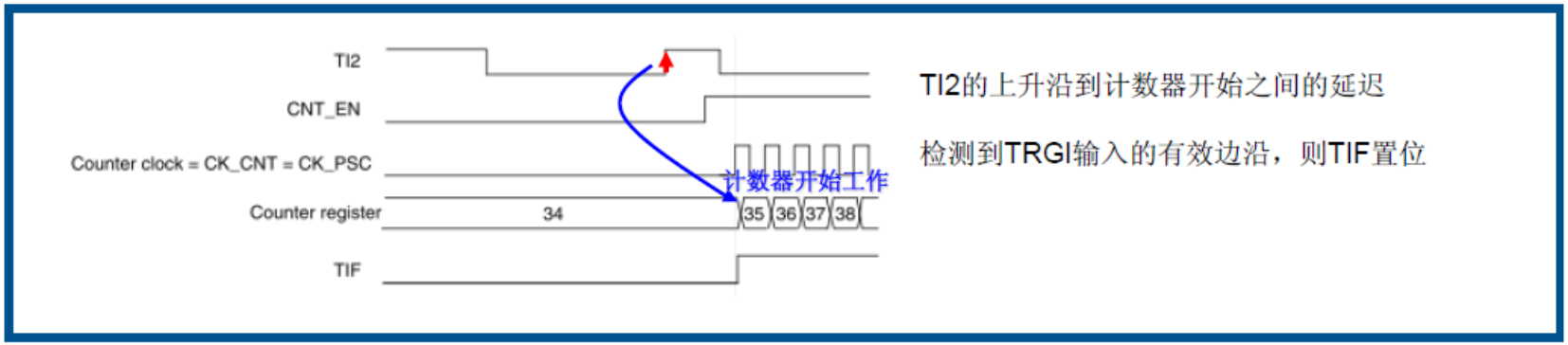
Control circuit in trigger mode





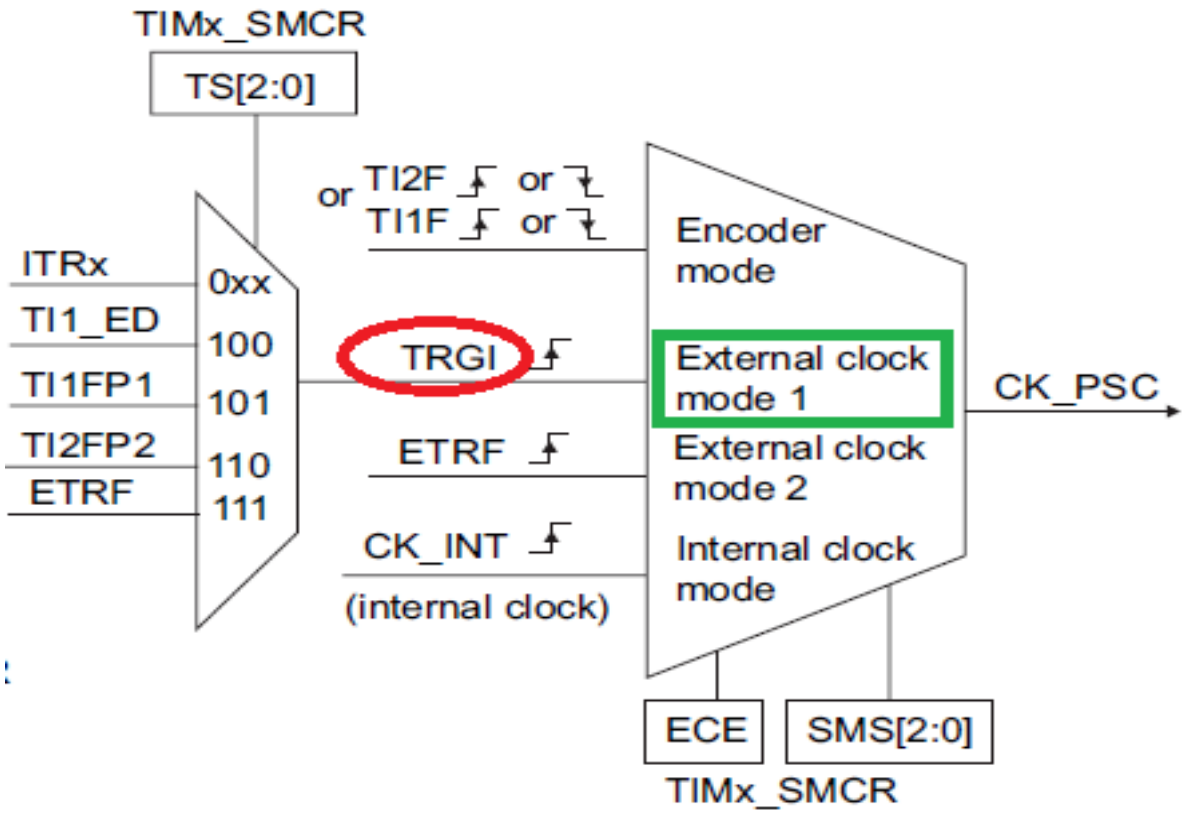
从模式之三：触发模式【2】

- 工作原理：TRGI信号有效沿启动计数器
- 举例：TI2输入引脚上升沿启动计数器
  - 检测TI2的上升沿
    - CC1S=01@TIMx_CCMR1 CC2NP/CC2P=00@TIMx_CCER
  - 定时器工作在触发模式，配置TRGI触发源
    - TS=110(TI2FP2) SMS=110@TIMx_SMCR



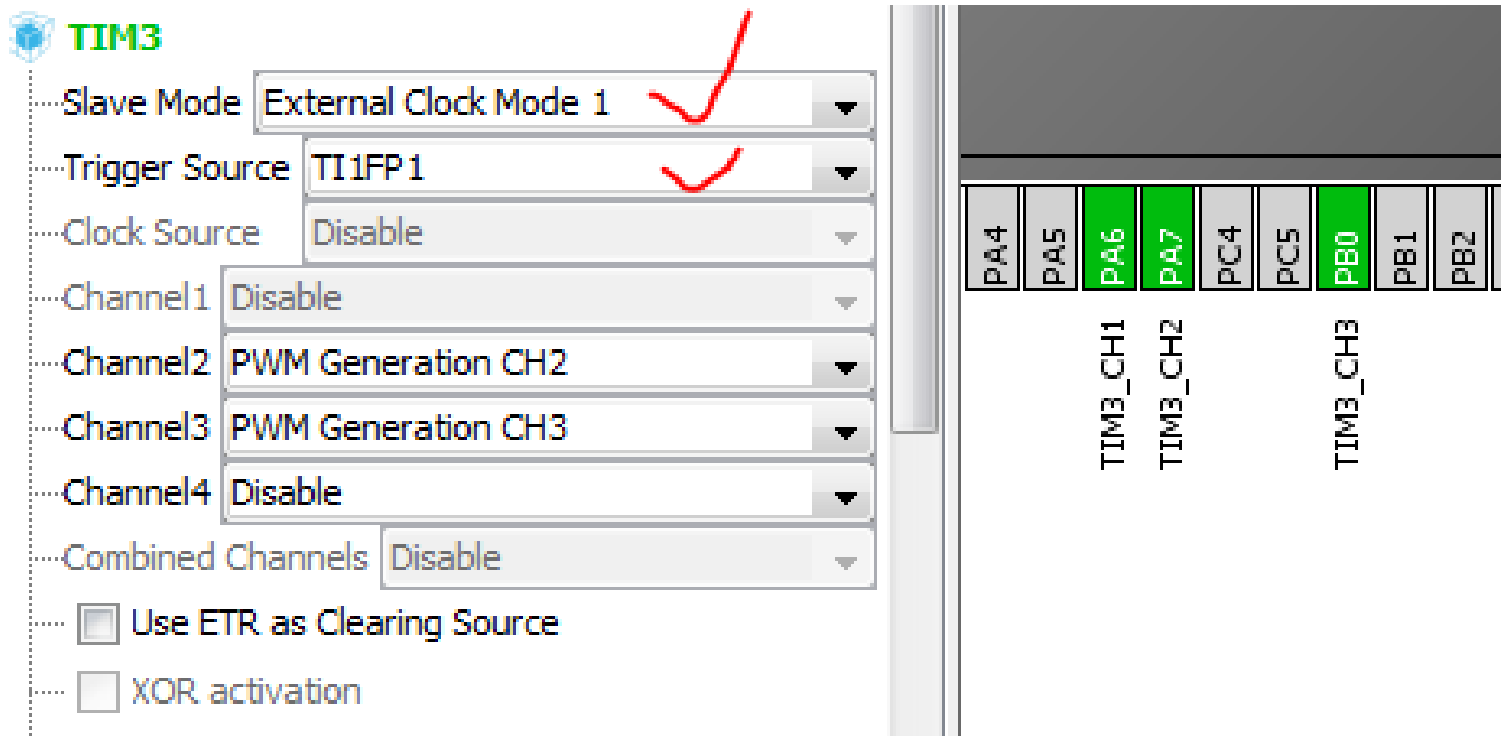
从模式之四---外部时钟模式1【1】

- 来自定时器外部的触发信号作为定时器的时钟源时，此时定时器就工作在 外部时钟模式1 从模式。它是定时器之间做级联分频时最常用的模式。



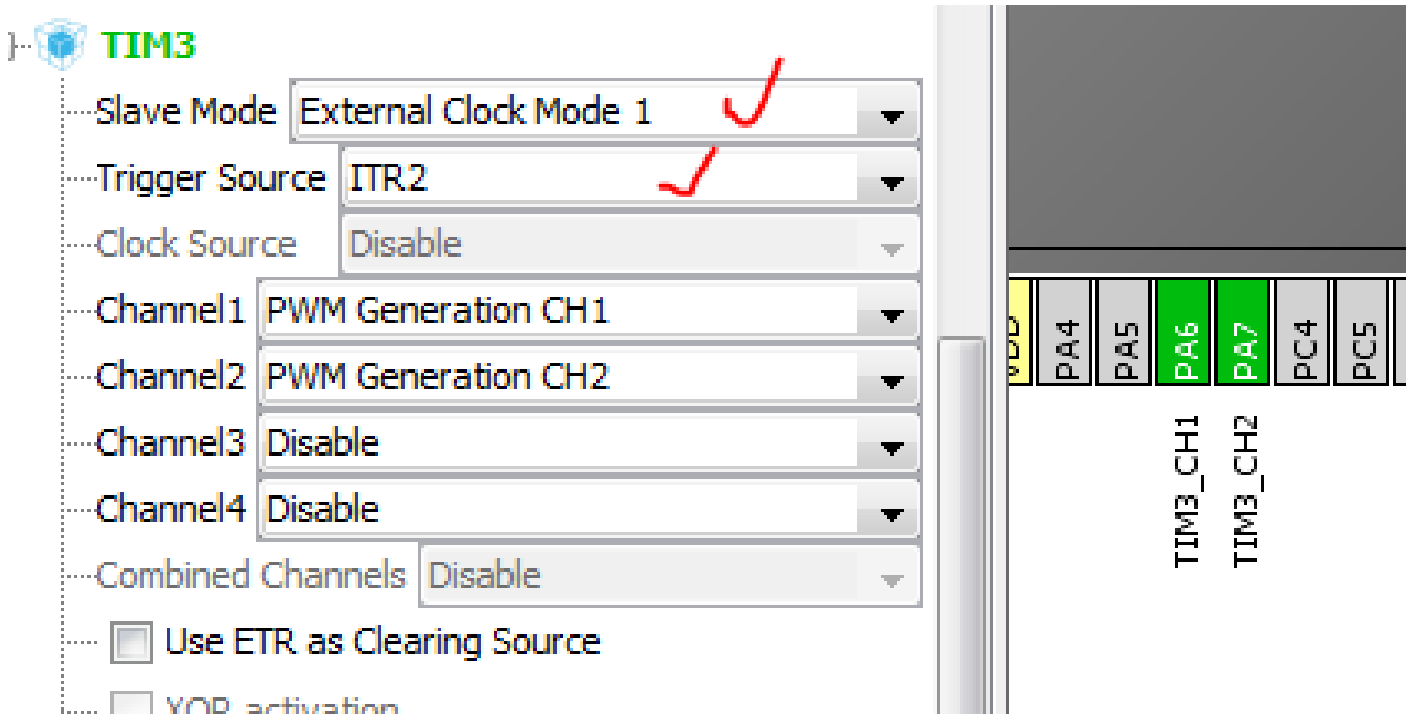
从模式之四---外部时钟模式1【2】

- 来自定时器通道1的TI1滤波信号作为定时器的时钟，工作在外部时钟模式1.



从模式之四---外部时钟模式1【3】

- 来自其它定时器外部的触发输出信号 连接到本定时器的内部触发输入端ITRx, 将ITRx信号作为本定时器的时钟源。本定时器使用该触发信号时钟生成相关PWM输出。



主从定时器之间同步级联

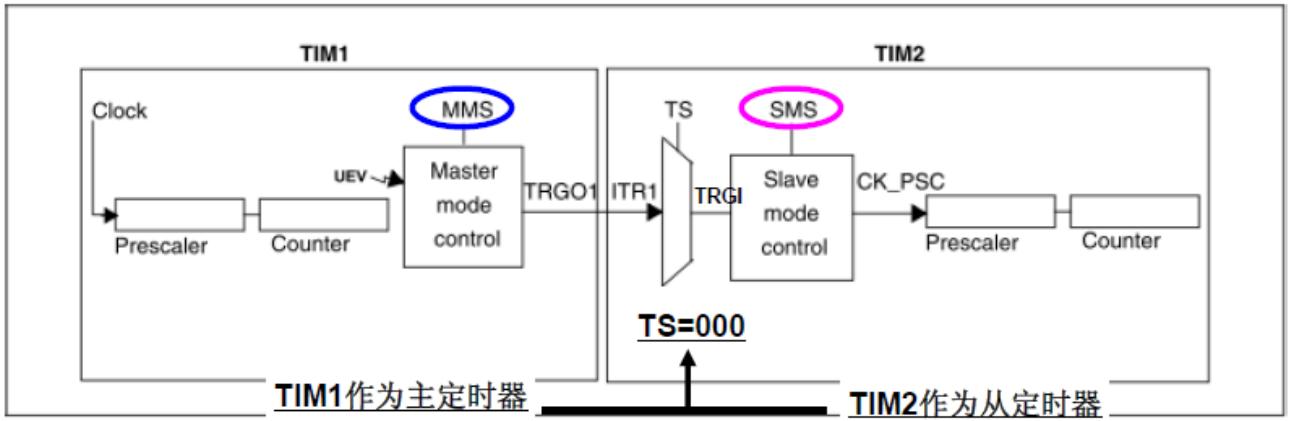
- 主定时器的TRGO连到从定时器的TRGI，作为从定时器的同步信号

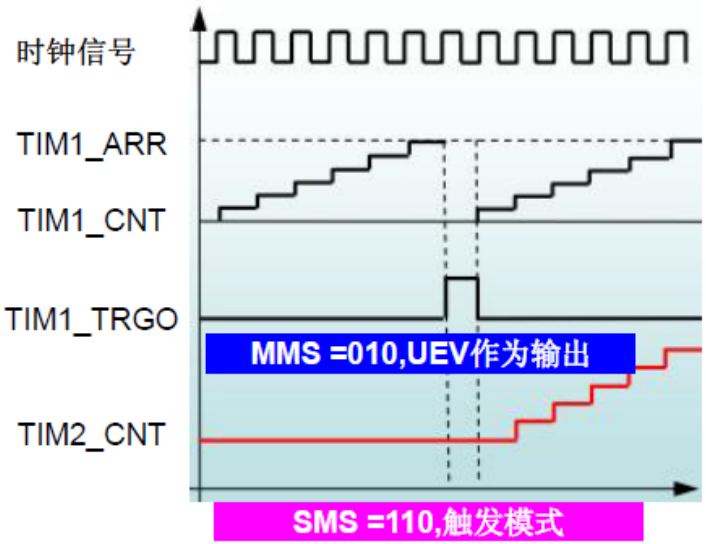
TS @ TIMx_SMCR 触发选择信号	000~011	内部触发信号0/1/2/3：ITR0/1/2/3
	100	TI1边沿检测信号：TI1F_ED
	101	定时器输入1的滤波信号：TI1FP1
	110	定时器输入2的滤波信号：TI2FP2
	111	外部触发输入：ETRF

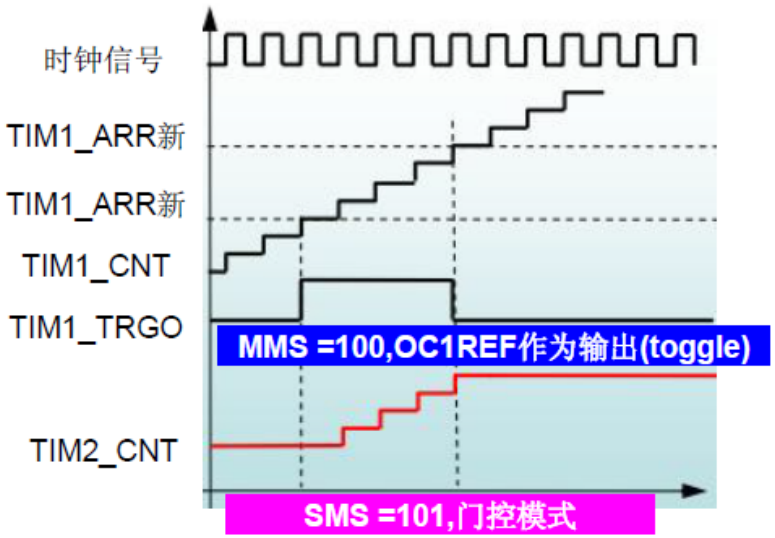
- 主/从定时器之间的相连

Slave TIM	ITR0 (TS = 000)	ITR1 (TS = 001)	ITR2 (TS = 010)	ITR3 (TS = 011)
TIM1	TIM5	TIM2	TIM3	TIM4
TIM8	TIM1	TIM2	TIM4	TIM5
TIM2	TIM1	TIM8	TIM3	TIM4
TIM3	TIM1	TIM2	TIM5	TIM4
TIM4	TIM1	TIM2	TIM3	TIM8
TIM5	TIM2	TIM3	TIM4	TIM8
TIM9	TIM2	TIM3	TIM10	TIM11
TIM12	TIM4	TIM5	TIM13	TIM14

常见主/从定时器连接







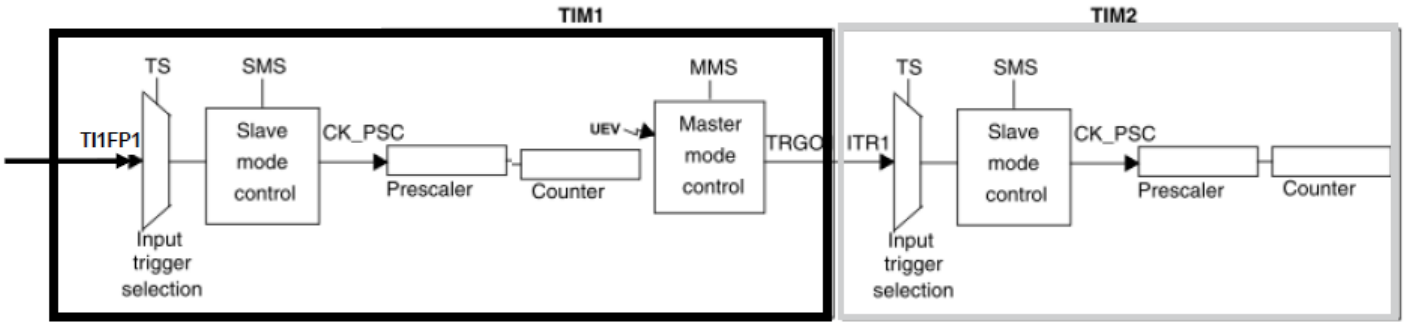
主/从定时器常见用法

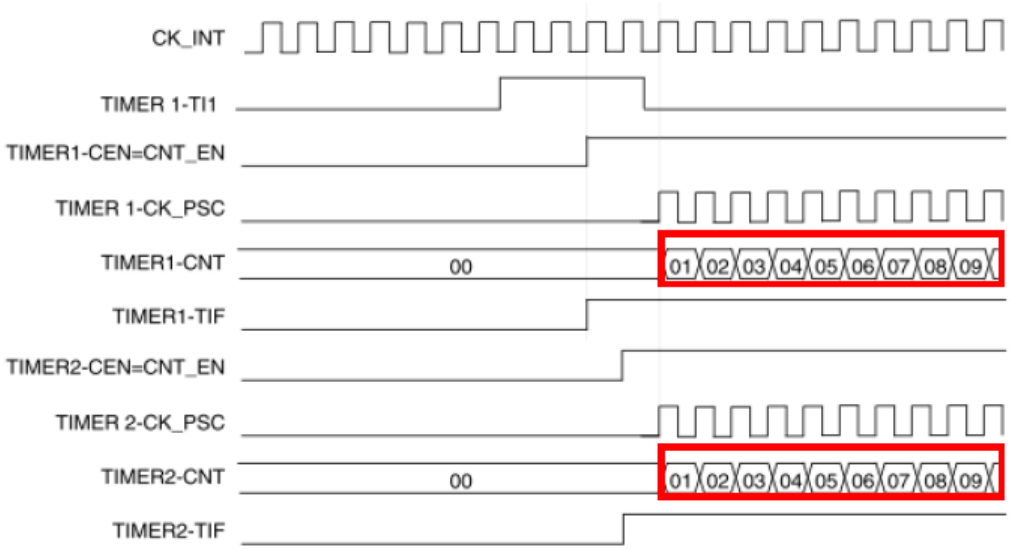
主定时器：TIM1 从定时器：TIM2		TS=000		
		TIM1配置	TIM2配置	使能各自计数器
TIM1作为TIM2的分频器	实现举例	MMS=010 更新模式	SMS=111 外部时钟模式1	置位各自的CEN
通过TIM1控制TIM2的运行和停止		MMS=100(OC1REF作为TRGO) 并在TIM1_ CCMR1配置OC1REF波形	SMS=101 门控模式	置位各自的CEN*
通过TIM1来启动TIM2的运行		MMS=010 更新模式	SMS=110 触发模式	置位TIM1的CEN**

* 注：门控模式下，必须置位从定时器的CEN，否则无论TRGI电平如何，其计数器都不运行

** 注：触发模式下，从定时器收到TRGI有效边沿则开始计数，并硬件置位CEN。直到软件复位CEN，计数器才停止。

同一外部信号同步启动2个定时器





TIM1和TIM2都使用内部时钟驱动各自的计数器

配置：
TIM1在TI1上升沿开始工作
TS=101(TI1FP1)
SMS =110(触发模式)

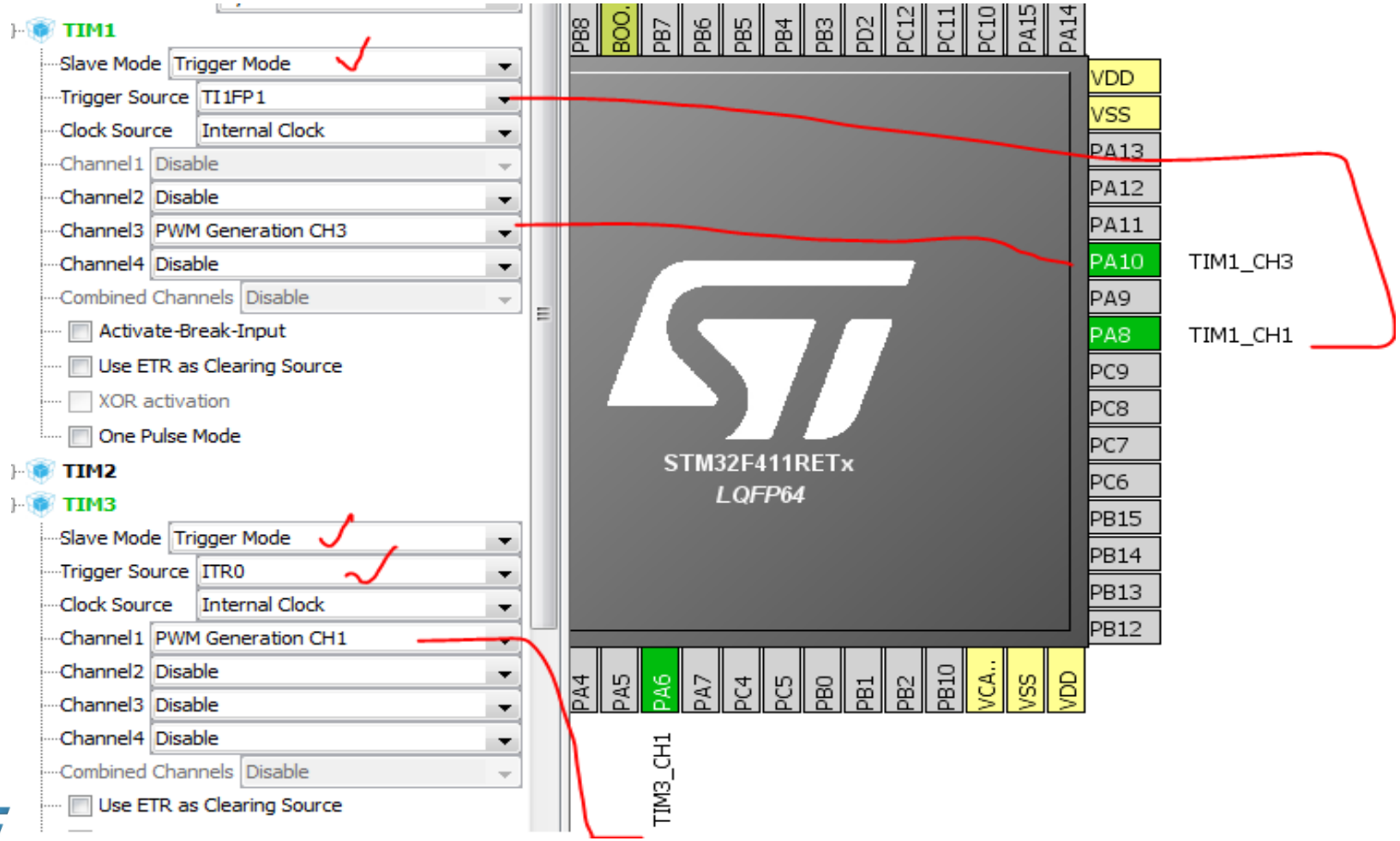
TIM1的使能信号发送给TRGO
MMS=001(enable)

置位MSM来保证TIM1和TIM2同时开始计数

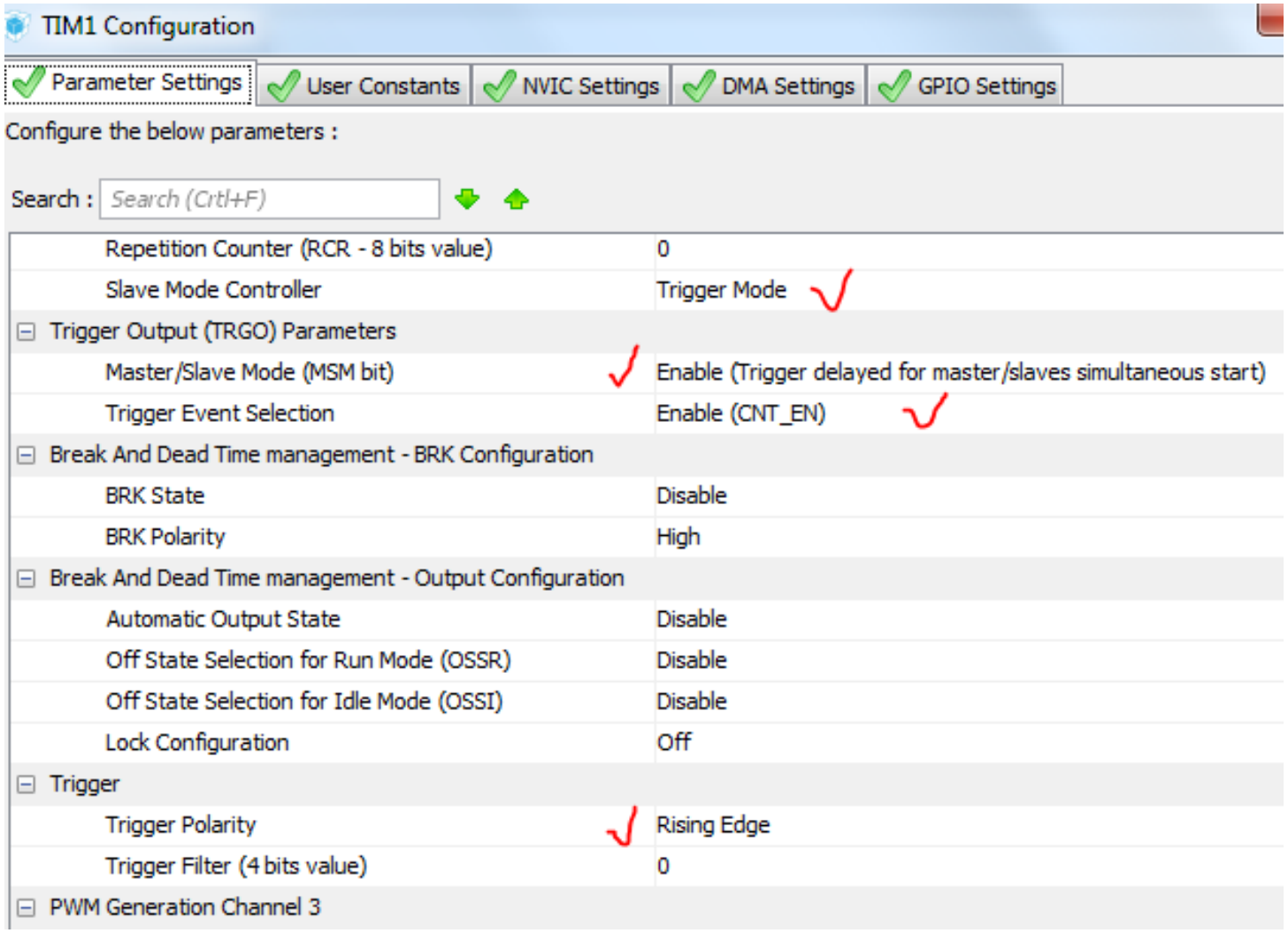
TIM2在TRGO触发下开始工作
TS=000(ITR0)
SMS=110(触发模式)

外部信号同步启动2个定时器示例【1】

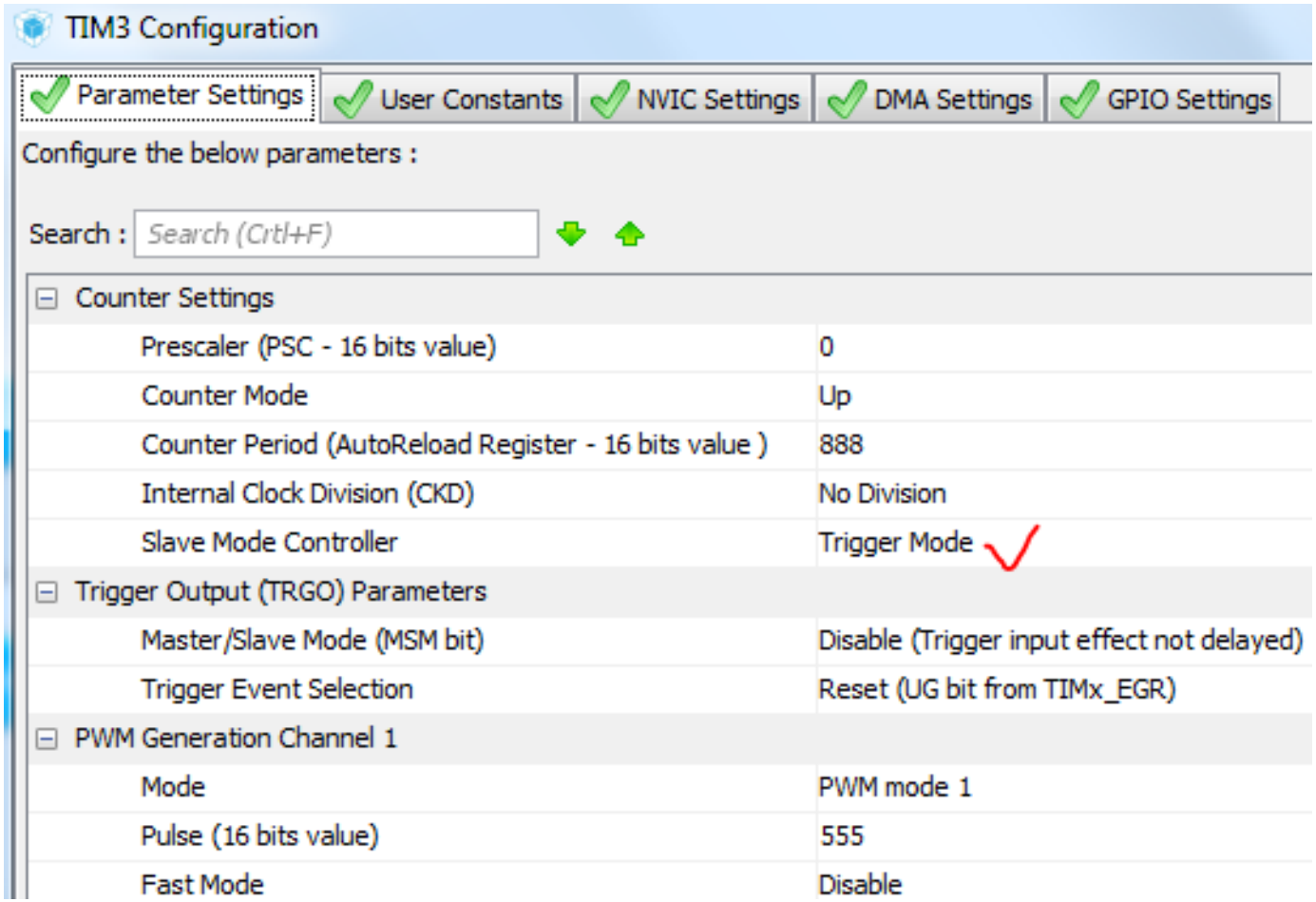
- 以TIM1-MASTER, TIM3 SLAVE ,TIM1受TI1FP1触发为例配置



外部信号同步启动2个定时器示例【2】



外部信号同步启动2个定时器示例【3】



外部信号同步启动2个定时器示例【4】

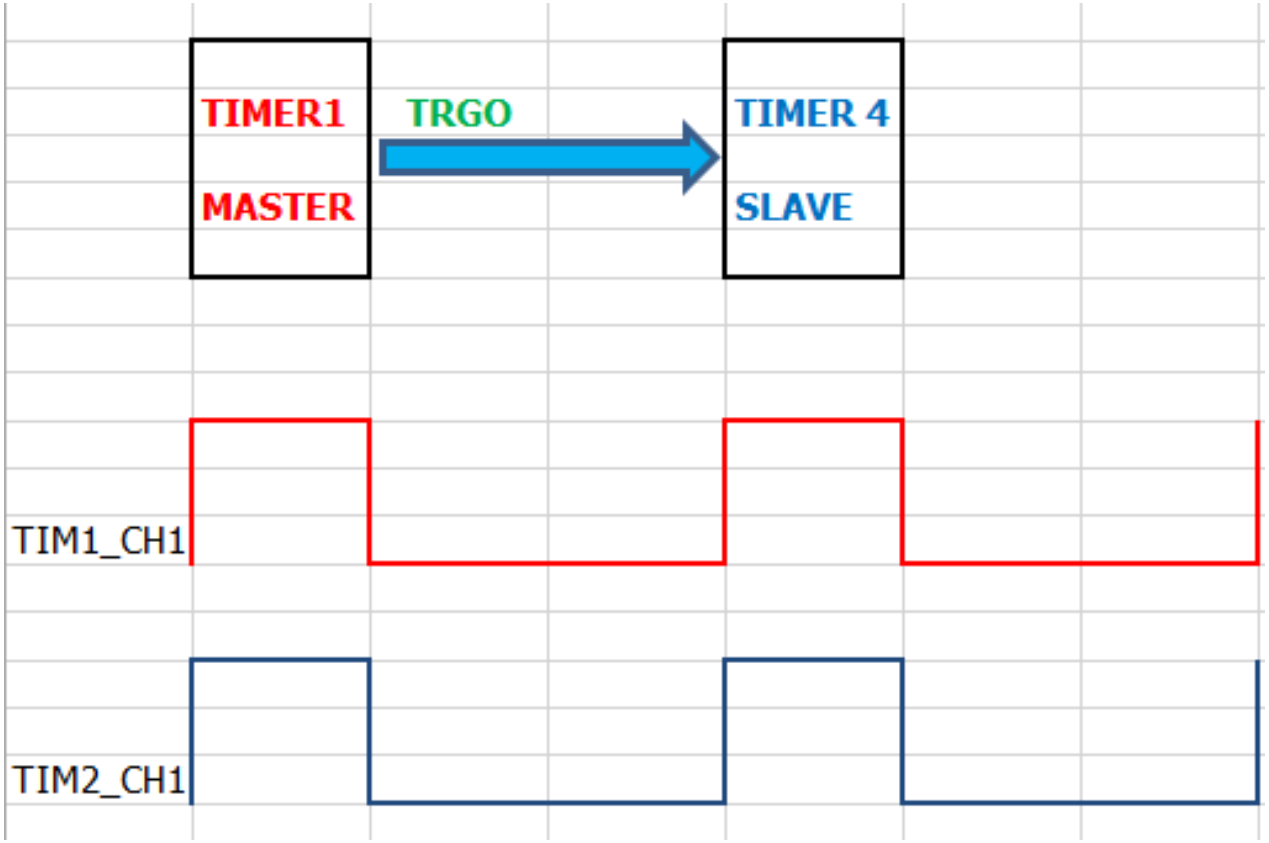
- 注意用户程序里不用启动定时器，因为二者此时都是配置为触发模式。它们的定时器使能启动交给触发信号来实现。

- 需添加的相关用户代码如下：

```
TIM_CCxChannelCmd(TIM1, TIM_CHANNEL_3, TIM_CCx_ENABLE);
  HAL_TIM_MOE_ENABLE(&htim1);

TIM_CCxChannelCmd(TIM3, TIM_CHANNEL_1, TIM_CCx_ENABLE);
```

两定时器输出同频同相波形示例【1】

- 基本原理：主模式定时器通过定时器使能操作产生触发输出信号【TRGO】启动配置在触发模式的从定时器，其它时基参数完全一样。



两定时器输出同频同相波形示例【2】

- 配置要点：

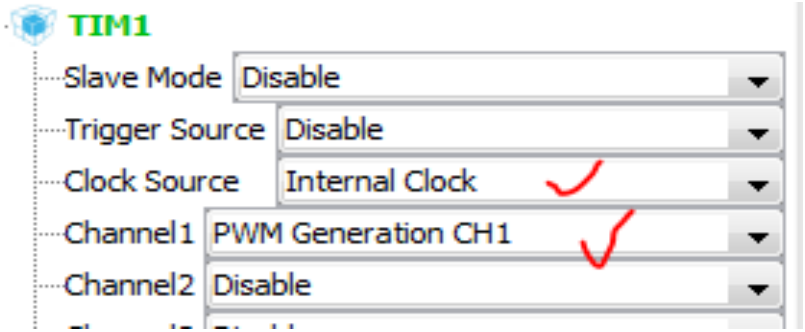
- 主定时器TIM1的计数器使能信号作为TRGO给到从模式定时器TIM2,通过查看手册得知，TIM1的TRGO连接到TIM2的IRT0输入端。

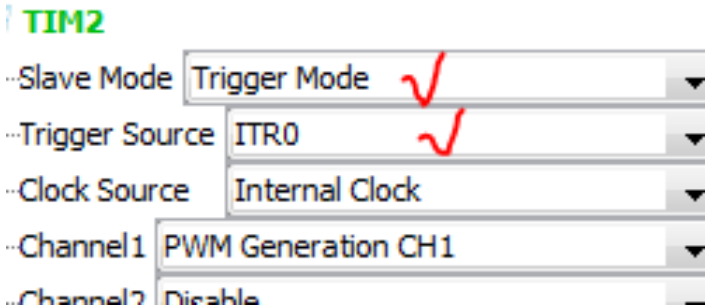
Table 53. TIMx internal trigger connection

Slave TIM	ITR0 (TS = 000)	ITR1 (TS = 001)	ITR2 (TS = 010)	ITR3 (TS = 011)
TIM2	TIM1_TRGO	Reserved	TIM3_TRGO	TIM4_TRGO
TIM3	TIM1_TRGO	TIM2_TRGO	TIM5_TRGO	TIM4_TRGO
TIM4	TIM1_TRGO	TIM2_TRGO	TIM3_TRGO	Reserved
TIM5	TIM2_TRGO	TIM3_TRGO	TIM4_TRGO	Reserved

- 从模式定时器TIM2配置在触发模式【Trigger Mode】从模式。主从定时器的时基参数完全一样。

两定时器输出同频同相波形示例【3】





TIM1

Prescaler (PSC - 16 bits value)	0
Counter Mode	Up
Counter Period (AutoReload Register - 16 bits value)	888
Internal Clock Division (CKD)	No Division
Repetition Counter (RCR - 8 bits value)	0
er Output (TRGO) Parameters	
Master/Slave Mode (MSM bit)	Disable (Trigger input effect not delayed)
Trigger Event Selection	Enable (CNT_EN)
And Dead Time management - BRK Configuration	

TIM2

Prescaler (PSC - 16 bits value)	0
Counter Mode	Up
Counter Period (AutoReload Register - 32 bits value)	888
Internal Clock Division (CKD)	No Division
Save Mode Controller	Trigger Mode
er Output (TRGO) Parameters	
Master/Slave Mode (MSM bit)	Disable (Trigger input effect not delay
Trigger Event Selection	Reset (UG bit from TIMx_EGR)

两定时器输出同频同相波形示例【4】

- 代码实现：

```
TIM_CCxChannelCmd(TIM2, TIM_CHANNEL_1, TIM_CCx_ENABLE);

HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1);
```

- 温馨提示：因为从定时器TIM2需要外来的触发信号使能其计数器，用户代码中不要使能TIM2，更不可先于主定时器的开启，否则没法做到同频同相的输出。

- 比如用户代码这样写：

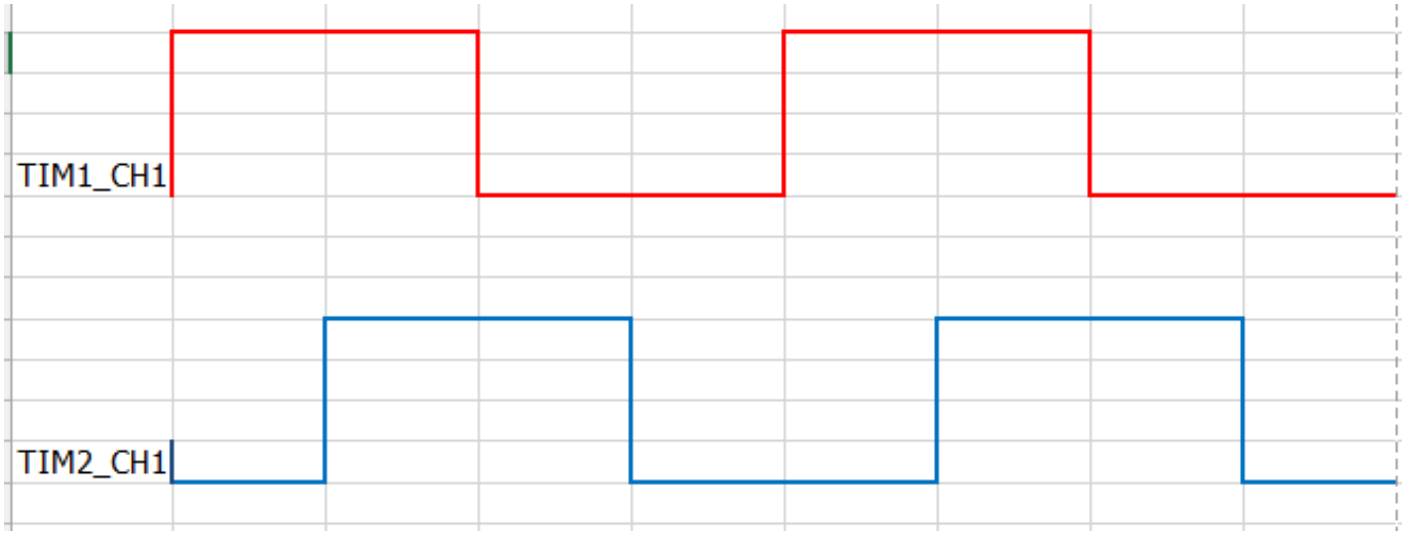
```
__HAL_TIM_ENABLE(&htim2);
TIM_CCxChannelCmd(TIM2, TIM_CHANNEL_1, TIM_CCx_ENABLE);

HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1);
```

```
HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1);
HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1);
```

两定时器输出同频同相波形示例【5】

- 基于上面同频同相应用原理，我们可以产生任意相位差的相同波形。
- 通过预置某个定时器的计数器的初始值。

```
TIM_CCxChannelCmd(TIM2, TIM_CHANNEL_1, TIM_CCx_ENABLE);
TIM2->CNT = Initial_Value;
HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1);
```



一定时器作为另一定时器的分频器示例【1】

- 基本原理：主定时器TIM1的溢出更新事件做触发输出信号TRGO连接到从定时器TIM4的内部触发输入端ITR0，并作为从定时器的时钟，此时从定时器工作在外部时钟模式1从模式。

Table 53. TIMx internal trigger connection

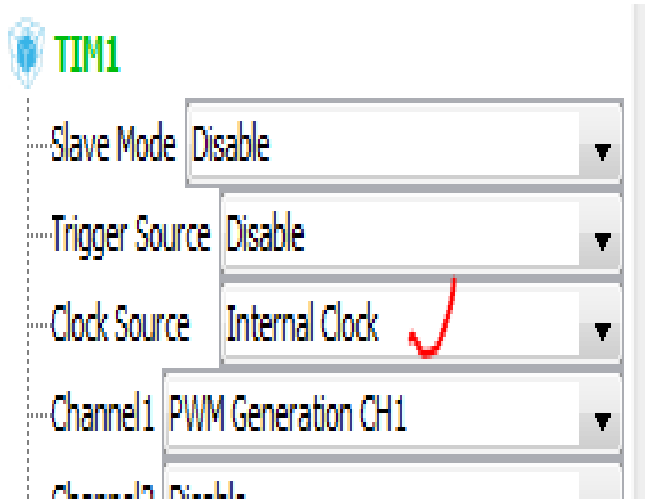
Slave TIM	ITR0 (TS = 000)	ITR1 (TS = 001)	ITR2 (TS = 010)	ITR3 (TS = 011)
TIM2	TIM1_TRGO	Reserved	TIM3_TRGO	TIM4_TRGO
TIM3	TIM1_TRGO	TIM2_TRGO	TIM5_TRGO	TIM4_TRGO
TIM4	TIM1_TRGO	TIM2_TRGO	TIM3_TRGO	Reserved

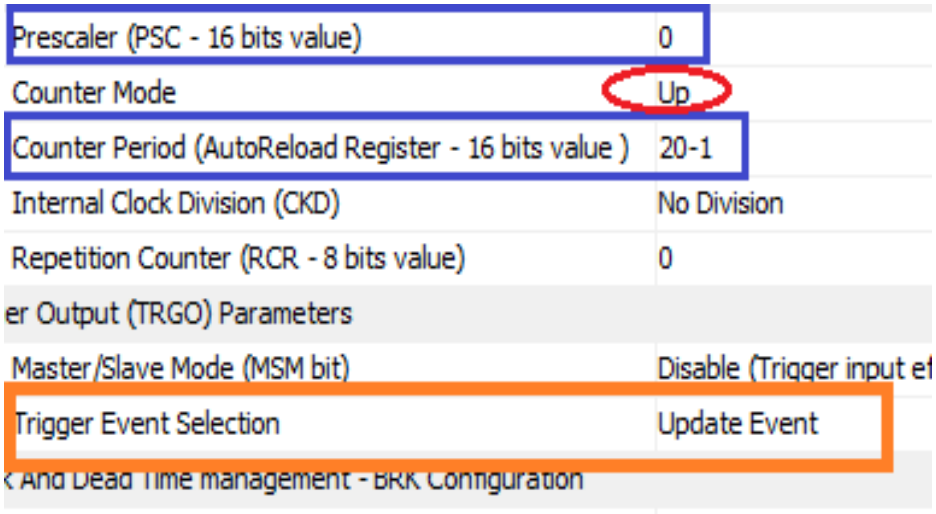
- [本PPT中案例均以32F411为主芯片,配合实验STM32F411Nucleo板。

一定时器作为另一定时器的分频器示例【2】

- 假设TIM1使用系统内部时钟【100MHz】作为计数器时钟源，向上计数模式，溢出更新频率为5Mhz,更新输出信号给到TIM4作为时钟，TIM4基于该时钟源，输出占空比40%的50Khz的PWM波形。

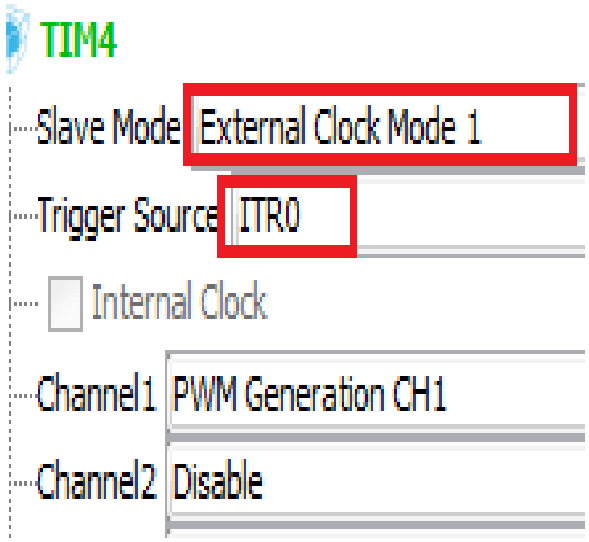
- 确定好时基参数、做配置





一定时器作为另一定时器的分频器示例【3】

- TIM4时钟源于TIM1输出的5Mhz触发输出信号。



TIM4

Prescaler (PSC - 16 bits value)	0
Counter Mode	Up
Counter Period (AutoReload Register - 16 bits value)	100-1
Internal Clock Division (CKD)	No Division
Slave Mode Controller	ETR mode 1
er Output (TRGO) Parameters	
Master/Slave Mode (MSM bit)	Disable (Trigger input effect not
Trigger Event Selection	Reset (UG bit from TIMx_EGR)
Generation Channel 1	
Mode	PWM mode 1
Pulse (16 bits value)	40
Fast Mode	Disable
CH Polarity	High

```
HAL_TIM_PWM_Start(&htim4, TIM_CHANNEL_1);

HAL_TIM_Base_Start(&htim1);
//HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1);
```

有关主从定时器应用的小结

- 1、定时器主从的双重性
- 2、定时器间内联不能随意，需结合手册核对。

Slave TIM	ITR0 (TS = 000)	ITR1 (TS = 001)	ITR2 (TS = 010)	ITR3 (TS = 011)
TIM2	TIM1_TRGO	Reserved	TIM3_TRGO	TIM4_TRGO
TIM3	TIM1_TRGO	TIM2_TRGO	TIM5_TRGO	TIM4_TRGO

- 3、为保证多个上下级联关系的定时器完美同步,主模式定时器需要配置MSM位

- 4、复位模式下的从定时器在收到触发信号时，定时器的寄存器会被重新初始化。【即影子寄存器的更新、计数器重置】

十、定时器事件与其它外设的关联

- 定时器除了定时器之间建立同步、级联关系外，它的各类事件或信号还能与其它外设关联，建立定时触发关系。比如：触发DMA请求、触发ADC转换、DAC转换等

DAC trigger selection

If the TENx control bit is set, cc
counter, external interrupt line).
possible events will trigger con

Source
Timer 6 TRGO event
Timer 8 TRGO event
Timer 7 TRGO event
Timer 5 TRGO event
Timer 2 TRGO event
Timer 4 TRGO event

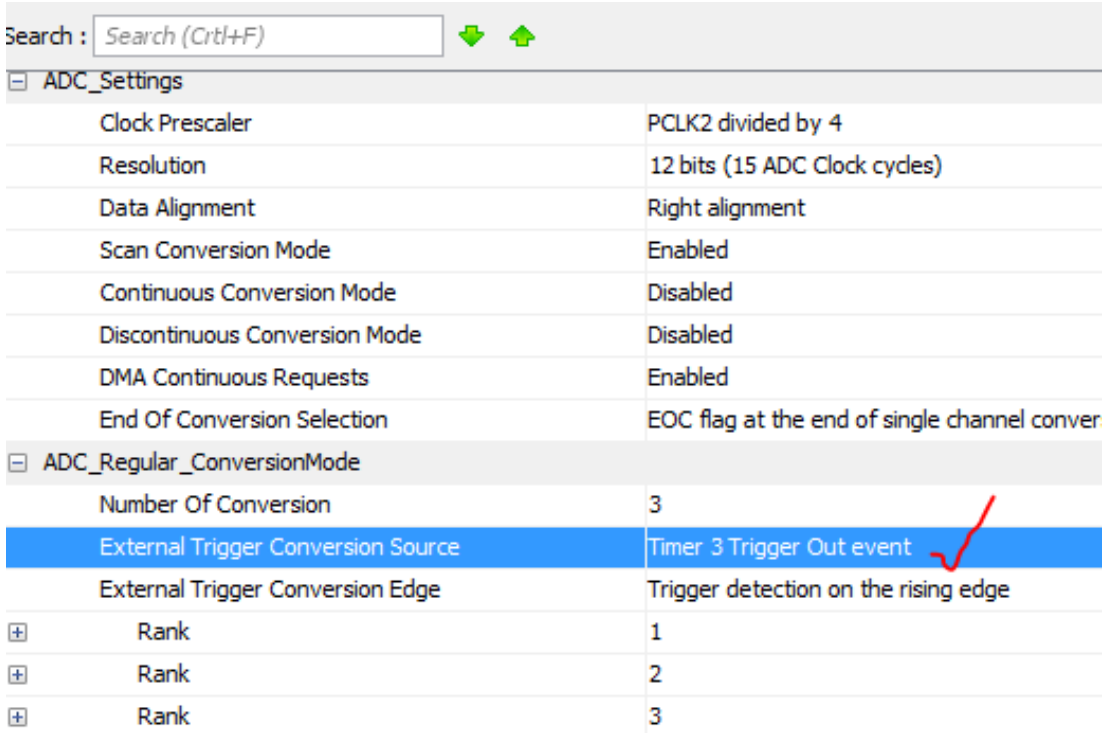
Analog-to-digital converter (ADC)

Ta

Source
TIM1_CH1 event
TIM1_CH2 event
TIM1_CH3 event
TIM2_CH2 event
TIM2_CH3 event
TIM2_CH4 event
TIM2_TRGO event
TIM3_CH1 event

TIMER+ADC+DMA实例分享

- 结合前面介绍可知ADC的转换启动可以通过外部触发来启动
- TIMER提供TRGO触发ADC的转换
- 比如：使用TIM3周期性触发ADC1的转换，开启3个AD规则通道.
- ADC1_C0/ADC1_1/Vrefint
- 使用TIM3_TRGO
- Update 事件做为TRGO
- 开启ADC的DMA请求



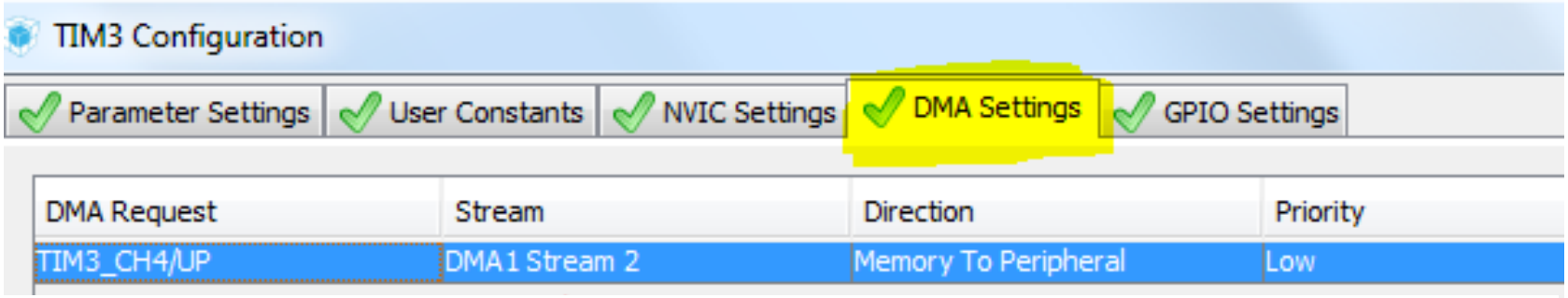
TIMER+ADC+DMA实例分享

```
hdma_adc1.Instance = DMA2_Stream0;
hdma_adc1.Init.Channel = DMA_CHANNEL_0;
hdma_adc1.Init.Direction = DMA_PERIPH_TO_MEMORY;
hdma_adc1.Init.PeriphInc = DMA_PINC_DISABLE;
hdma_adc1.Init.MemInc = DMA_MINC_ENABLE;
hdma_adc1.Init.PeriphDataAlignment = DMA_PDATAALIGN_HALFWORD;
hdma_adc1.Init.MemDataAlignment = DMA_MDATAALIGN_HALFWORD;
hdma_adc1.Init.Mode = DMA_CIRCULAR;
hdma_adc1.Init.Priority = DMA_PRIORITY_LOW;
hdma_adc1.Init.FIFOMode = DMA_FIFOMODE_DISABLE;
if (HAL_DMA_Init(&hdma_adc1) != HAL_OK)
{
  _Error_Handler(__FILE__, __LINE__);
}

__HAL_LINKDMA(hadc,DMA_Handle,hdma_adc1);
```

```
if(HAL_ADC_Start_DMA(&hadc1, (uint32_t*)ADCxConvertedValue, 3) != HAL_OK)
{
  /* Start Conversation Error */
  Error_Handler();
}
HAL_TIM_Base_Start(&htim3);
```

TIMER事件触发DMA做数据传输【1】

- TIM3更新事件触发DMA请求，DMA从内存将数据送到SPI 2数据寄存器，从而完成数据发送。【基于STM32F411nucleo板。



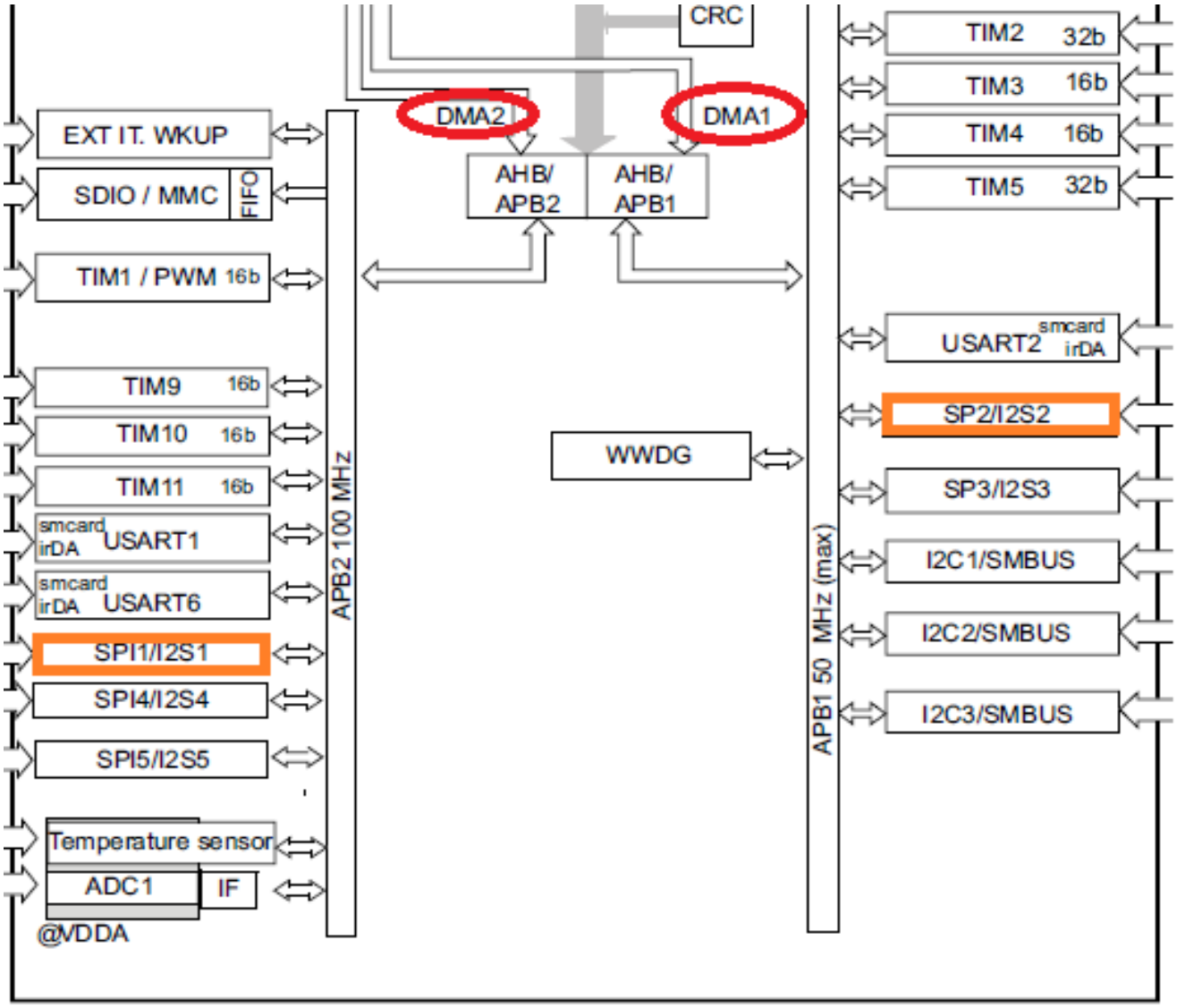

```
/* TIM3 DMA Init */
/* TIM3_CH4_UP Init */
hdma_tim3_ch4_up.Instance = DMA1_Stream2;
hdma_tim3_ch4_up.Init.Channel = DMA_CHANNEL_5;
hdma_tim3_ch4_up.Init.Direction = DMA_MEMORY_TO_PERIPH;
hdma_tim3_ch4_up.Init.PeriphInc = DMA_PINC_DISABLE;
hdma_tim3_ch4_up.Init.MemInc = DMA_MINC_ENABLE;
hdma_tim3_ch4_up.Init.PeriphDataAlignment = DMA_PDATAALIGN_HALFWORD;
hdma_tim3_ch4_up.Init.MemDataAlignment = DMA_MDATAALIGN_HALFWORD;
hdma_tim3_ch4_up.Init.Mode = DMA_CIRCULAR;
hdma_tim3_ch4_up.Init.Priority = DMA_PRIORITY_LOW;
hdma_tim3_ch4_up.Init.FIFOMode = DMA_FIFOMODE_DISABLE;
if (HAL_DMA_Init(&hdma_tim3_ch4_up) != HAL_OK)
{
  _Error_Handler(__FILE__, __LINE__);
}
__HAL_LINKDMA(htim_base,hdma[TIM_DMA_ID_UPDATE],hdma_tim3_ch4_up);
```

TIMER事件触发DMA做数据传输【2】


```
HAL_DMA_Start(&hdma_tim3_ch4_up, (uint32_t )&Tx_Buffer,(uint32_t )&SPI2->DR,Data_Number );
__HAL_TIM_ENABLE_DMA(&htim3, TIM_DMA_UPDATE);
__HAL_SPI_ENABLE(&hspi2);

HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_1);
```

- 温馨提示：
- 我们利用TIMER事件来作为DMA请求源时，作为数据传输的源端或目的端，都是我们用户指定的，这时一定要**注意源端和目标端是当前DMA流所支持的。**
- 比方上面的例子，如果改成SPI1就出不来了。因为DMA1根本访问不到SPI1.[上面例子是基于STM32F411的】

TIMER事件触发DMA做数据传输【3】



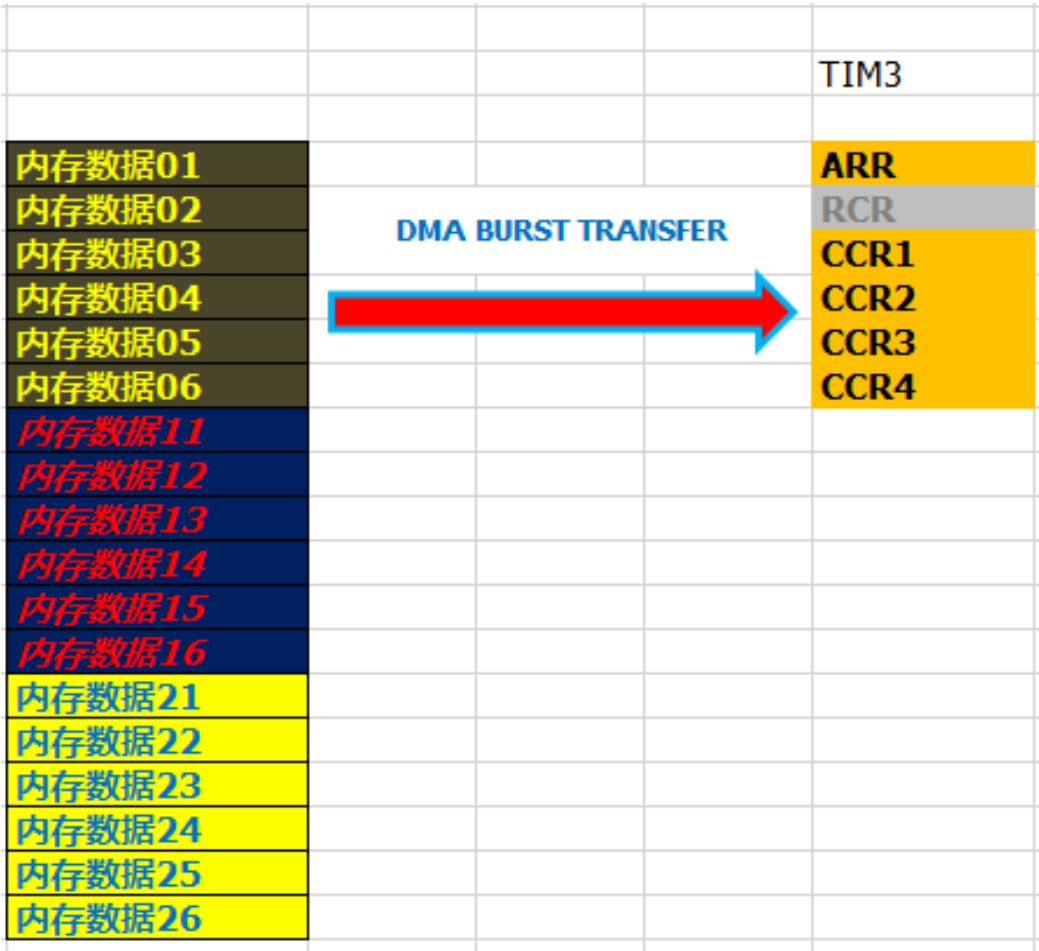
life.augmented

分享最后一个案例【1】

- 使用定时器的DMA批量传输，同时修改ARR/CCR等多个寄存器。
- 它开启了ARR/CCR的预装载功能，初始化时令ARR=0,CCR=0
- 使用TIMER的更新事件触发DMA. 该TIMER也支持定时器的DMA批量传输。
- 结果发现根本没有波形出来，怎么回事呢？
- 1、ARR=0? 有错？
- 2、 ARR,CCRx初始为0是有意为之，不存在错的说法。
- 3、本意是想软件方式手动复位，即对UG@TIMx_EGR置位产生更新事件。然后触发DMA，传输相关数据到寄存器组。

分享最后一个案例【2】

- 手动做定时器的复位操作并激发更新事件，让DMA将内存区的数据传输到ARR/CCRx的等寄存器；
- 添加软件复位代码：
- TIM3->EGR |= TIM_EGR_UG;
- 运行测试，还是无波形输出！
- 为什么呢？
- 此次数据只传到预装载寄存器。
- 既然这样，再来一次手动复位？



分享最后一个案例【3】

- 添加软件复位代码： TIM3->EGR |= TIM_EGR_UG，产生更新事件。
- 这次就可以实现上次预装载的数据更新到影子寄存器中发挥作用。
- 再次编译调试,运行正常。
- 不过，实际调试过程中，有时还是会遇到不启动的情形，估计是两个复位操作太接近了，可能硬件没识别为2个更新事件。建议两个复位操作间随便插入一条不影响功能的语句。

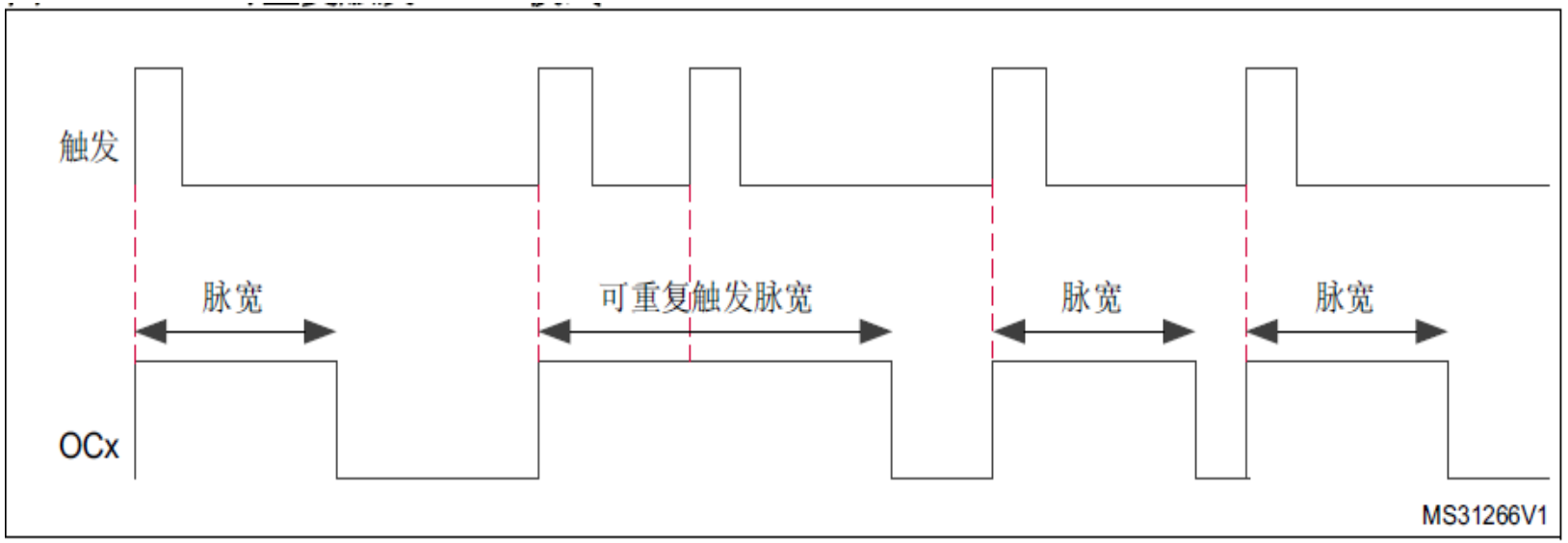
```
TIM3->EGR |= TIM_EGR_UG; //prepare data in buffer

__HAL_TIM_ENABLE(&htim3); //insert a gag between 2 consecutive Update event

TIM3->EGR |= TIM_EGR_UG; //write data into shadow
```

- 本案例考察定时器的批量传输、更新事件、影子寄存器的更新等

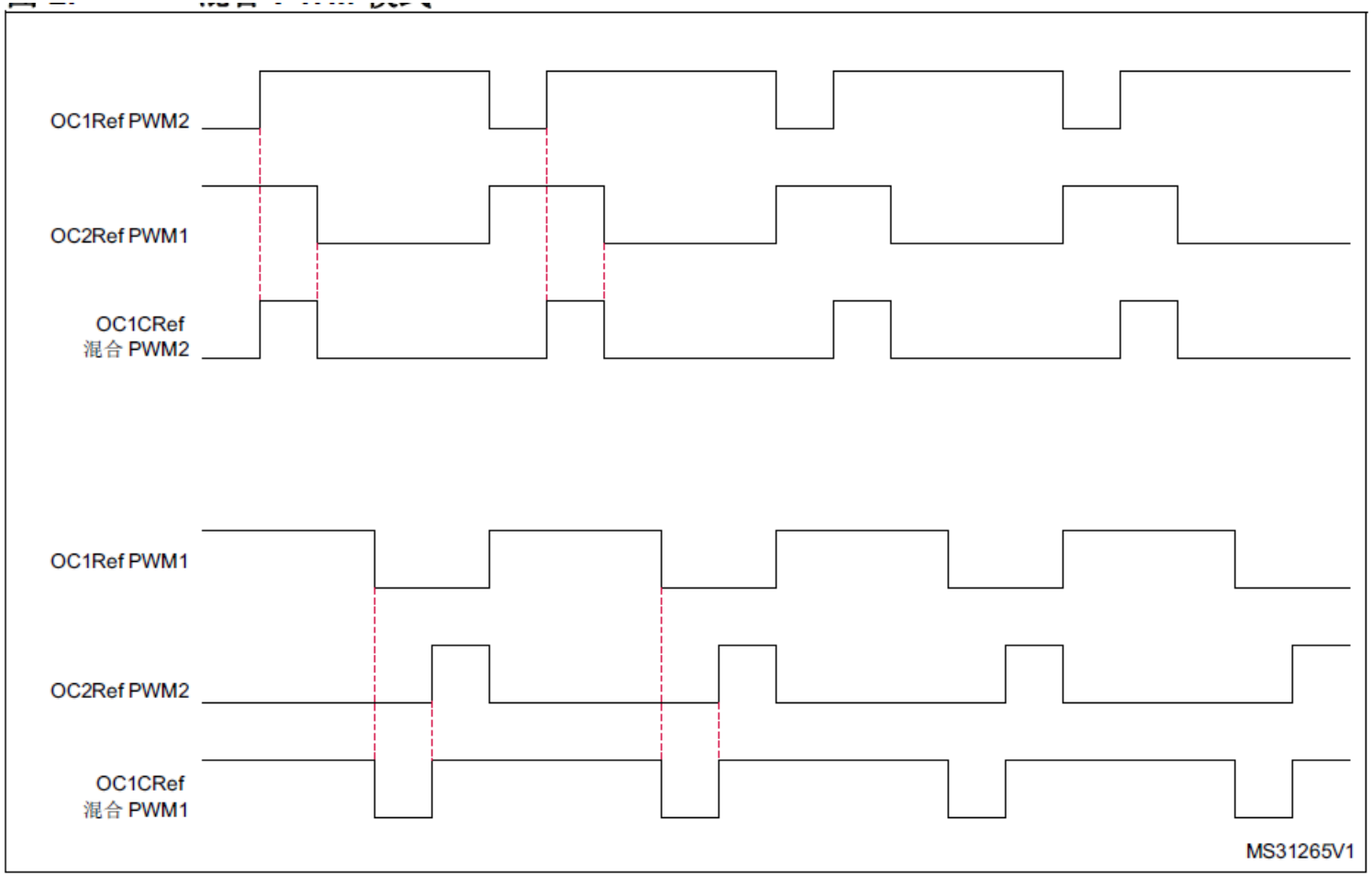
十一、STM32F7/F3/L4新增PWM模式

- 1、STM32L4/F7和STM32F30X/STM32F3X8的非对称PWM模式
- 2、 STM32L4/F7和STM32F30X/STM32F3X8的混合PWM模式
- 3、 STM32L4/F7和STM32F30X/STM32F3X8的重触发单脉冲模式

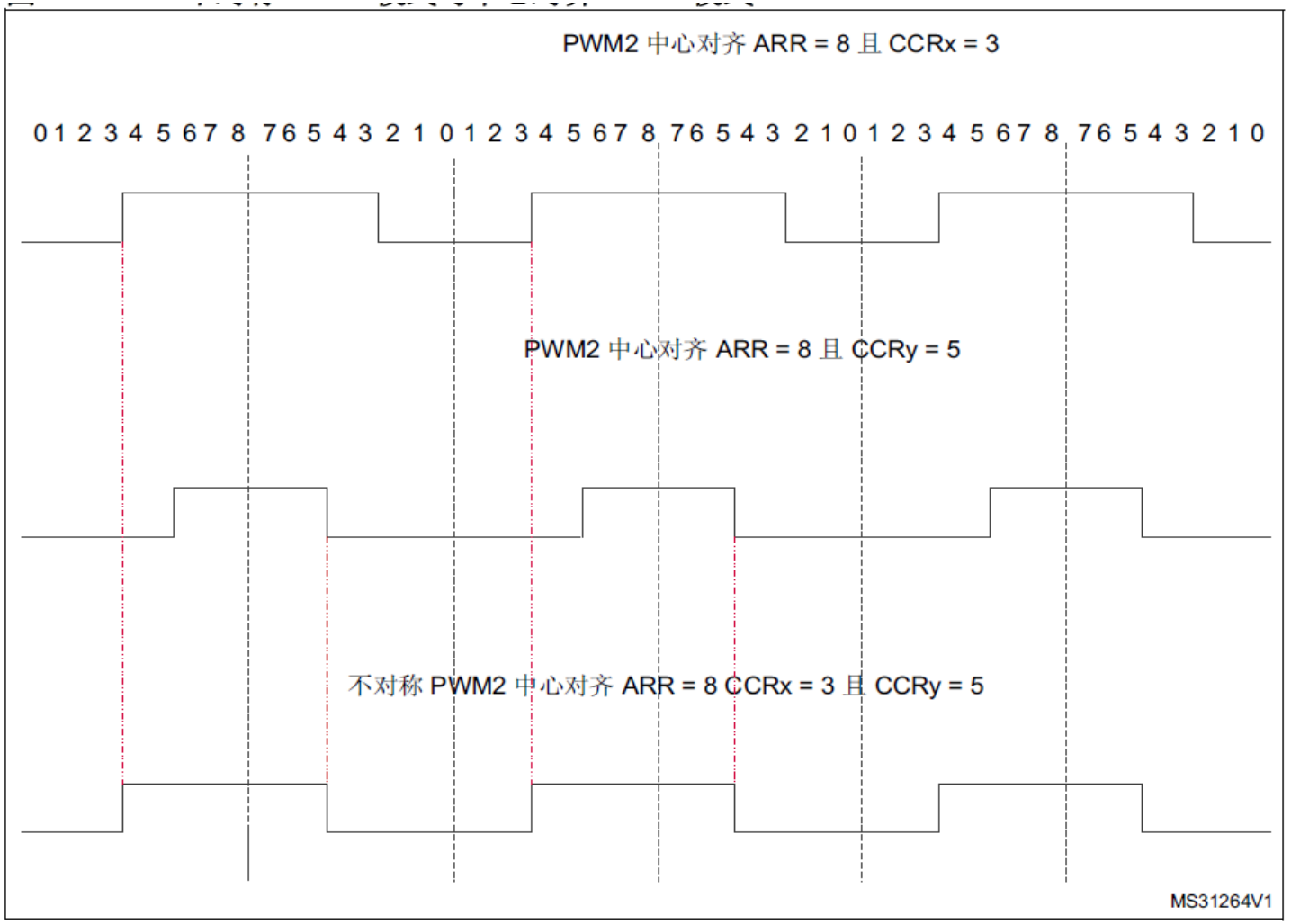
可重触发单脉冲模式



混合PWM模式



非对称PWM模式



实验练习环节

- 1、通过定时器主从级联实现pwm输出
- 2、实现2个定时器的完美同步输出[同频同相]。
- 完成之后，再在此基础上再实现两路固定相差为120°的相同形状的波形输出。
- 3、自主实现有限脉冲个数的输出 【方法很多】

通过定时器主从级联实现pwm输出

- 开发板：STM32F411-Nucleo
- MASTER: TIM3；SLAVE: TIM4
- [tim3 使用内部时钟 100Mh作为TIM3的时钟源]
- TIM3的更新事件做触发输出并作为TIM4的计数时钟。
- TIM3的更新输出频率为2Mhz，希望TIM4_CH1输出占空比为40%、频率为10Khz的PWM波形。

411

Table 53. TIMx internal trigger connection

Slave TIM	ITR0 (TS = 000)	ITR1 (TS = 001)	ITR2 (TS = 010)	ITR3 (TS = 011)
TIM2	TIM1_TRGO	Reserved	TIM3_TRGO	TIM4_TRGO
TIM3	TIM1_TRGO	TIM2_TRGO	TIM5_TRGO	TIM4_TRGO
TIM4	TIM1_TRGO	TIM2_TRGO	TIM3_TRGO	Reserved
TIM5	TIM2_TRGO	TIM3_TRGO	TIM4_TRGO	Reserved

实现2个定时器的完美同步输出[同频同相]

- 这是个比较实用而常让有些人不知如何实现的一个功能。
- Master:TIM5　Slave: TIM3

Table 53. TIMx internal trigger connection

Slave TIM	ITR0 (TS = 000)	ITR1 (TS = 001)	ITR2 (TS = 010)	ITR3 (TS = 011)
TIM2	TIM1_TRGO	Reserved	TIM3_TRGO	TIM4_TRGO
TIM3	TIM1_TRGO	TIM2_TRGO	TIM5_TRGO	TIM4_TRGO
TIM4	TIM1_TRGO	TIM2_TRGO	TIM3_TRGO	Reserved
TIM5	TIM2_TRGO	TIM3_TRGO	TIM4_TRGO	Reserved

实现2个定时器的固定相差输出

- 在前面的基础上，再实现两路固定相差为120°的相同形状的波形。

Table 53. TIMx internal trigger connection

Slave TIM	ITR0 (TS = 000)	ITR1 (TS = 001)	ITR2 (TS = 010)	ITR3 (TS = 011)
TIM2	TIM1_TRGO	Reserved	TIM3_TRGO	TIM4_TRGO
TIM3	TIM1_TRGO	TIM2_TRGO	TIM5_TRGO	TIM4_TRGO
TIM4	TIM1_TRGO	TIM2_TRGO	TIM3_TRGO	Reserved
TIM5	TIM2_TRGO	TIM3_TRGO	TIM4_TRGO	Reserved

自主实现有限脉冲个数的输出

- 参考方法：
- 1、使用高级定时器的RCR功能输出指定个数脉冲
- 这个使用cube简单配置即可完成，较为简单。但很可能会遇到脉冲个数跟预期不一致问题。
- 2、使用通用定时器输出指定个数脉冲
- 可以使用PWM输出模式，开启比较中断。在比较中断里进行脉冲个数计数，到数时直接修改CCR值为0或大于ARR的值。这样实现较为简洁。
- 3、上面的方法改为DMA传输数据，到达指定个数后DMA直接传输一个等于0或比ARR还大的数据给到CCR。以UP+PWM1为例，CCR=0输出低，CCR=ARR+N【N>=1】时输出高。

Thank you

Releasing your creativity with the STM32