芯海科技
CHIPSEA

CS32F0XX TIMER
外设模块应用笔记

REV 1.0

芯海科技（深圳）股份有限公司
地　址：深圳市南山区蛇口南海大道1079号花园城数码大厦A座9楼
电　话：+(86 755)86169257　　传　真：+(86 755)86169057
网　站：www.chipsea.com　　邮　编：518067
微信号：芯海科技

版本历史

历史版本	修改内容	版本日期
REV 1.0	初版生成	2019-11-15

目录

1 使用概要

1.1 前言

本应用笔记旨在展示使用 CS32F0xx 微控器，针对定时器外设的应用。帮助用户了解 CS32F0xx 定时器的基本特性、操作模式及相关应用的示例代码。提供的一些高级应用以便缩短用户开发周期。对所介绍的特定用户示例做了工作原理与相关代码介绍， 以方便用户快速移。

本应用笔记分为四部分:

- 定时器特性概述
- 基本定时器功能介绍
- 高级定时器功能介绍
- 特定场景应用示例

下表列出了本应用笔记覆盖的 CS32 微控器类型及 PACK 版本号。

表 1.1 适用类型

类型	编号
微控制器	CS32F03x(CS32F030, CS32F031)
PACK 包	Chipsea.CS32F0xx_DFP.1.0.0

2 定时器特性概述

CS32F03X 系列微控器内集成了 8 个定时器。按照功能的不同，可以分为高级定时器、通用定时器和基本定时器 3 种。

- 基本定时器：具有基本的定时和计数功能
- 通用定时器：具有基本的定时、计数、输入捕获、输出比较以及 PWM 输出功能
- 高级定时器：具有基本的定时、计数、输入捕获、输出比较以及 PWM 输出、支持互补输出、死区时间控制、定时器从模式功能

CS32F03X 微控制器内集成了 8 个具有定时器功能的定时器。包含 1 个 16 位高级定时器、1 个 32 位及 5 个 16 位通用定时器、1 个 16 位基本定时器。定时器具体配置及功能详见下表 2-1。

表 2.1 定时器类型及功能

类型	定时器	分辨率	计数类型	预分频	DMA 请求	捕获比较通	互补输出
高级控制	TIM1	16 位	向上、向下、向上/向下	1-65536	支持	4	3
通用	TIM2	32 位	向上、向下、向上/向下	1-65536	支持	4	-
	TIM3	16 位	向上、向下、向上/向下	1-65536	支持	4	-
	TIM14	16 位	向上	1-65536	不支持	1	-
	TIM15(1)	16 位	向上	1-65536	支持	2	1
	TIM16	16 位	向上	1-65536	支持	1	1
	TIM17	16 位	向上	1-65536	支持	1	1

基本	**TIM6(1)**	**16 位**	**向上**	**1-65536**	**支持**	**0**	-

注：1 兼容 CS32F030C8

2.1 高级定时器（TIM1）

TIM1 属于高级定时器，具有 16 位自动重载向上/向下计数器和 16 位预分频器。该定时器具有 4 个独立通道可以用于：

- 输入捕获
- 输出比较
- 产生 PWM(边缘或中央对齐模式)
- 单脉冲输出

配置为 16 位标准定时器时，它与通用定时器具有相同的功能。配置为 16 位 PWM 发生器时，它具有全调制能力(0~100%) 。

在调试模式下，计数器可以被冻结。

很多功能都与标准的 TIM 定时器相同，内部结构也相同，因此高级控制定时器可以通过定时器链接功能与 TIM 定时器协同操作，提供同步或事件链接功能。

2.2 通用定时器（TIM2，3,14,15,16,17）

CS32FX031 内置了 6 个可同步运行的标准定时器。每个定时器都可以用于产生 PWM 输出，或者用作简单的定时器功能。

TIM2，TIM3

CS32FX031 提供两个可同步的四通道通用定时器。TIM2 基于一个 32 位自动重载向上/向下计数器和一个 16 位预分频器。TIM3 基于一个 16 位自动重载向上/向下计数器和一个 16 位预分频器。它们的四个通道可以独立地应用于输入捕获和输出比较，产生 PWM 和单脉冲模式输出功能。于是，在大封装的芯片中，可以提供 12 个输入捕获/输出比较通道/PWM 输出。

它们还能通过定时器链接功能与高级控制定时器共同工作，提供同步或事件链接功能。

它们两个都可以独立地产生 DMA 请求。

这些定时器还能够处理增量编码器的信号，也能处理 1 至 3 个霍尔传感器的数字输出。

在调试模式下，计数器可以被冻结。

TIM14

这个定时器基于一个 16 位自动重载向上计数器和一个 16 位预分频器。

TIM14 提供 1 个输入捕获/输出比较通道，PWM 或单脉冲输出。

在调试模式下，计数器可以被冻结。

TIM15,TIM16 和 TIM17

TIM15 / 16/17 定时器由一个可编程预分频器驱动的 16 位自动重载计数器组成。

它们分别有一个单通道用于输入捕获/输出比较，PWM 或单脉冲输出。

TIM16 和 TIM17 各有一个带死区时间插入的互补输出，能独立地产生 DMA 请求。

在调试模式下，计数器可以被冻结。

2.3 基本定时器（TIM6）

TIM6 是基于 16 位自动装载向上计数和 16 位预分频的定时器，没有输入输出通道，主要用于产生 DAC 触发事件或作为通用的 16 位时基使用。

3 基本定时器功能介绍

基本定时器的功能主要有两个：

- 基本定时功能，当累加的时钟脉冲超过预设定值时，就能触发中断或者 DMA 请求
- 专门用于驱动数模转换器

基本定时器的功能框图由时钟源、定时器以及计数器组成，如下图 3.1 所示。自动重装载寄存器、PSC 预分频器都存在一个阴影寄存器，该阴影寄存器可根据相关位的设定，在定时器更新（Update）事件（U 事件）时传送预装载寄存器至实际寄存器。该寄存器在物理上有两个寄存器构成：可以写入或读出的寄存器，称为预装载寄存器，另一个是看不见的、无法真正对其读写操作的，但是在使用 中真正起作用的寄存器，称为影子寄存器。

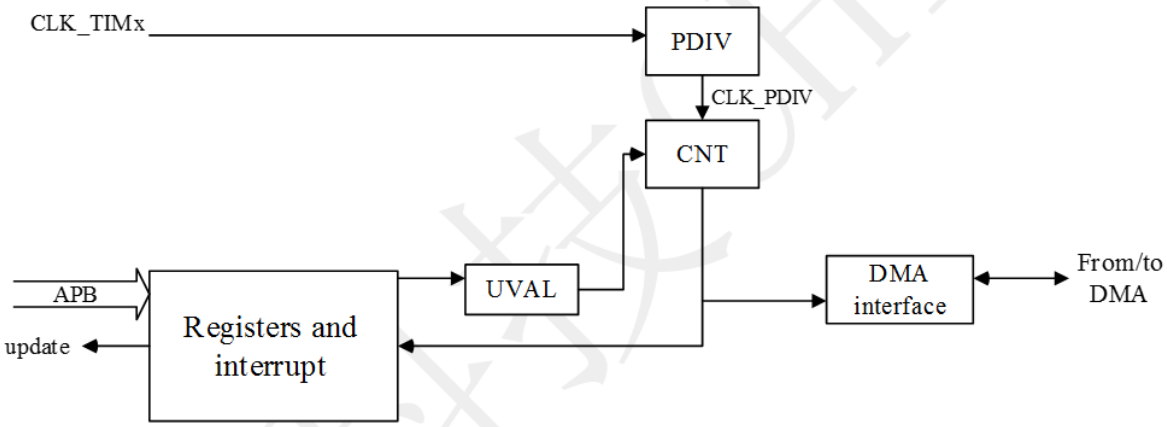

图 3.1 基本定时器功能框图

注：根据 TIMx_CTR1 寄存器中的 UVAL 影子寄存器使能位（UVALSEN），预载寄存器的内容将立刻传输到影子寄存器或每次更新事件 UVAL。当计数器达到溢出值并且 TIMx_CTR1 寄存器中的 UPD 位等于 0 时，将生成更新事件。

3.1 时钟源

基本定时器的时钟源只能来自内部时钟（CK_INT），预分频时钟 CK_ PDIV 由内部时钟 CK_INT 提供。当 TIMx_CTR1 寄存器的 CEN 位置 1 后，计数器开始使用内部时钟的分频时钟计数。

3.2 控制器

定时器控制器，包含一个触发输出控制器。控制器通过定时器配置寄存器（TIMx_CTR1）实现对定时器功能配置。控制器实现对定时器的复位、使能以及计数的控制。

触发控制器是专门用于控制定时器输出一个信号，这个信号可以输出到 CS32F0XX 内部其他外设（作为其他外设一个输入信号）。基本定时器的触发输出功能专门用于 ADC/DAC 转换触发启动。

3.3 计数器

定时器计数过程涉及三个寄存器：计数器寄存器（TIMx_CNT）、预分频寄存器（TIMx_PDIV）、计数器更新寄存器（TIMx_UVAL）。三个均是 16 位寄存器，可设置数值范围为 0~65535。

预分频器 PDIV 有一个输入时钟 CLK_TIMx 和一个输出时钟 CLK_PDIV。输出时钟来源于控制器部分（即 CLK_TIMx），通过设置预分频的数值，可以得到不同的 CLK_PDIV，计算公式为：CLK_PDIV=CLK_TIMx/(PDIV[15:0]+1)。

按照设置的不同，预装载寄存器的内容可以被立即或在每次更新事件（UVAL）产生时传送到影子寄存器。当自动预装载使能（TIMx_CTR1 寄存器 UVALSEN=1）时，写入自动装载寄存器中的数据将被保存在预装载寄存器中，并在下一个更新事件时传送到影子寄存器并立即生效；当自动预装载禁止时（UVALSEN=0），写入到自动装载寄存器的数据将立即传送到影子寄存器。

更新事件（UVAL）是由计数器达到溢出条件（上溢），且在更新事件允许（TIMx_CTR1 寄存器的 UPD=0）时产生的。更新事件也可以由软件定时器复位时产生。

4 高级定时器功能介绍

TIM 定时器由一个 16 位向上、向下、中央对齐自动装载计数器构成，具有 16 位可编程、可实时配置的预分频器，预分频系数可以在 1~65535 的任意数值调整。TIM1 定时器具有以下的功能：

- ADC 与 DAC 开始转换触发
- 输入捕获：脉冲计数、上升沿或下降沿时间检测、PWM 输入检测
- 输出比较：脉冲输出、步进电机控制
- 脉冲宽度调节 PWM：电压输出控制、直流减速电机控制、直流无刷电机控制
- 单脉冲输出模式
- 编码器接口、霍尔传感器接口
- 位于高速的 APB 总线上，内部时钟最高可达 48MHz
- 16 位自动重载计数器和预分频器
- 4 个 16 位高精度捕捉比较通道
  - 可编程设置通道方向：输入/输出
    1. 输入捕获、PWM 输入捕获
    2. 输出比较
    3. 脉冲调制宽度 PWM
    4. 单脉冲模式
- 使用外部信号控制定时器且可实现多个定时器互连的同步电路
  - 定时器主设备/从设备同步
  - 与外部触发同步
  - 触发或门控模式
- 重复计数器：用于仅在给定数目的计数器周期后更新定时器寄存器
- 死区时间：可编程的互补输出（通道和互补通道）
- 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
- 针对定位的增量（正交）编码器和霍尔传感器接口
- 独立的 IRQ/DMA 请求生成器
  - 更新事件：计数器上溢/下溢、计数器初始化（通过软件或内部/外部触发）
  - 触发事件（计数器启动、停止、初始化或通过内部/外部触发计数）
  - 输入捕获
  - 输出比较
  - 刹车信号输入

TIM1 内部结构如下图 4-1 所示，包含时钟源、控制器、时机单元、输入捕获、输出比较以及刹车功能 6 个部分组成。

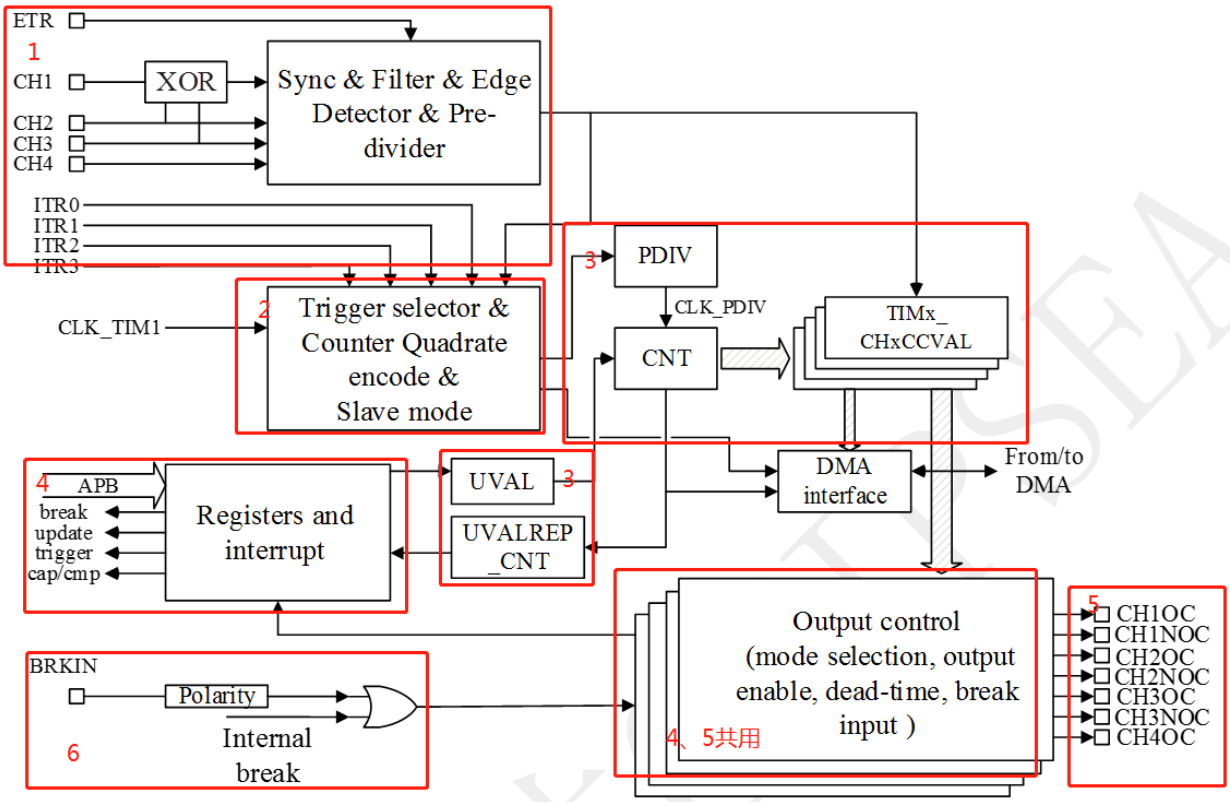

图 4.1 高级定时器功能框图

4.1 时钟源

高级定时器有 4 个时钟源

- **内部时钟源 CK_INT**
- **外部时钟模式 1：外部通道输入引脚 TINx（x=1,2）**
- **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**

时钟源的结构框图如图 4.2 所示。

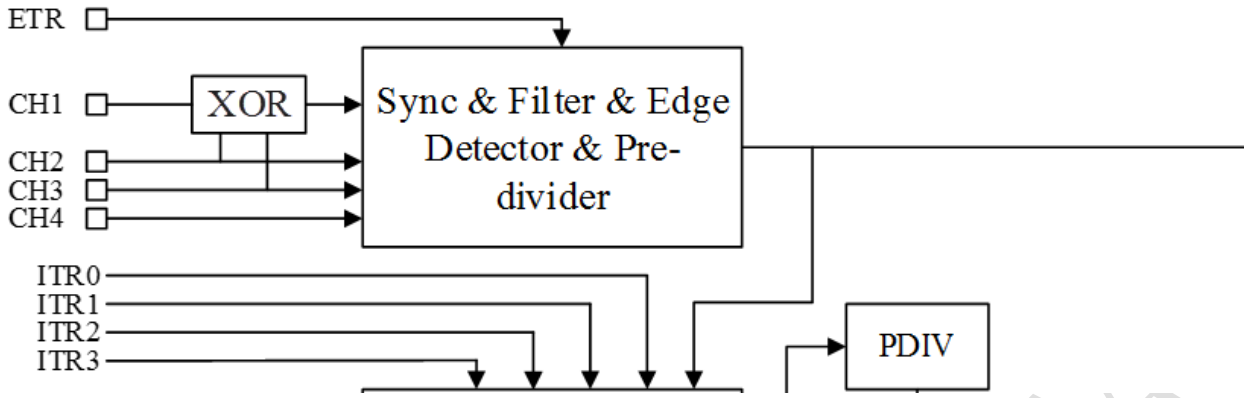

图 4.2 高级定时器时钟源

4.1.1 内部时钟源 CK_INT

当从模式控制寄存器 TIMx_SMCFG 的 SMCFG=000 时，内部时钟 CK_INT 即来自于 CS32 芯片内部 APB 总线，大多数的控制电路中，内部时钟分频因子为 1，则 TIM1 的时钟为 48MHz。

如下图 4.3 中显示了当预分频系数为 1 且 TIM1_UVAL 寄存器的值为 “0x0018” 时，向上计数的时序。图中计数器在每 1 个预分频时钟时计数 1 次，当计数值达到 “0x0018” 时发生计数溢出，计数值清 0 并产生更新事件，同时更新中断标志位将被设置。

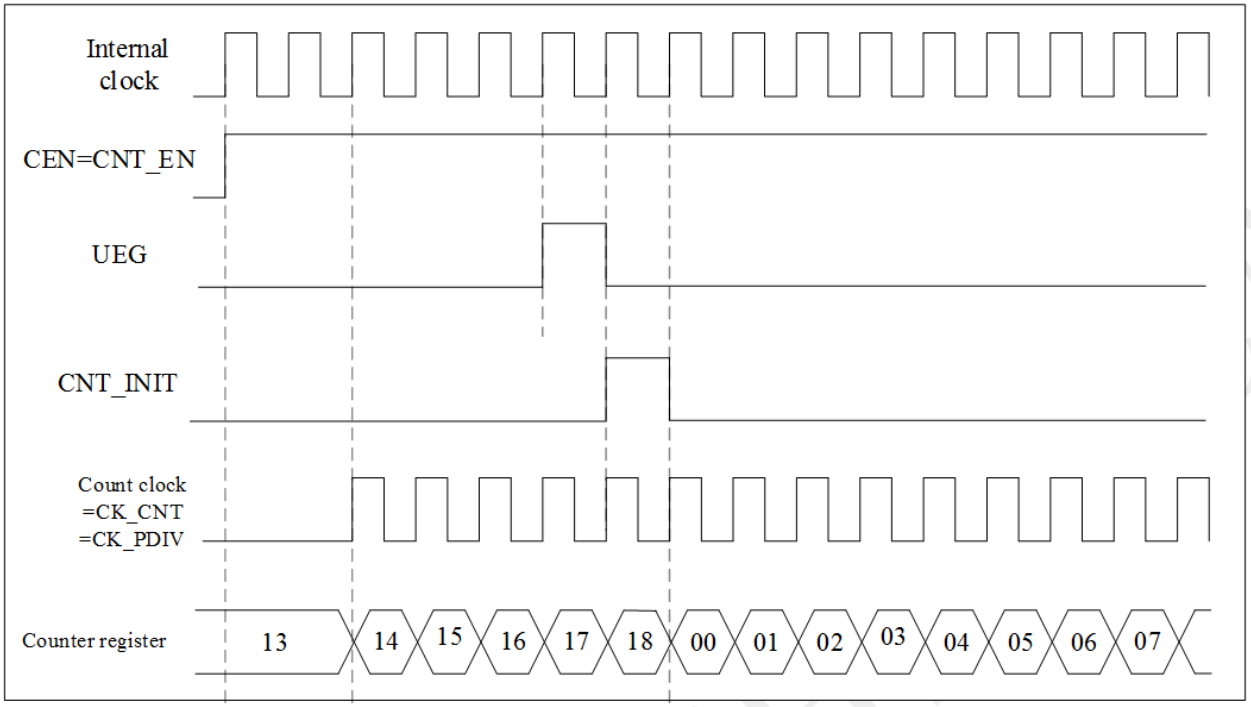

图 4.3 高级定时器时序图-内部时钟分频因子为 1

当预分频器分频系数从 1 变为 2 时的计数器时序图，如下图 4.4 所示：

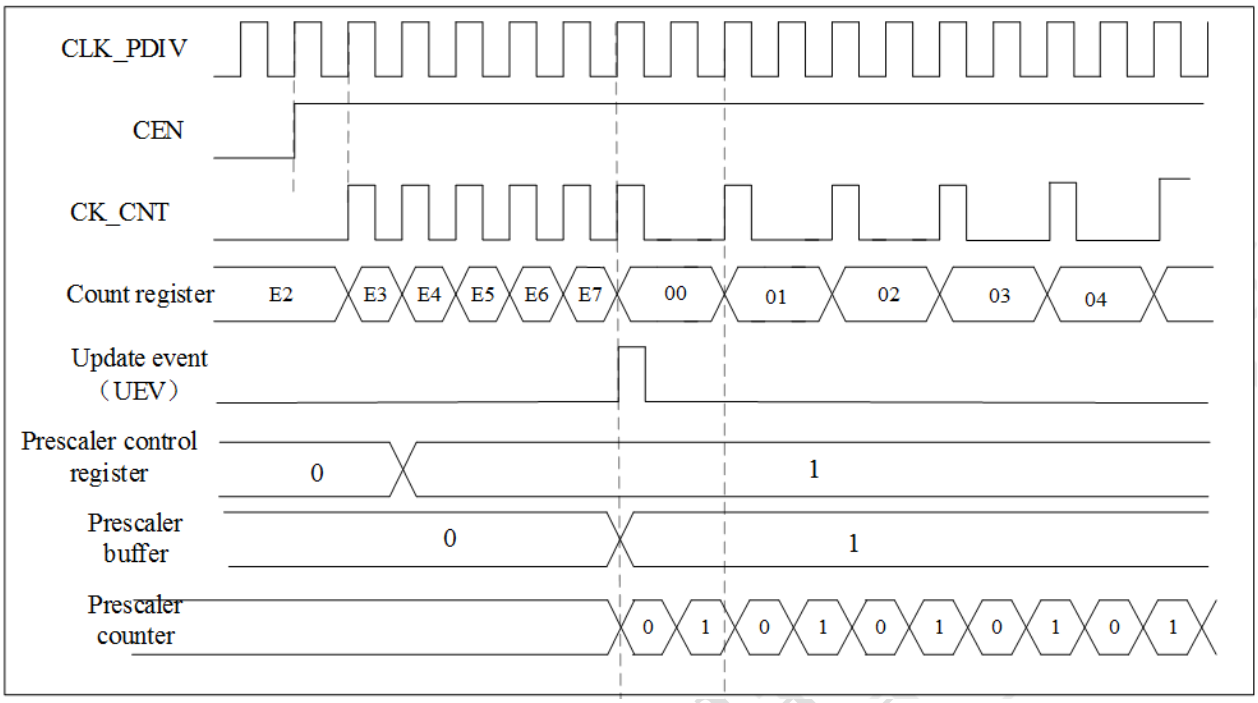

图 4.4 高级定时器预分频器分频系数从 1 变为 2 时的计数器时序图

当预分频器分频系数从 1 变为 2 时的计数器时序图，如下图 4.5 所示：

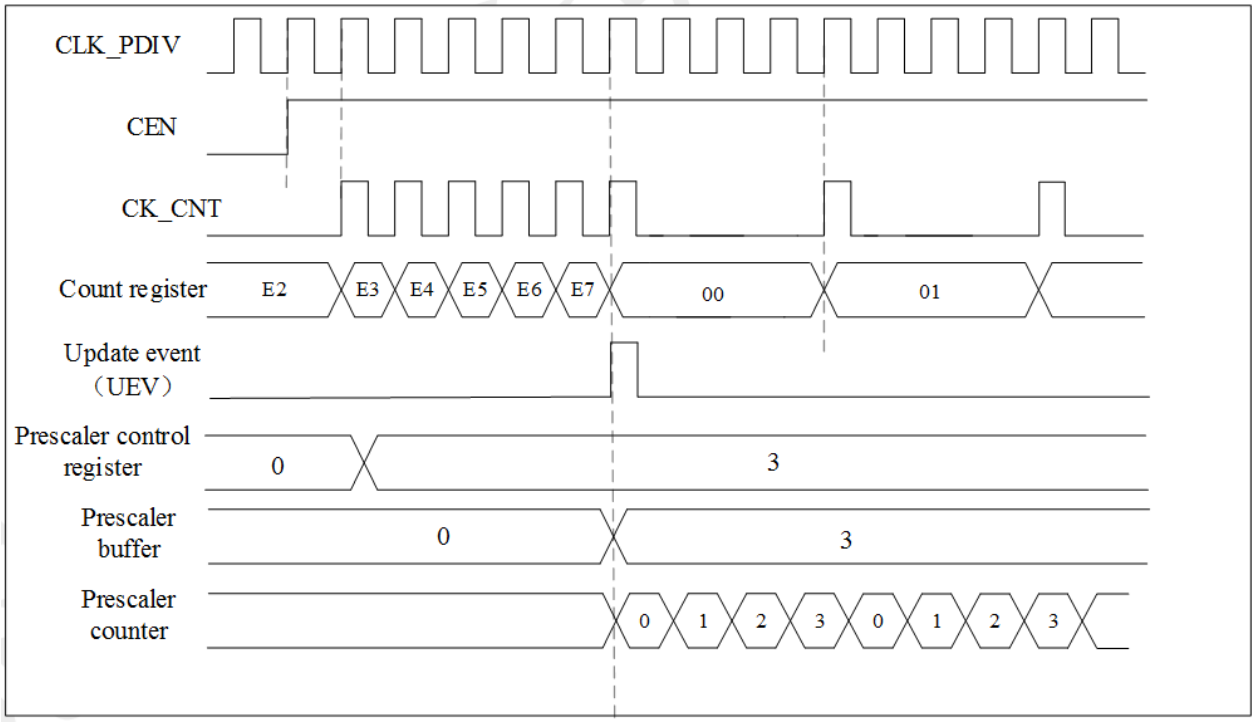

图 4.5 高级定时器预分频器分频系数从 1 变为 4 时的计数器时序图

4.1.2 外部时钟模式 1

当 TIM1_SMCFG 寄存器的 “SMCFG=111 ” 时，计时器在选定输入端的每个上升沿或者下降沿计数。外部时钟模式 1 结构框图如下图 4.6 所示。

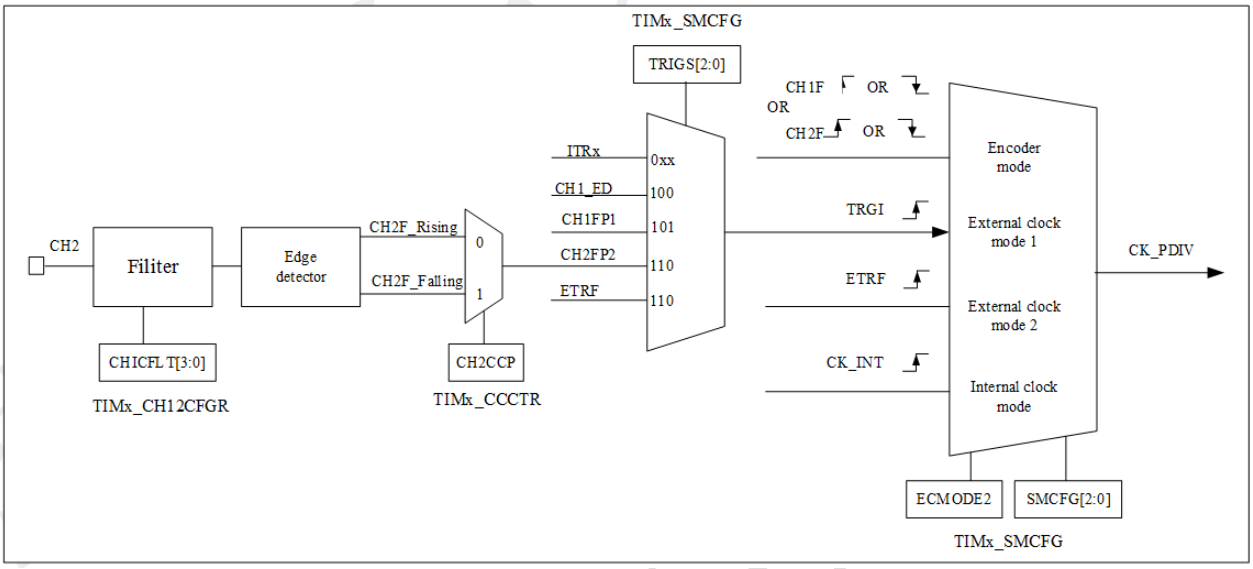

图 4.6 CH2 外部时钟连接示例

①时钟输入信号引脚

使用外部时钟模式 1 时，有 2 个来自于定时器输入通道的时钟信号，分别是 CH1FP1 和 CH2FP2，CH1FP1 对应 CH1（TIMx_CH1）,CH2FP2 对应 IC2（TIMx_CH2），配置 TIMx_CH12CFGR 位的 CCxFS[1:0]来选择那一路的信号。另外，CH3 与 CH4 不能作为外部时钟模式 1 的引脚。

注意：定时器的多种高级功能是需要外部引脚配合使用，比如输入捕获（脉冲计数、上升沿或下降沿检测、PWM 输入检测）、输出比较（脉冲输出、步进电机控制）、脉冲宽度调节 PWM（电压输出控制、直流减速电机控制、直流无刷电机控制）、编码器接口、霍尔传感器接口。IO 作为定时器功能需配置为 IO 复用模式。

②滤波器

这里可以通过寄存器配置不同的滤波时间，从而滤除掉输入信号上的高频干扰。

③边沿检测

为了检测滤波后的信号，通过配置 TIMx_CCCTR 的位 CHxCCP 和 CHxCCEN，检测上升沿有效还是下降沿有效。

CH1CCP：捕获/比较 1 输出极性

CH1 通道配置为输入：

CC1NP/CH1CCP 位可针对触发或捕获操作选择 CH1FP1 和 CH2FP2 的极性。

00：非反相/上升沿触发

电路对 CH2FP1 上升沿敏感（在复位模式、外部时钟模式或触发模式下执行捕获或触发操作），CH2FP1 未反相（在门控模式或编码器模式下执行触发操作）。

01：反相/下降沿触发

电路对 CH2FP1 下降沿敏感（在复位模式、外部时钟模式或触发模式下执行捕获或触发操作），CH2FP1 反相（在门控模式或编码器模式下执行触发操作）。

10：保留，不使用此配置。

11：非反相/上升沿和下降沿均触发

电路对 CH2FP1 上升沿和下降沿都敏感（在复位模式、外部时钟模式或触发模式下执行捕获或触发操作），CH2FP1 未反相（在门控模式下执行触发操作）。编码器模式下不得使用此配置。

④触发选择

若选择外部时钟模式 1，此时有两个触发源，一个是滤波后的定时器输入 1（CH1FP1）和滤波后的定时器输入 2（CH2FP2），具体的由 TIMx_SMCFG 位的 TRIGS[2:0]配置。

⑤从模式选择

选定触发源后，信号连接到 TRGI 引脚，配置 TIMx_SMCFG 寄存器的 SMCFG=111，配置触发信号为外部时钟模式 1 的输入，即 CK_PDV。

编程实例：要配置向上计数器来响应 CH2 输入的上升沿进行计数

- 配置 TIMx_CH12CFGR 寄存器中写入 CH2FS = 01 将通道 2 配置为检测 CH2 输入上升沿
- 配置 TIMx_CH12CFGR 寄存器中的 CH2ICFLT [3：0]位来配置输入滤波器持续时间（如果不需要滤波器，则保持 CH2ICFLT= 0000）
- 配置 TIMx_CCCTR 寄存器中写入 CH2CCP = 0 和 CH2NCCP = 0 来选择上升沿极性
- 配置 TIMx_SMCFG 寄存器中写入 SMCFG = 111，在外部时钟模式 1 下配置定时器
- 配置 TIMx_SMCFG 寄存器中写入 TRIGS= 110，选择 CH2 作为输入源
- 设置 TIMx_CTR1 寄存器的 CEN=1，启动计数器

如下图 4.7 所示，当 CH2 上出现上升沿时，计数器计数一次，并且 TRIGIF 标志被设置。CH2 的上升沿与计数器的实际时钟之间的延迟是由 CH2 输入上的重新同步电路引起的。

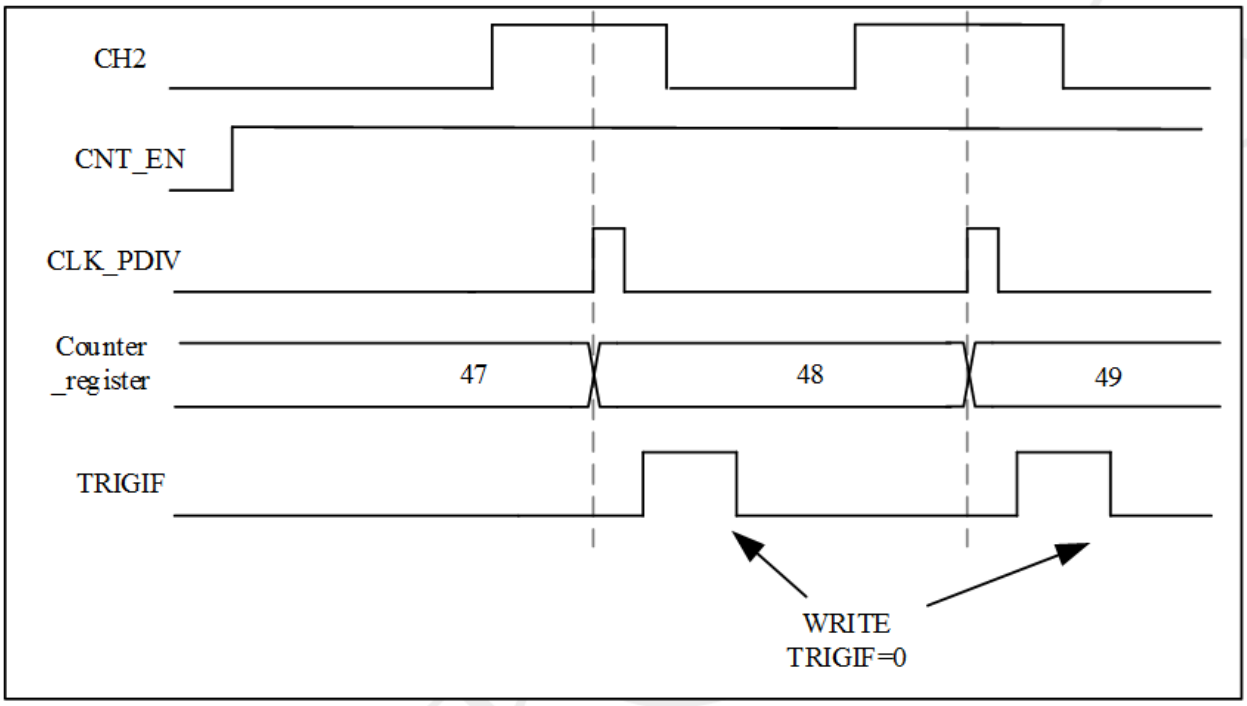

图 4.7 CH2 上升沿计数时序图

4.1.3 外部时钟模式 2

通过在 TIMx_SMCFG 寄存器中写入 ECMODE2 = 1 来选择该模式。计数器可以在外部触发输入 ETR 的每个上升沿或下降沿计数。外部时钟模式 2 的原理图如下图 4.8 所示。

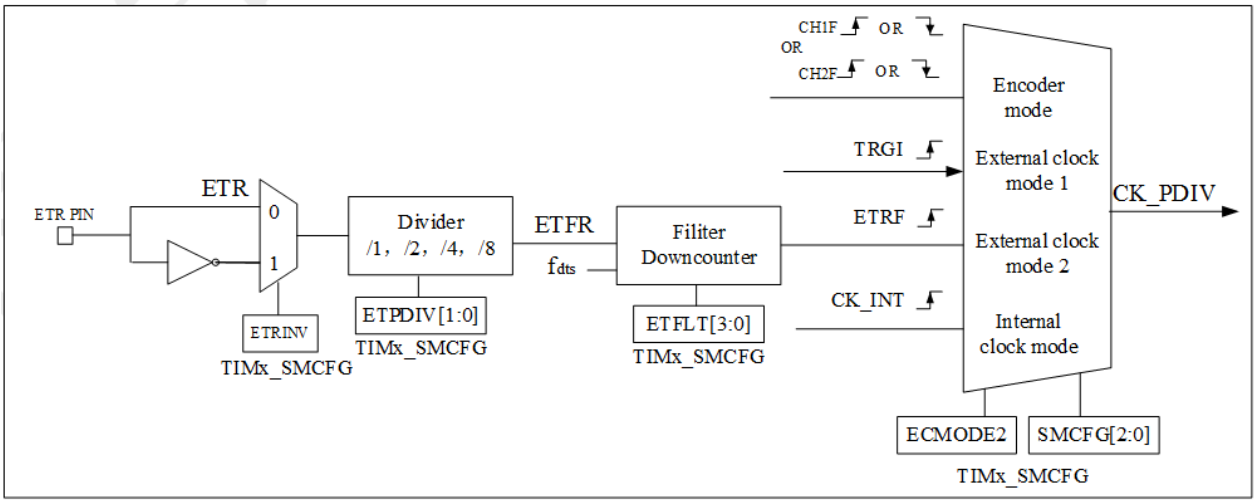

图 4.8 外部时钟模式 2 时序图

编程实例：配置 ETR 下每 2 个上升沿计数一次

- **不使用滤波器，置 TIMx_SMCFG 寄存器中的** ETFLT **[3:0]=0000**
- **设置预分频器，置 TIMx_SMCFG 寄存器中的** ETPDIV **[1:0]=0，1,2 分频**

- 选择在 TIMx_SMCFG 中写入 ETRINV= 0，在 ETR 引脚上选择上升沿检测
- 在 TIMx_SMCFG 寄存器中写入 ECMODE2 = 1 来使能外部时钟模式 2
- 启动计数器，写 TIMx_CTR1 寄存器中的 CEN=1

如下图 4.9 所示，计数器在每 2 个 ETR 上升沿计数一次。在 ETR 上升沿和计数器实际时钟之间的延时取决于在 ETRP 信号端的重新同步电路。

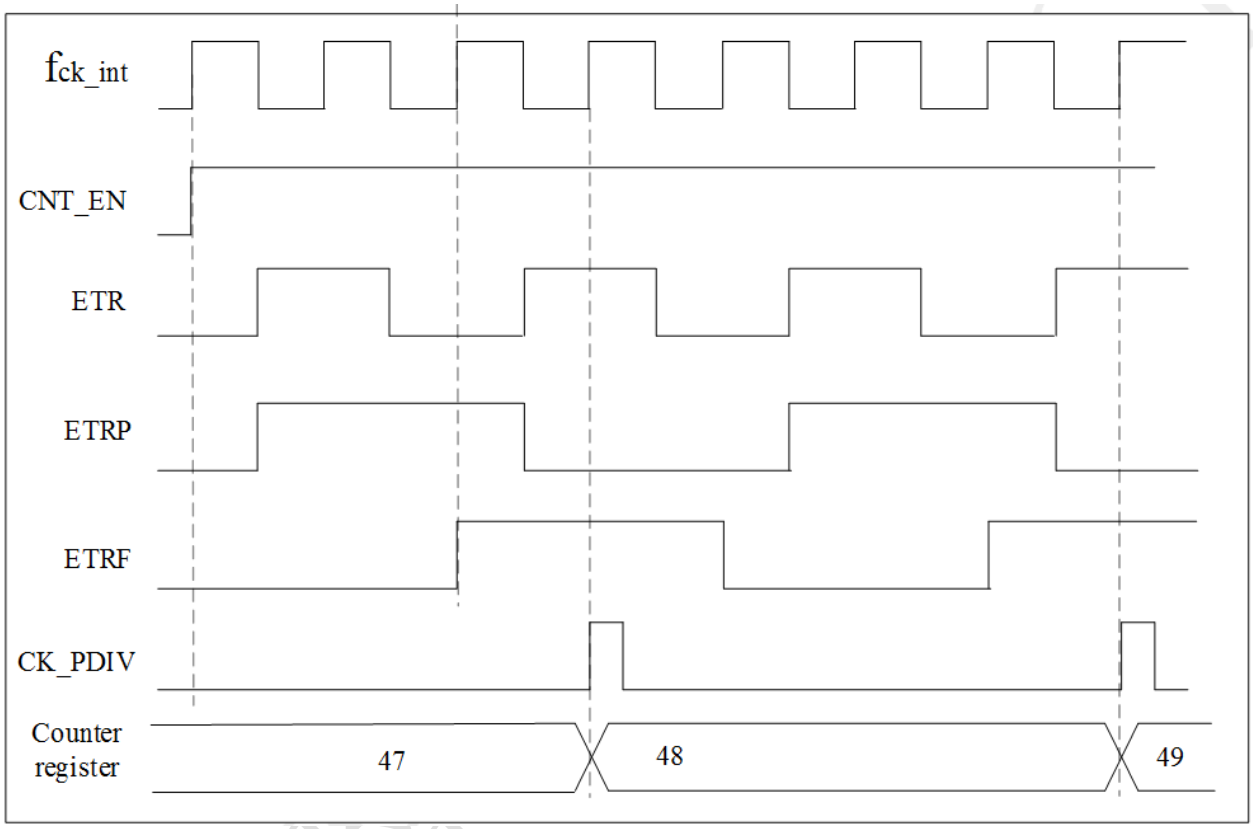

图 4.9 ETR 上升沿计数时序图

4.1.4 内部触发输入（ITRx）

TIMx 定时器从内部连接在一起，以实现定时器同步或级联。当某个定时器配置为主模式时，可对另一个配置为从模式的定时器的计数器执行复位、启动、停止操作或为其提供时钟。例如，可以将定时器 1 配置为定时器 2 的预分频器。

4.2 控制器

控制器结构原理如下图 4.10 所示，由三部分组成：

- 触发控制器：用来针对片内外设输出触发信号，为其它定时器提供时钟和触发 DAC/ADC 转换
- 编码器接口：专门针对编码器计数而设计
- 从模式控制器：可以控制计数器复位、启动、递增/递减、计数

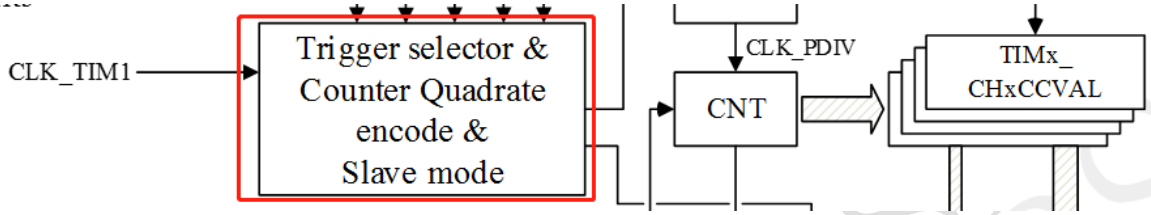

图 4.10 控制器结构图

4.3 时基单元

高级定时器是一个 16 位计数器及其相关的自动重载寄存器组成。计数器可递增计数、递减计数或交替进行递增和递减计数。计数器的时钟可通过预分频器进行分频。计数器、自动重载寄存器和预分频寄存器即使在计数器运行中也可通过软件进行读写。时基单元的结构图如下图 4.11 所示。时基单元主要包括以下四个部分：

- 计数器寄存器（TIMx_CNT）
- 预分频寄存器（TIMx_ PDIV）
- 自计数器更新寄存器（TIMx_UVAL）
- 重复计数器寄存器（TIMx_UVALREP）

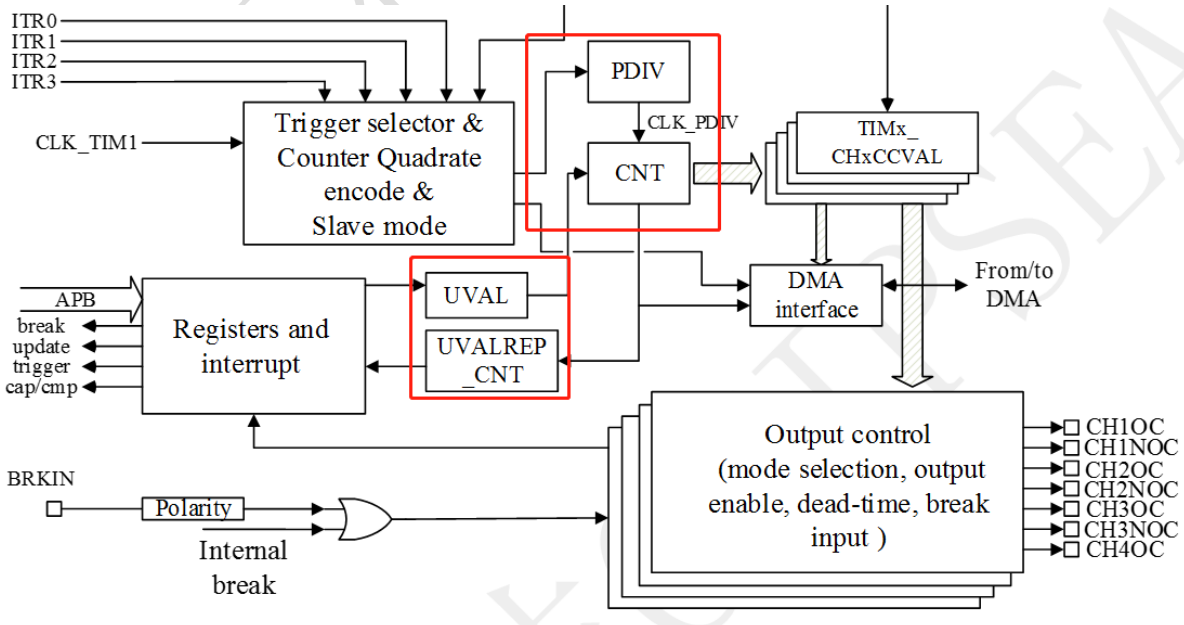

图 4.11 时基单元结构图

4.3.1 预分频器

预分频器 PDIV 有一个输入时钟 CLK_TIM1 和一个输出时钟 CLK_PDIV。输入时钟来源于控制器部分，通过设置预分频的数值，可以得到不同的 CLK_PDIV，计算公式为：

CLK_PDIV= CLK_TIM1/(PDIV [15:0]+1)，因为 TIMx_PDIV 控制器具有缓冲（影子寄存器），可以在运行过程中改变数值，新的预分频数值将在下一个更新事件时起作用。

如下图 4.12 所示，当预分频值 PDIV[15:0]=0 时，分频比为 1，预分频时钟与计数器时钟相同。当在 TIM1_PSC 寄存器中写入新的预分频值（PDIV[15:0]=3）时，分频比并没有改变而是在更新事件到来时分频比才从 1 变成 4。

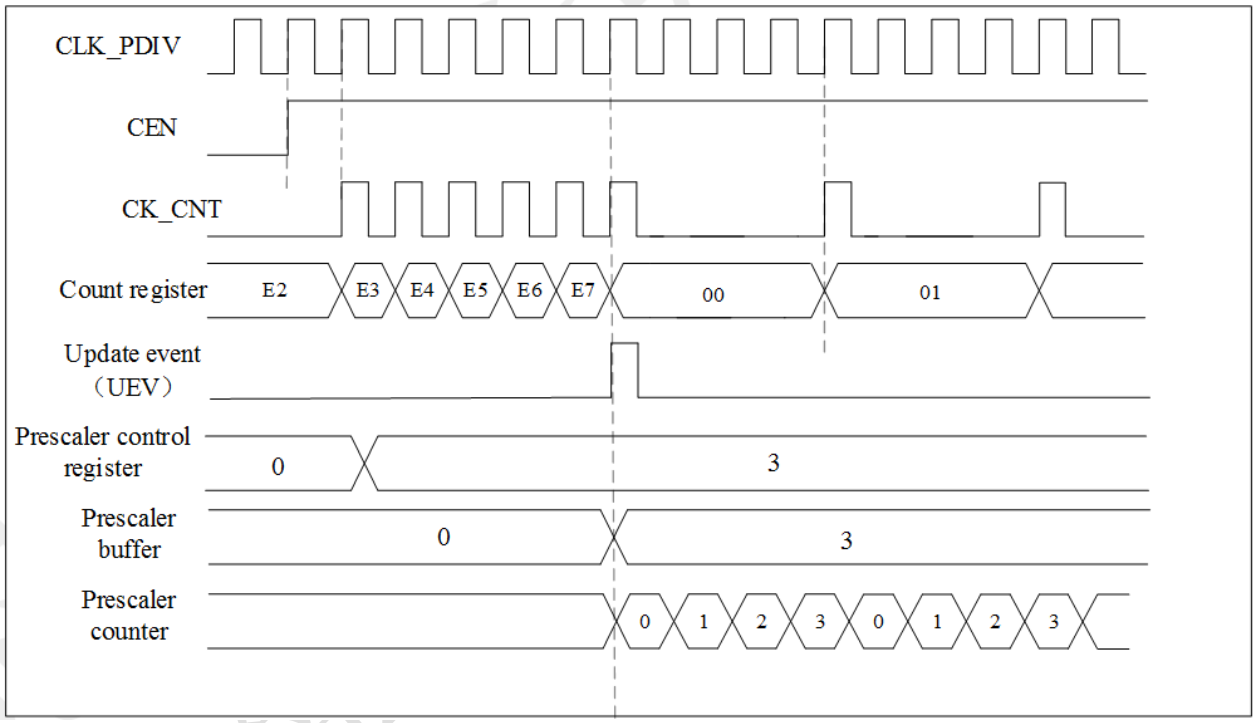

图 4.12 预分频器分频由 1 变成 4 时的计数器时序图

4.3.2 计数器

计数器的计数模式可分为递增、递减以及中央对齐。每当产生上溢或下溢事件时将产生更新事件，如下图 4.13 所示。

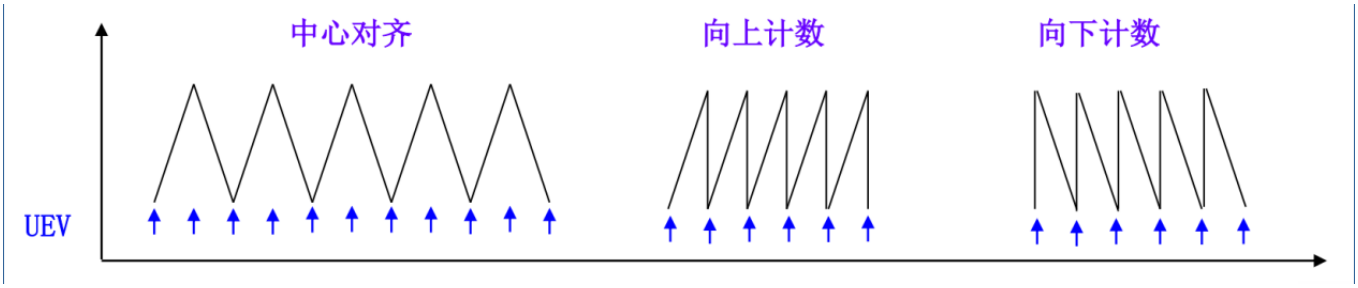

图 4.13 更新事件图

向上计数模式：

在向上计数模式下，计数器从 0 开始计数到自动重载值（TIMx_UVAL 寄存器的值），然后从 0 重新开始并产生计数器溢出事件。

如果使用重复计数功能，在计数达到重复计数器寄存器（TIMx_UVALREP）中次数之后将产生更新事件（UEV）。否则，每次计数器溢出时都会生成更新事件。

将 TIMx_SWEGR 寄存器中的 UEG 位（通过软件或使用从模式控制器）置 1 也会产生更新事件。

通过设置 TIMx_CTR1 寄存器中的 UPD 位，可以通过软件禁用 UEV 事件。这是为了避免在预加载寄存器中写入新值时更新影子寄存器。在 UPD 位写入 0 之前不会发生更新事件。然而，计数器从 0 重新开始，以及预分频器的计数器（但预分频率不会改变）。此外，如果 TIMx_CTR1 寄存器中的 URSEL 位（更新请求选择）已设置，则设置 UEG 位会生成更新事件 UEV 但不设置 UPIF 标志（因此不会发送中断或 DMA 请求）。这是为了避免在捕获模式下清除计数器时，同时产生更新和捕获中断。

发生更新事件时，将更新所有寄存器并依据 URSEL 设置更新标志（TIMx_STS 寄存器中的 UPIF 位）：

- 重复计数器重新加载 TIMx_UVALREP 寄存器的内容
- 使用预加载值（TIMx_UVAL）更新自动重载影子寄存器
- 预分频器的缓冲区重载了预载值（TIMx_PDIV 寄存器的内容）

下图显示了 TIMx_UVAL = 0x18 时不同时钟频率的计数器行为的一些示例。

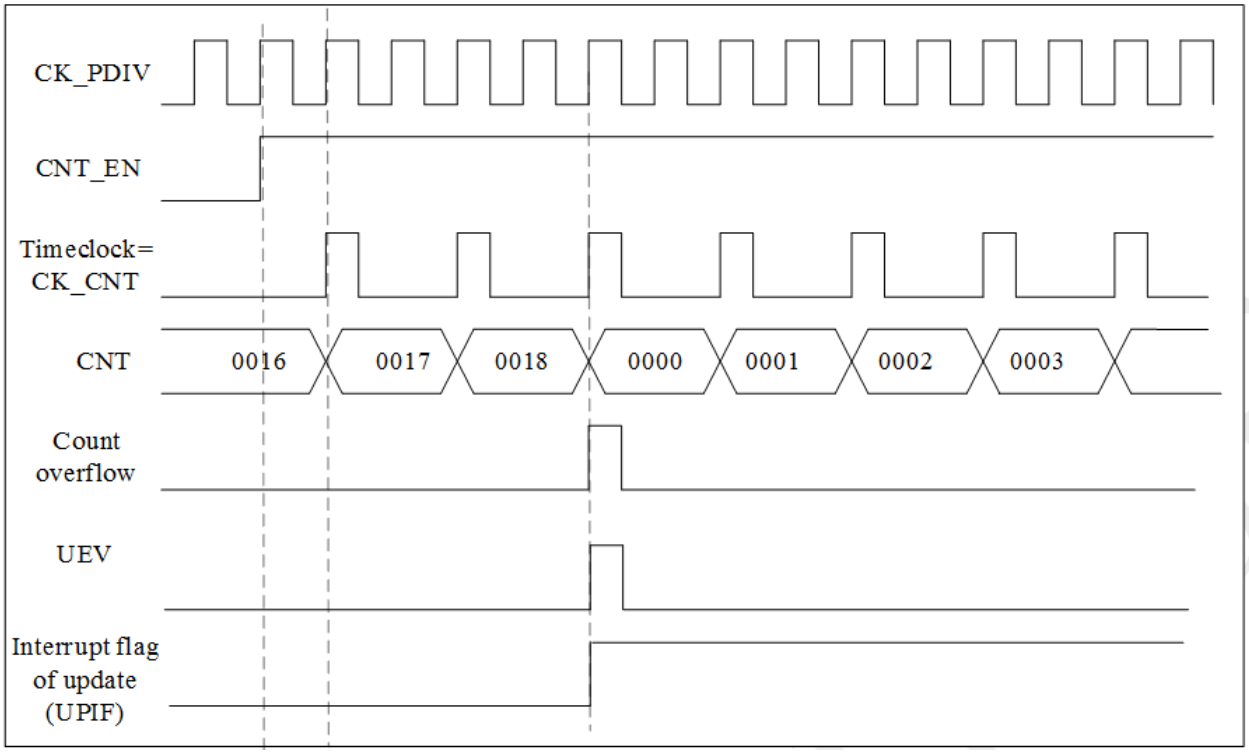

图 4.14 向上计数模式内部分频为 2 时序图

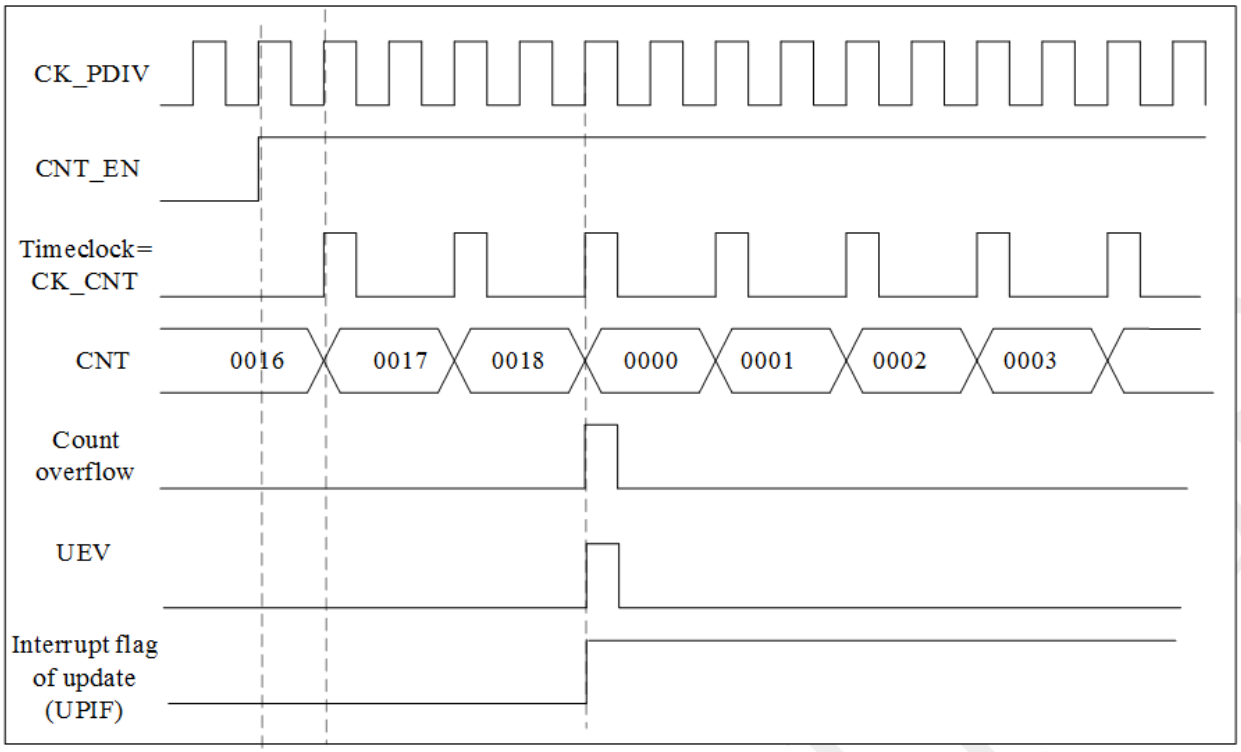

图 4.15 向上计数模式内部分频为 4 时序图

向下计数模式：

在向下计数模式下，计数器从自动重载值（TIMx_UVAL 寄存器的内容）向下计数到 0，然后从自动重载值重新启动并生成计数器下溢事件。

如果使用重复计数器，则在重复计数器寄存器（TIMx_UVALREP）中的次数之后产生更新事件（UEV）。否则，在每个计数器下溢时生成更新事件。将 TIMx_SWEGR 寄存器中的 UEG 位（通过软件或使用从模式控制器）置 1 也会产生更新事件。

通过将 TIMx_CTR1 寄存器中的 UPD 位置 1，可以通过软件禁用 UEV 更新事件。这是为了避免在预加载寄存器中写入新值时更新影子寄存器。在 UPD 位写入 0 之前不会发生更新事件。但是，计数器从当前自动重载值重新开始，而预分频器的计数器从 0 重新开始（但预分频率不会改变）。

此外，如果 TIMx_CTR1 寄存器中的 URSEL 位（更新请求选择）已设置，则设置 UEG 位会生成更新事件 UEV 但不设置 UPIF 标志（因此不会发送中断或 DMA 请求）。这是为了避免在清除捕获模式下的计数器同时生成更新和捕获中断。发生更新事件时，将更新所有寄存器并依据 URSEL 设置更新标志（TIMx_STS 寄存器中的 UPIF 位）。

- 重复计数器重新加载 TIMx_UVALREP 寄存器的内容
- 预分频器的缓冲区重载了预载值（TIMx_PDIV 寄存器的内容）
- 使用预装载值（TIMx_UVAL 寄存器的内容）更新自动重载激活寄存器。

下图显示了 TIMx_UVAL = 0x18 时不同时钟频率的计数器行为的一些示例：

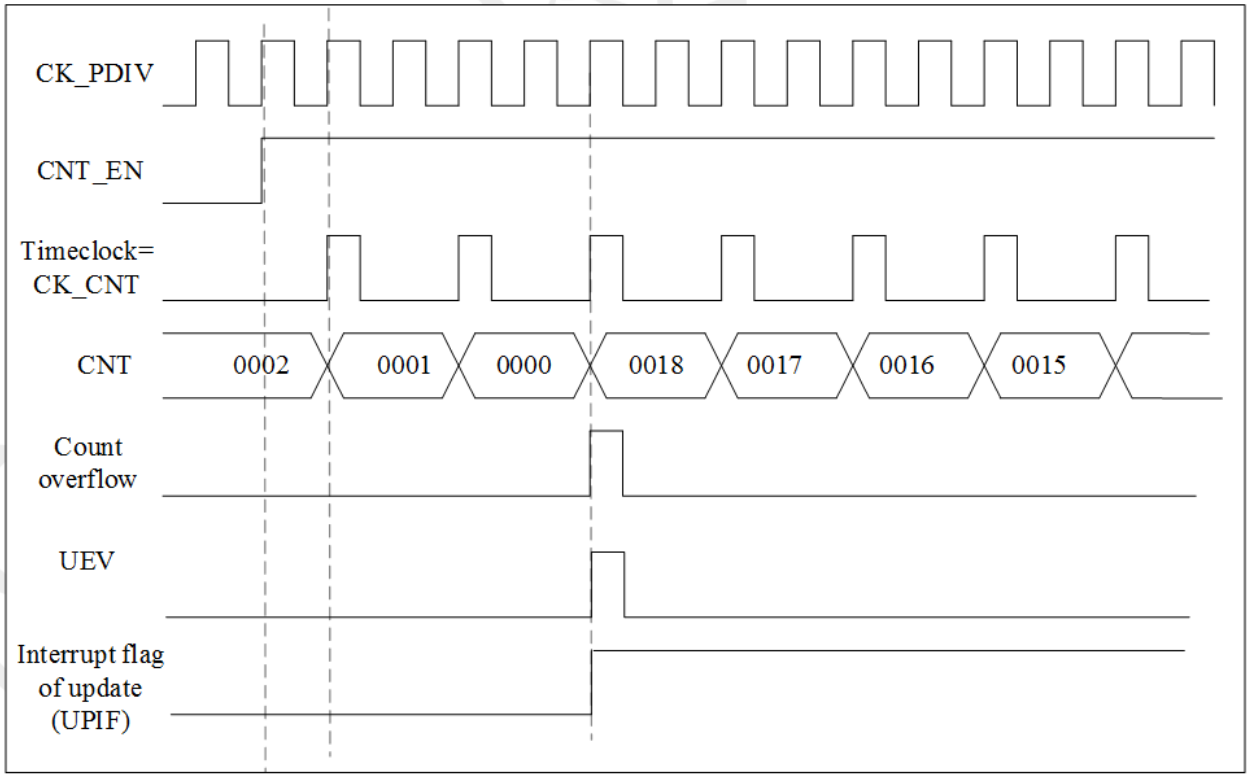

图 4.16 向下计数模式分频系数为 2 时序图

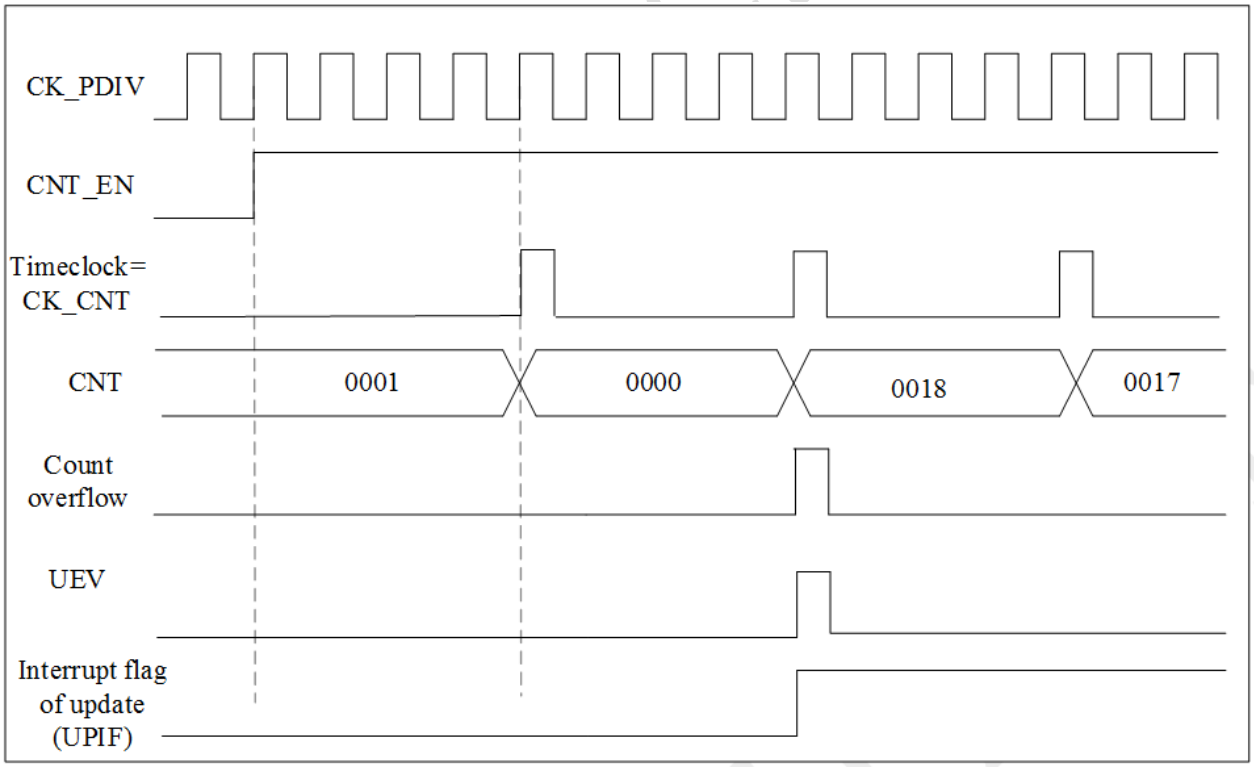

图 4.17 向下计数模式分频系数为 4 时序图

中央对齐计数模式：

在中央对齐模式下，计数器从 0 计数到自动重载值（TIMx_UVAL 寄存器的内容）-1，产生计数器溢出事件，然后从自动重载值向下计数到 1 并生成计数器下溢事件。然后重新从 0 开始计数。当 TIMx_CTR1 寄存器中的 CPS 位不等于 00 时，中央对齐模式有效。输出中配置的通道的输出比较中断标志在以下情况下设置：计数器向下计数时（中央对齐模式 1，CPS = “ 01 ”），计数器向上计数（中央对齐模式 2，CPS = “10 ”）计数器向上和向下计数（中央对齐模式 3，CPS = “11 ”）。

此模式下，无法写入 TIMx_CTR1 寄存器中的 DIR 方向位。它由硬件更新并给出计数器的当前方向。可以在每个计数器溢出和每个计数器下溢时生成更新事件，或者通过设置 TIMx_SWEGR 寄存器中的 UEG 位（通过软件或使用从模式控制器）也会生成更新事件。在这种情况下，计数器从 0 开始计数，预分频器的计数器也是如此。

通过设置 TIMx_CTR1 寄存器中的 UPD 位，可以通过软件禁用 UEV 更新事件。这是为了避免在预加载寄存器中写入新值时更新影子寄存器。在 UPD 位写入 0 之前不会发生更新事件。但是，计数器会根据当前的自动重载值继续向上和向下计数。

此外，如果 TIMx_CTR1 寄存器中的 URSEL 位（更新请求选择）已设置，则设置 UEG 位会生成 UEV 更新事件，但不会设置 UPIF 标志（因此不会发送中断或 DMA 请求）。这是为了避免在捕获模式下清除计数器时同时产生更新和捕获中断。发生更新事件时，将更新所有寄存器并依据 URSEL 设置更新标志（TIMx_STS 寄存器中的 UPIF 位）。

- **重复计数器重新加载 TIMx_UVALREP 寄存器的内容**
- **预分频器的缓冲区重载了预载值 TIMx_PDIV 寄存器的内容**

下图显示了不同时钟频率的计数器行为的一些示例：

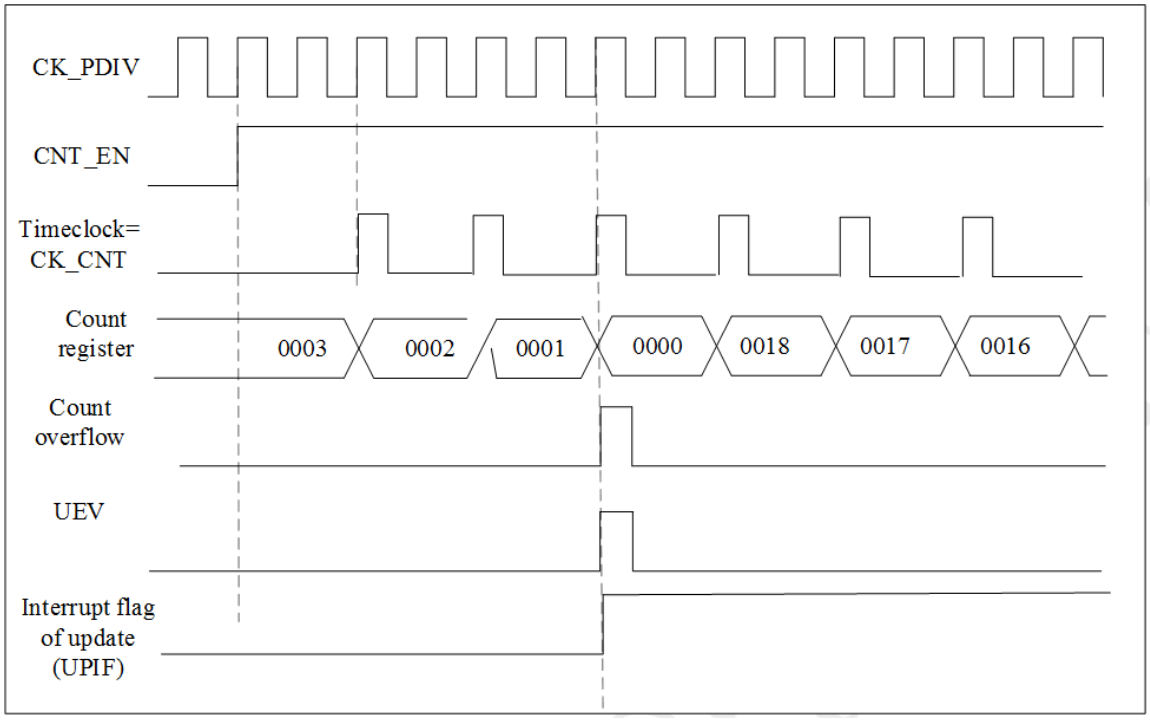

图 4.18 中央对齐计数模式分频系数为 2 时序图

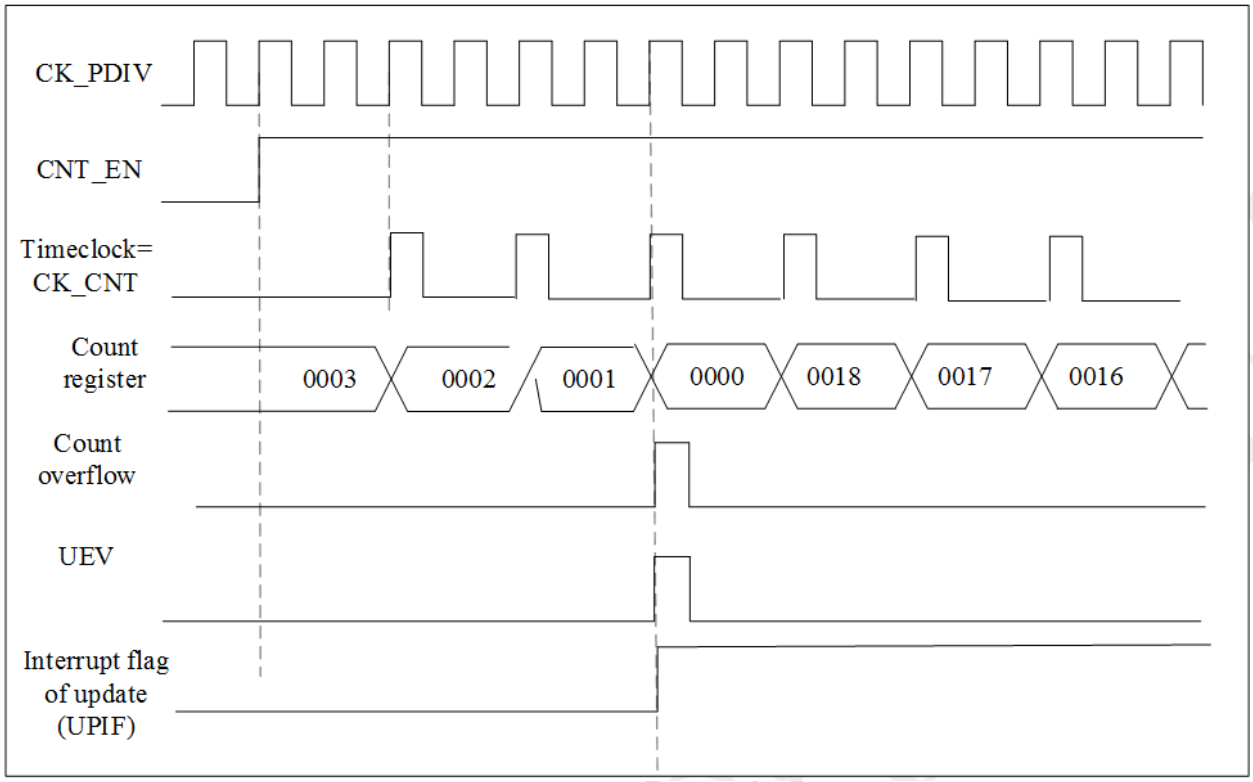

图 4.19 中央对齐计数模式分频系数为 4 时序图

4.4 输入捕获

4.4.1 输入捕获通道结构

输在输入捕捉模式下，在相应 CHx 信号检测到数据变化后,将计数器的值锁存到捕捉/比较寄存器（TIMx_CH1CCVAL）。发生捕获时，会设置相应的 CHxCCIF 标志（TIMx_STS 寄存器），如果使能了中断或 DMA 请求，则可以产生中断和 DMA 请求。如果在 CH1CCIF 标志已经为高电平时发生捕获，则设置过捕获标志 CH1ICOF（存在于 TIMx_STS 寄存器中）。 CHxCCIF 可以通过软件将其写入 0 或读取存储在 TIMx_CHxCCVAL 寄存器中的捕获数据来清除。当将其写入 0 时，CHxCCOF 将被清除。

编程实例：在 CH1 输入上升沿时捕获计数器的值到 TIMx_ CH1CCVAL 寄存器

- 选择有效输入：TIMx_CH1CCVAL 必须链接到 CH1 输入，因此在 TIMx_CH12CFGR 寄存器中将 CH1FS 位写入 01。一旦 CH1FS 不是 00，通道被配置成输入，TIMx_CH1CCVAL 寄存器变为只读
- 根据连接到定时器的信号配置所需的输入滤波器持续时间（TIMx_CHxxCFGR 寄存器中的 CHxICFLT 位），例如，当数据翻转时，输入信号在 5 个内部时钟周期时不稳定。
- 通过将 TIMx_CCCTR 寄存器中的 CH1CCP 和 CH1NCCP 位写入 0（在本例中为上升沿），选择 CH1 通道上有效转换的边沿
- 配置输入预分频器。在我们的示例中，我们希望在每次有效转换时执行捕获，因此禁用预分频器（在 TIMx_CH12CFGR 寄存器中将 CH1ICPS 位写入 00）
- 通过将 TIMx_CCCTR 寄存器中的 CH1CCEN 位置 1，使从计数器可以捕获到捕获寄存器
- 如果需要，通过设置 TIMx_DIEN 寄存器中的 CH1INTEN 位来使能中断请求，通过将 TIMx_DIEN 寄存器中的 CH1DEN 位置 1 来使能相关的 DMA 请求

注意：①发生有效跳变沿时，TIMx_CH1CCVA 寄存器会获取当前计数器的值

②若发生了中断 CH1CCIF 标志置 1，如果发生了两次连续捕获，但 CH1CCOF 标志未清零，这样 CH1CCOF 捕获溢出标志会置 1

③根据 CH1INTEN 位生成中断

④根据 CH1DEN 位生成 DMA 请求

⑤处理重复捕获，建议在读出捕获溢出标志之前读取数据。这样可避免在读取捕获溢出标志之后与读取数据之前可能出现的重复捕获信息

4.4.2 PWM 输入模式

PWM：脉冲宽度调制，即占空比可变的脉冲波形。

PWM 输入模式就是用定时器功能测量 PWM 波的周期和占空比。定时器有 PWM 输入这个模式，用于检测 PWM 波形。PWM 输入模式时定时器输入捕获一种特例，普通的输入捕获功能只能测量信号的周期。

以输入通道 CH1 为例，PWM 信号由输入通道 CH1 进入，信号分为两路，一路是 CH1FP1，另一路是 CH1FP2，其中触发输入一路作为周期，另一路作为占空比。当其中一个 CHxFP 信号被选做触发输入信号时，从模式控制器被配置成复位模式。

PWM 信号由 CH1 进入，配置 CH1FP1 为触发信号，上升沿捕获。当上升沿时 CH1 和 CH2 同时捕获，计数器 CNT 清 0，经过下降沿时，CH2 捕获，此时计数器 CNT 的值锁存到捕获寄存器 CH2CCVAL 中，到了下一个上升沿的时候，CH1 捕获，计数器 CNT 的值锁存到 CH1CCVAL 中。其中 CH2CCVAL 测量的是脉冲宽度，CH1CCVAL 测量的是周期。PWM 输入时序框图如下图 4.20 所示。

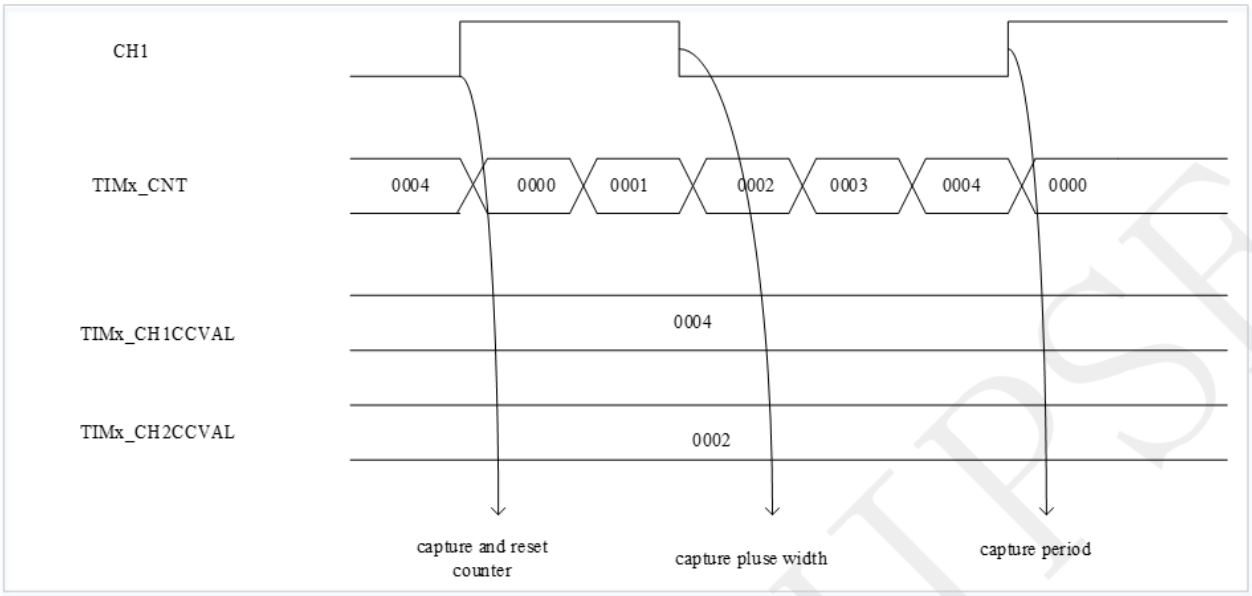

图 4.20 PWM 输入捕获时序图

4.5 输出比较

4.5.1 输出比较通道结构

输出比较是通过定时器的外部引脚对外输出波形。可以设置为不同的模式：冻结、匹配时输出有效电平、匹配时输出无效电平、电平翻转、强制为无效电平、强制为有效电平、PWM1 与 PWM2 共 8 种模式。输出阶段产生一个中间波形 OCxREF 作为基准，链的末端决定最终输出信号的极性。

在比较模式下，预装载寄存器的内容复制到影子寄存器中，然后将影子寄存器的内容与计数器进行比较。

当捕获/比较寄存器和计数器之间发现匹配时，输出比较功能：

- 相应的输出引脚的值可以配置，由输出比较模式（TIMx_CHxCFGR 寄存器中的 CHxOCMSEL 位）和输出极性（TIMx_CCCTR 寄存器中的 CHxCCP 位）共同决定。输出引脚可以保持其电平（CHxOCMSEL = 000），设置为有效（CHxOCMSEL = 001），设置为无效（CHxOCMSEL = 010）或可以匹配时翻转（CHxOCMSEL = 011）
- 在中断状态寄存器中设置一个标志（TIMx_STS 寄存器中的 CHxCCIF 位）。
- 如果相应的中断被置 1（TIMx_DIEN 寄存器中的 CHxINTEN 位），则产生中断。

如果相应的 DMA 使能位置 1（TIMx_DIEN 寄存器中的 CHxDEN 位，TIMx_CTR2 寄存器中的 CHDMARS 位用于 DMA 请求选择），则发送 DMA 请求。

4.5.2 强制输出模式

在输出模式下（TIMx_CHxCFGR 寄存器中的 CCxFS 为 00），每个输出比较信号（OCxREF 和 CHxOC）可以通过软件直接强制为有效或无效电平，与输出比较寄存器和计数器之间的任何比较无关。置输出比较信号（OCxREF / CHxOC）为有效电平，只需在相应的 TIMx_CHxCFGR 寄存器的 CHxOCMSEL 位中写入 101。因此，OCxREF 被强制为高（OCxREF 始终为高电平有效），CHxOC 与 CHxCCP 极性位的值相反。

例如：CHxCCP = 0（CHxOC 高电平有效）=> CHxOC 被强制为高电平。

通过在 TIMx_CHxCFGR 寄存器中将 CHxOCMSEL 位写入 100，可以将 OCxREF 信号强制为低电平。

无论如何，仍然执行 TIMx_CHxCCVAL 影子寄存器和计数器之间的比较，并允许设置标志。可以相应地发送中断和 DMA 请求。输出比较模式部分对此进行了描述。

4.5.3 输出比较模式

用来控制 输出一个波形，或者指示一段给定的时间已经到时。当捕获/比较寄存器和计数器之间发现匹配时，输出比较功能：

- 相应的输出引脚的值可以配置，由输出比较模式（TIMx_CHxCFGR 寄存器中的 CHxOCMSEL 位）和输出极性（TIMx_CCCTR 寄存器中的 CHxCCP 位）共同决定。输出引脚可以保持其电平（CHxOCMSEL = 000），设置为有效（CHxOCMSEL = 001），设置为无效（CHxOCMSEL = 010）或可以匹配时翻转（CHxOCMSEL = 011）
- 在中断状态寄存器中设置一个标志（TIMx_STS 寄存器中的 CHxCCIF 位）
- 如果相应的中断被置 1（TIMx_DIEN 寄存器中的 CHxINTEN 位），则产生中断

如果相应的 DMA 使能位置 1（TIMx_DIEN 寄存器中的 CHxDEN 位，TIMx_CTR2 寄存器中的 CHDMARS 位用于 DMA 请求选择），则发送 DMA 请求

编程实例：输出比较模式配置

- **选择计数器时钟（内部、外部及设置预分频比）**
- **在** TIMx_UVAL **和** TIMx_CHxCCVAL **寄存器中写入所需数值**
- **如果生成中断请求，则需将** CHxINTEN/CHxDEN **位置 1**
- **选择输出模式：**

  ①当 CNT **与** CHxCCVAL **匹配时，写入** CHxOCMSEL=011 **以翻转** CHxOC **输出引脚**

  ②写入 CHxOCVPEN=0 **以禁止预装载寄存器**

  ③写入 CHxCCP=0 **以选择高电平有效极性**

  ④写入 CHxCCEN =1 **使能输出**

- **将 TIMx_CTR1 寄存器中的 CEN 位置 1 来使能计数器**

注：可通过软件随时更新 TIMx_CCRx 寄存器以控制输出波形，但不使能预加载寄存器 CHxOCVPEN =0**，否则仅当生成下一个更新事件时，才会更新事件 UVAL 时，才会更新** TIMx_CHxCCVAL **影子寄存器。**

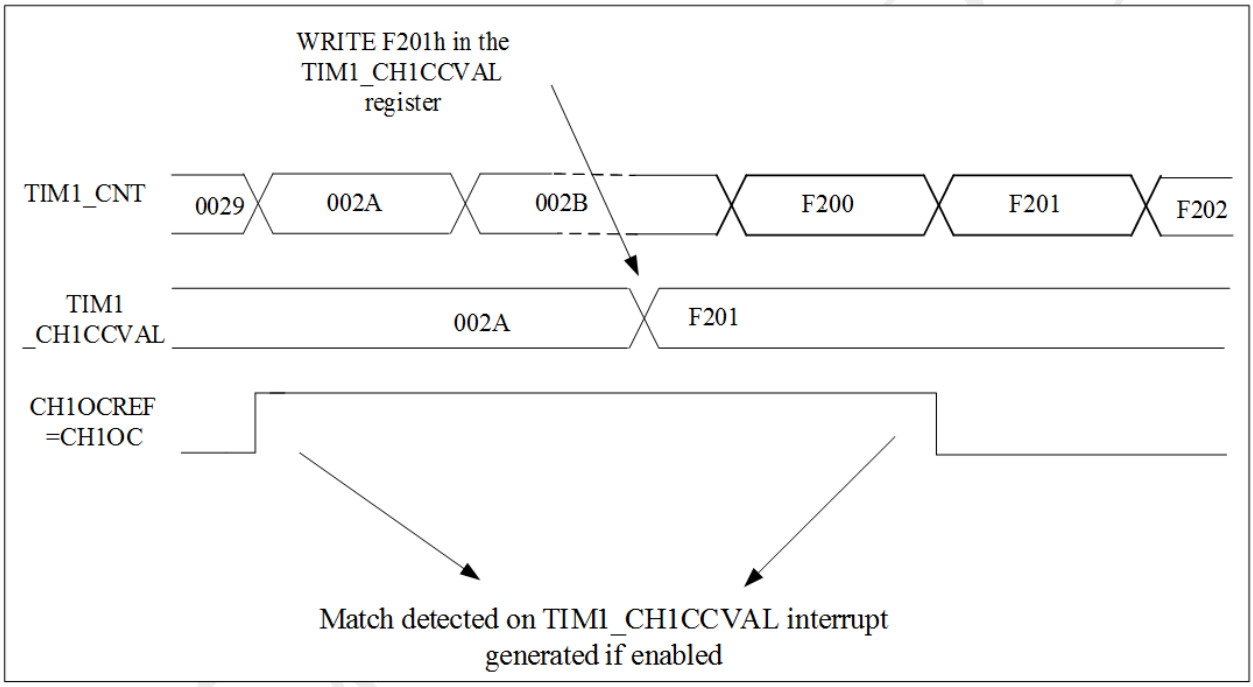

图 4.25 输出比较工作时序图

4.5.4 PWM 模式

脉冲宽度调制模式允许生成一个信号，其频率由 TIMx_UVAL 寄存器的值决定，占空比由 TIMx_CHxCCVAL 寄存器的值决定。

通过在 TIMx_CHxCFGR 寄存器的 CHxOCMSEL 位中写入 110（PWM 模式 1）或 111（PWM 模式 2），可以在每个通道上独立选择 PWM 模式（每个 CHxOC 输出一个 PWM）。必须通过设置 TIMx_CHxCFGR 寄存器中的 CHxOCVPEN 位来启用相应的预加载寄存器，并通过设置 TIMx_CTR1 寄存器中的 UVALSEN 位来设置自动重载预载寄存器（在向上计数或中央对齐模式下）。

由于仅在发生更新事件时将预加载寄存器传送到影子寄存器，因此在启动计数器之前，必须通过将 TIMx_SWEGR 寄存器中的 UEG 位置 1 来初始化所有寄存器。CHxOC 极性可通过软件编控制 TIMx_CCCTR 寄存器 CHxCCP。它可以编程为高电平有效或低电平有效。通过 TIMx_CCCTR 寄存器中的 CHxCCEN 位使能 CHxOC 输出。

4.5.4.1 PWM 边沿对齐模式

向上计数配置：当 TIM1_CTR1 寄存器中的 DIR 位为 0 时是递增计数。

- 在 PWM 模式 1 中，当 TIM1_CNT<TIMx_CHxCCVAL 时，PWM 参考信号 OCxREF 为高电平，否则为低电平
- 如果 TIMx_CHxCCVAL 中的值比自动重装载值大时，则 CHxOCREF 保持为高电平
- 如果 TIMx_CHxCCVAL 中的值为 0，则 CHxOCxREF 保持为低电平

如下图 4.26 所示，当 TIM1_UVAL=8、TIMx_CHxCCVAL 为不同取值时，边沿对齐的 PWM 输出波形

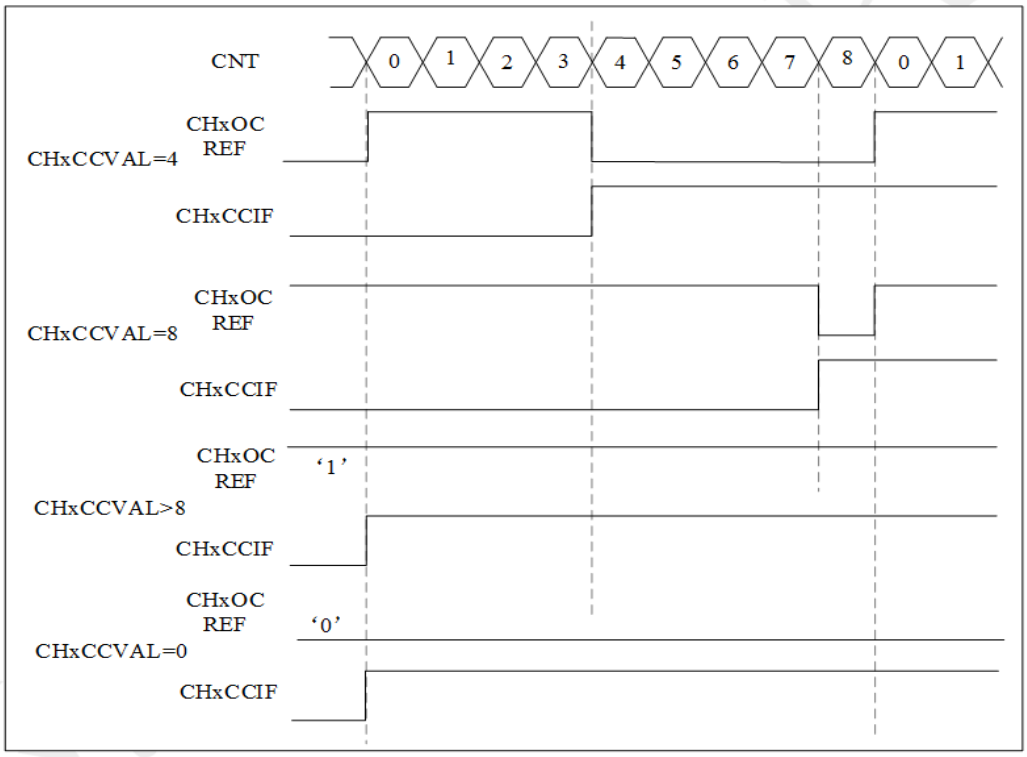

图 4.26 边沿对齐的 PWM 波形图（UVAL=8）

向下计数配置：当 TIM1_CTR1 寄存器中的 DIR 位为 1 时是递减计数。

- **在 PWM 模式 1 中，当 TIM1_CNT>** TIMx_CHxCCVAL **时，PWM 参考信号 CHxOCREF 为低电平，否则为高电平**
- **如果** TIMx_CHxCCVAL **中的值比自动重装载值大时，则 CHxOCREF 保持为高电平**
- **此模式不能产生 0%的 PWM 波形**

4.5.4.2 PWM 中央对齐模式

当 TIM1_CTR1 寄存器中的 CPS **位域不为 “00” 时 PWM 设置为中央对齐模式。按照** CPS **位域的设置不同，比较标志位可以在计数器向上、向下或中央对齐时被置 1。在此模式下，TIM1_CTR1 寄存器的方向位由硬件更新，无需软件干预。当 TIM1_UVAL=0、**CPS=01**，PWM 模式 1 输出的波形如下图 4.27 所示。**

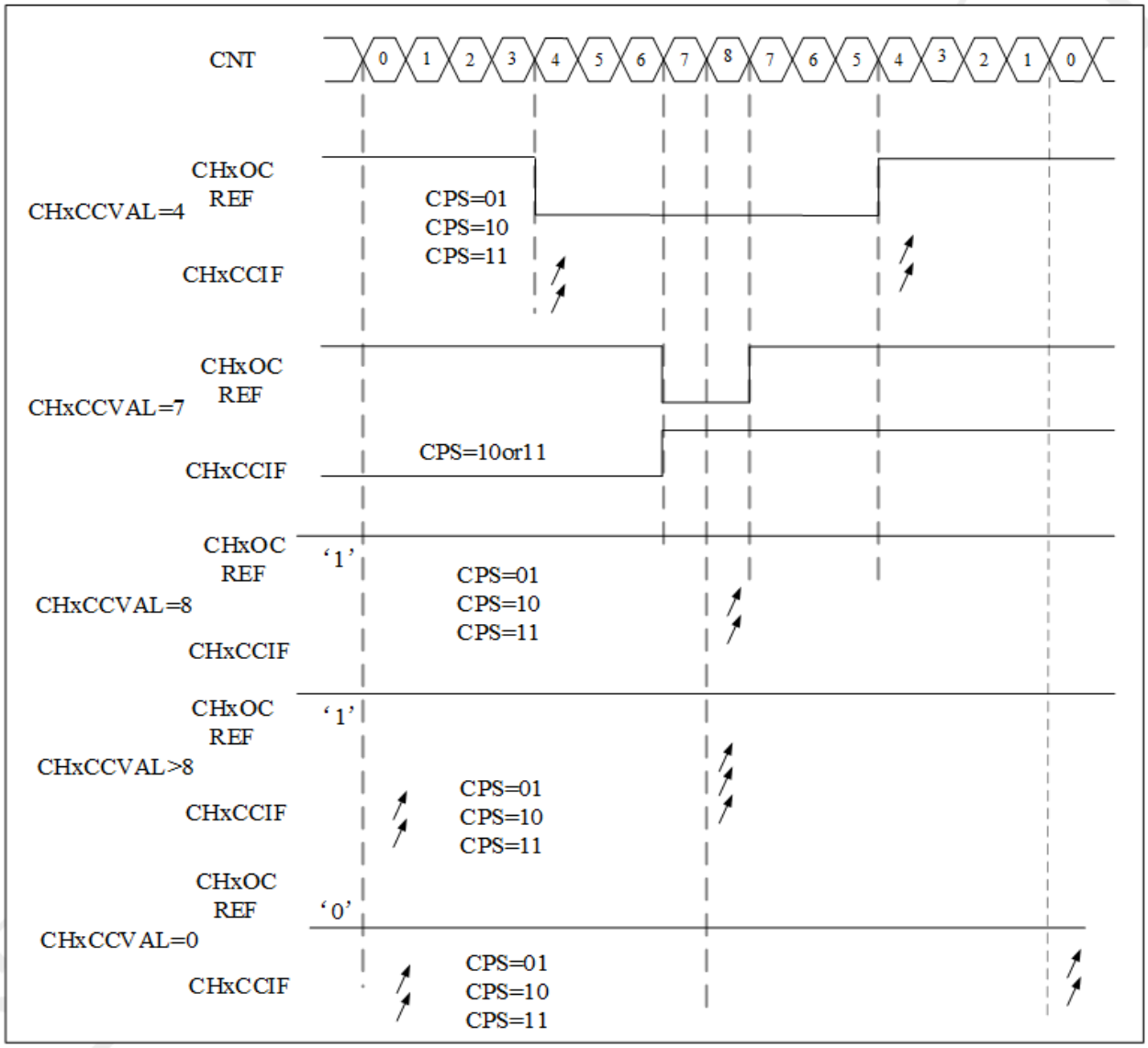

图 4.27 中央对齐的 PWM 波形图（UVAL **=8）**

在进入中央对齐模式中，计数开始的方向取决于 TIM1_CTR1 寄存器 DIR 位的当前值，软件不能同时修改 DIR 和 CPS 位。使用中央对齐模式下，在启动计数器之前需要软件产生一个更新事件（设置 UEG 位）以初始化个寄存器，在计数进行过程中禁止软件修改计数器的值。

4.5.5 互补输出与死区插入

4.5.5.1 互补输出

互补信号 CHxOC 和 CHxNOC 有效由几个控制位组合决定：TIMx_CCCTR 寄存器中的 CHxCCEN 和 CHxNCCEN 位以及 TIMx_CHOPR 和 TIMx_CTR2 寄存器中的 CHOPEN, IVOx，IVOxN，IDLEOS 和 RUNOS 位。特别是，当切换到 IDLE 状态（CHOPEN 下降到 0）时，死区时间被激活。

4.5.5.2 死区插入

通过置位 CHxCCEN 和 CHxNCCEN 位来启用死区插入，如果存在刹车电路则启用 CHOPEN 位。每个通道都有一个 10 位死区时间发生器。从参考波形 CHxOCREF，它产生 2 个输出 CHxOC 和 CHxNOC。如果 CHxOC 和 CHxNOC 处于高电平有效：

- CHxOC 输出信号与参考信号相同，但上升沿除外，该上升沿相对于参考上升沿存在延迟。
- 除上升沿之外，CHxNOC 输出信号与参考信号相反，上升沿相对于参考下降沿存在延迟。

4.6 刹车及清除参考信号

4.6.1 刹车

刹车功能：相对于汽车的手刹功能，用于紧急制动，关闭 PWM 输出，并且把通道输出锁定在安全输出状态（输出信号置于复位状态或者一个已知状态）。

当使用刹车功能时，依据相应的控制位（TIMx_BDTR 寄存器中的 MOE、OSSI 和 OSSR 位，TIMx_CR2 寄存器中的 OISx 和 OISxN 位），输出使能信号和无效电平都会被修改。任何情况下 OCx 和 OCxN 输出都不能同时置为有效电平。

4.6.2 外部事件清除 CHxOCREF 信号

当 OCREF_CLR_INPUT 上施加高电平时，可以清除给定通道的 CHxOCREF 信号（相应 TIMx_CHxCFGR 寄存器中的 CHxOCCEN 使能位设置为 1）。在下一次更新事件（UEV）发生之前，CHxOCREF 保持低电平。该功能只能用于输出比较和 PWM 模式。它在强制模式下不起作用。

编程实例：ETR 清除 CHxOCREF 信号配置

- 外部触发预分频器必须处于关闭：TIM1_SMCFG 寄存器的 ETPDIV [1:0]=00
- 外部时钟模式 2 必须禁止：TIM1_SMCFG 寄存器中的 ECMODE=0
- 外部触发极性（ETRINV）和外部触发滤波器（ETFLT [3:0]）可以根据需要设置

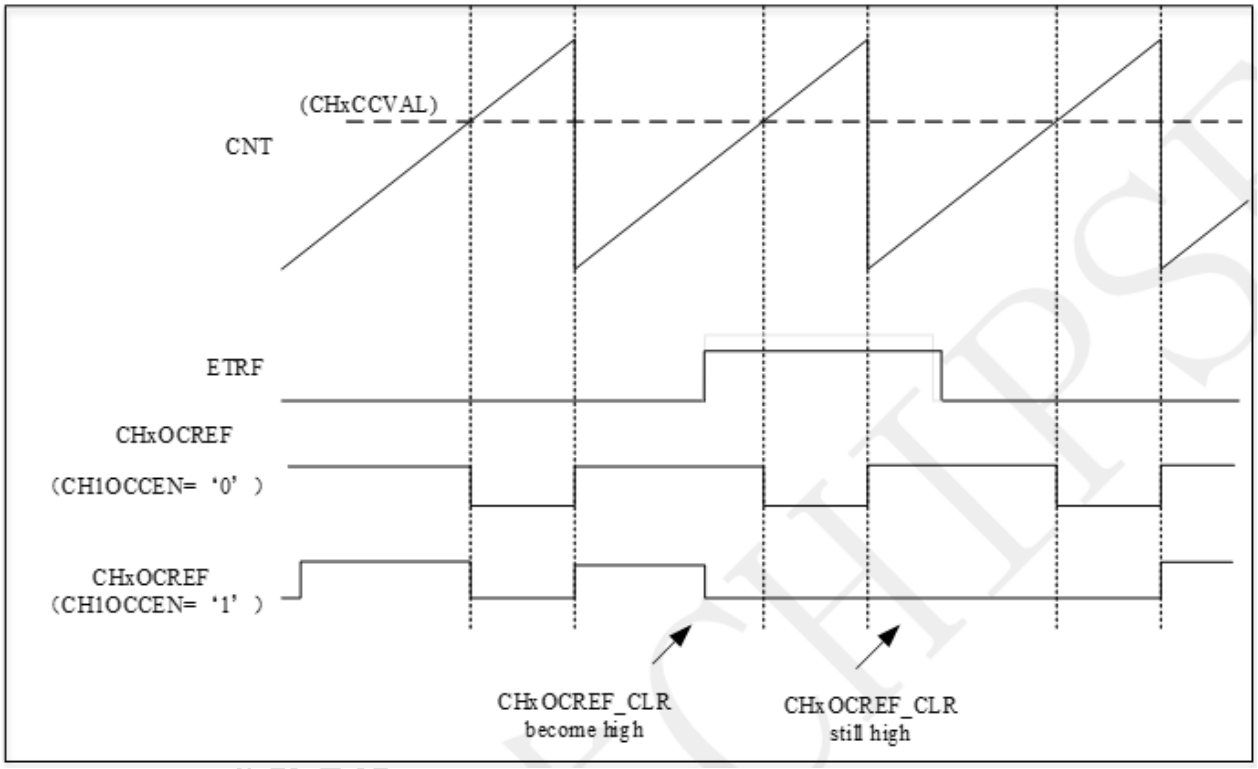

图 4.28 ETR 清除 CHxOCREF 信号

4.6.3 产生 6-step PWM 输出

产生 6-step PWM 输出，主要用于无刷电机驱动。COM 事件：换向事件，发生 COM 事件时产生控制更新，事件发生时，当 CCPC=1，允许更新 CHxCCEN、CHxNCCEN、CHxOCMSEL 位。通过 COM 事件可以同时更新 3 个互补通道的配置，解决软件编程中只能一次设置一个互补通道的问题。

COM 事件发生时（TIMx_STS 寄存器中的 CHCOMIF 位）设置一个标志，该标志可以产生中断（如果在 TIMx_DIEN 寄存器中设置了 COMINTEN 位）或 DMA 请求（如果在 TIMx_DIEN 寄存器中设置了 COMDEN 位）。

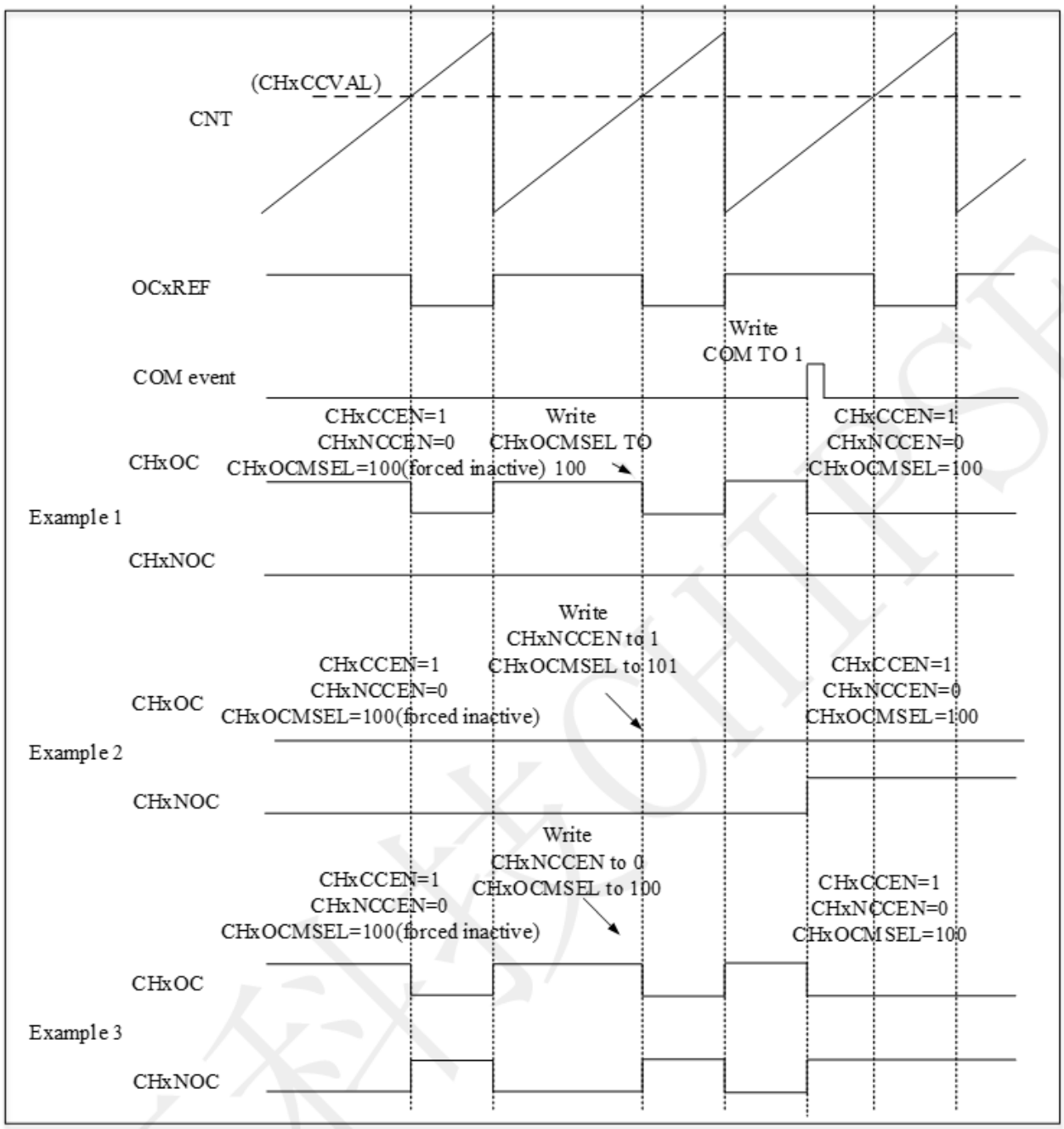

图 4.29 COM 事件对 CHxOC 和 CHxNOC 输出影响

4.7 单脉冲模式

单脉冲模式是先前模式的特定情况。它允许计数器响应激励而进行自启动，并在可编程延迟后产生具有可编程长度的脉冲。

可以通过从模式控制器控制启动计数器。生成波形可以在输出比较模式或 PWM 模式下完成。通过将 TIMx_CTR1 寄存器中的 SPEN 位置 1 选择单脉冲模式。这使得计数器在下一个更新事件 UEV 处自动停止。

编程实例：通过 CH2 触发产生一个可编程单脉冲

- **编程 TIM1_**CH12CFGR **寄存器的** CH2FS **=01，把 CH2FP2 映射到** CH2
- **编程 TIM1_**CCCTR **寄存器中的** CH2CCP **=0，使 CH2FP2 能够检测上升沿**
- **编程 TIM1_SMCFG 寄存器中的 TRIGS=110，CH2FP2 作为从模式控制器的触发 TRGI**
- **编程 TIM1_SMCFG 寄存器中的 SMCFG=110，将 CH2FP2 用作启动计数器**
- **编程 TIM1_**CH12CFGR **寄存器中的 CH1OCMSEL =111，选择 PWM 模式 2**
- **根据需要选择地使能预装载寄存器，即置 TIM1_**CH12CFGR **中的** CH1OCVPEN **=1 和 TIM1_CTR1 寄存器中的 UVALSEN**
- **向 TIM1_CH1CCVAL 寄存器中填写比较值，向寄存器中写入自动装载值**
- **设置 TIM1_EGR 寄存器的 UEG 位来产生一个更新事件，以初始化预装载寄存器**
- **等待一个 CH2 上的外部触发事件**

由 CH2 触发的单脉冲模式时序如下图 4.30 所示。

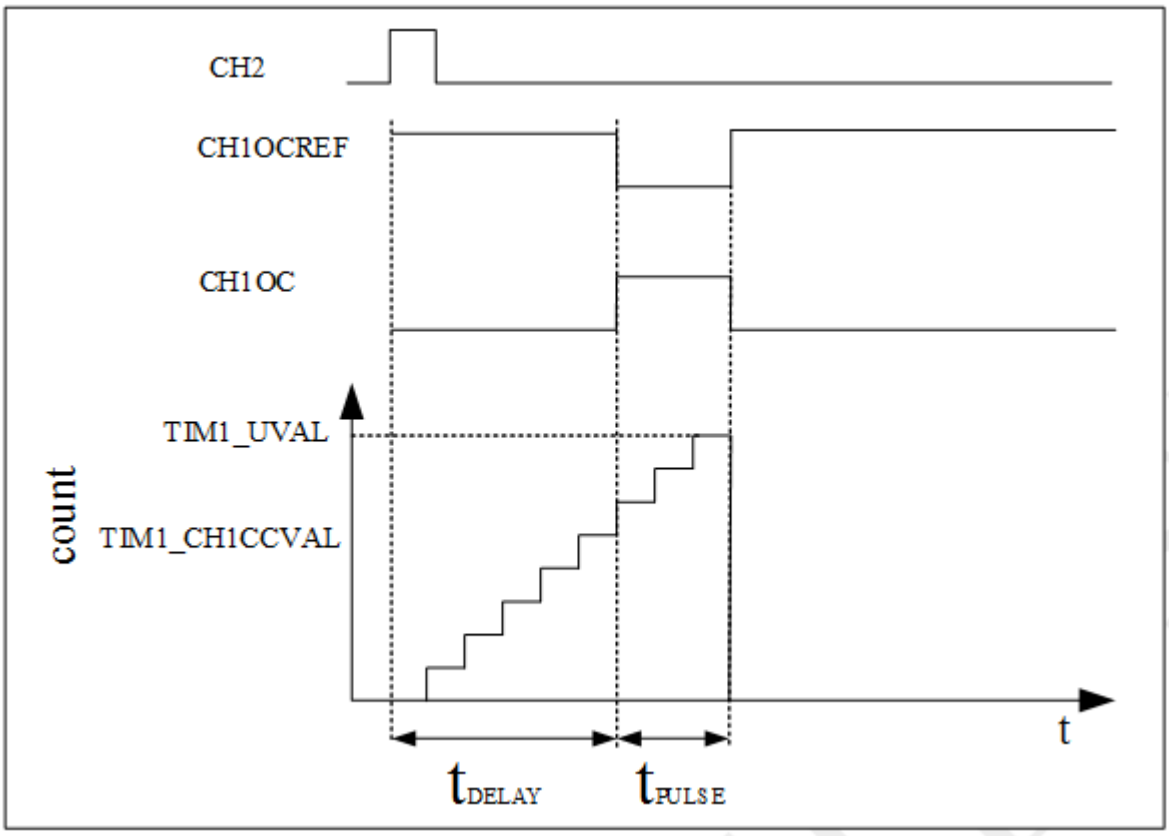

图 4.30 由 CH2 触发的单脉冲模式时序

4.8 编码器接口模式

选要选择编码器接口模式，如果计数器仅在 CH2 边沿上计数，则在 TIMx_SMCFG 寄存器中写入 SMCFG =001，如果仅在 CH1 边沿上，则 SMCFG = 010，如果在 CH1 和 CH2 边沿上，则 SMCFG = 011。通过编程 TIMx_CCCTR 寄存器中的 CH1CCP 和 CH2CCP 位选择 CH1 和 CH2 极性。 CH1NCCP 和 CH2NCCP 必须保持为低。如果需要，也可以对输入滤波进行编程。

编码器接口模式仅用作带方向选择的外部时钟。这意味着计数器仅在 0 和 TIMx_UVAL 寄存器中的自动重载值之间连续计数（0 到 UVAL 或 UVAL 下降到 0，具体取决于方向）。所以你必须在启动之前配置 TIMx_UVAL。同样，捕获，比较，预分频器，触发输出功能继续正常工作。在此模式下，计数器会根据增量编码器的速度和方向及其内容自动修改，因此始终代表编码器的位置。计数方向对应于连接的传感器的旋转方向。

在此模式下，计数器会根据增量编码器的速度和方向自动进行修改。因此其内容始终表示编码器的位置。计数方向对应所连接传感器的旋转方向。下图汇总了可能的组合。

有效边沿	反向信号电平	CH1FP1 信号		CH2FP2 信号	
		上升沿	下降沿	上升沿	下降沿
只在 CH1 通道计数	高	向下计数	向上计数	不计数	不计数
	低	向上计数	向下计数	不计数	不计数
只在 CH2 通道计数	高	不计数	不计数	向上计数	向下计数
	低	不计数	不计数	向下计数	向上计数
在 CH1、CH2 通道计数	高	向下计数	向上计数	向上计数	向下计数
	低	向上计数	向下计数	向下计数	向上计数

图 4.31 计数方向与编码器信号的关系

以计数器工作为例，说明了计数信号的生成和方向控制。同时也说明了选择双边沿时如何对输入抖动进行补偿。将传感器靠近其中一个切换点放置时可能出现这种情况。

编程实例：定时器编码器接口模式

- CH1FS = 01（TIMx_CH12CFGR 寄存器，CH1FP1 映射到 CH1）
- CH2FS = 01（TIMx_CH34CFGR 寄存器，CH2FP2 映射到 CH2）
- CH1CCP = 0，CH1NCCP =0（TIMx_CCCTR 寄存器，CH1FP1 同相，CH1FP1 = CH1）
- CH2CCP = 0，CH2NCCP =0（TIMx_CCCTR 寄存器，CH2FP2 同相，CH2FP2 = CH2）
- SMCFG = 011（TIMx_SMCFG 寄存器，两个输入在上升沿和下降沿均有效）
- CEN = 1（TIMx_CTR1 寄存器，计数器已启用）

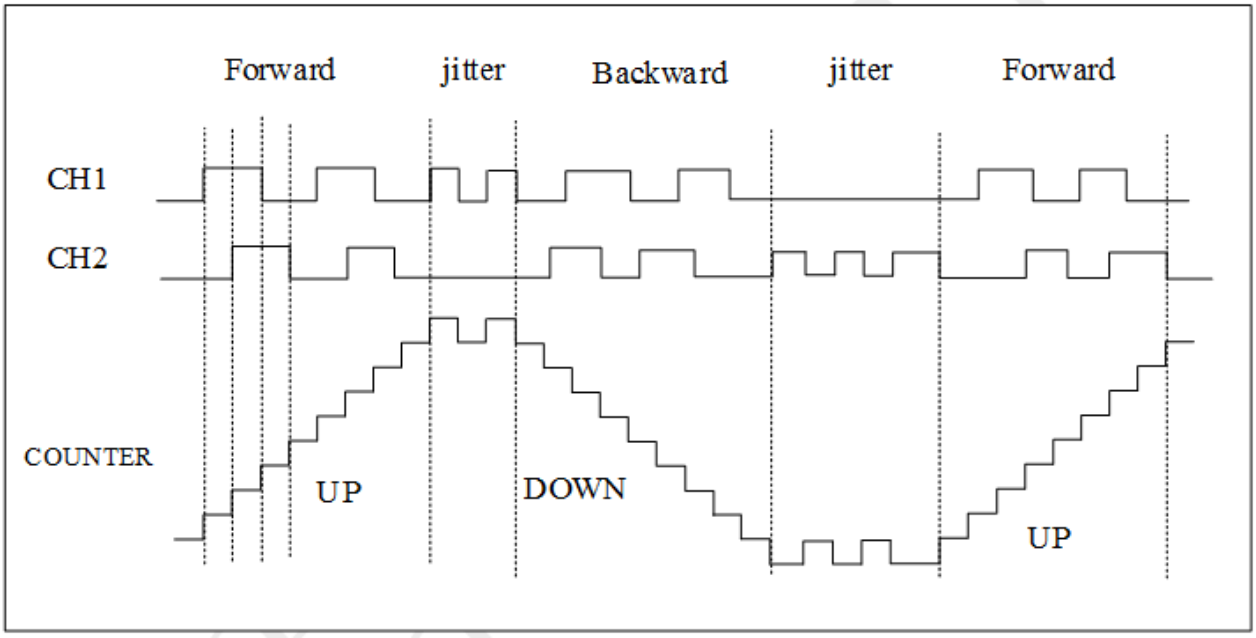

图 4.32 编码器接口模式计数工作示例

4.9 外部同步触发

TIMx 定时器在内部链接在一起，用于定时器同步或链接。当一个定时器配置为主模式时，它可以复位，启动，停止或计时在从模式下配置的另一个定时器的计数器。

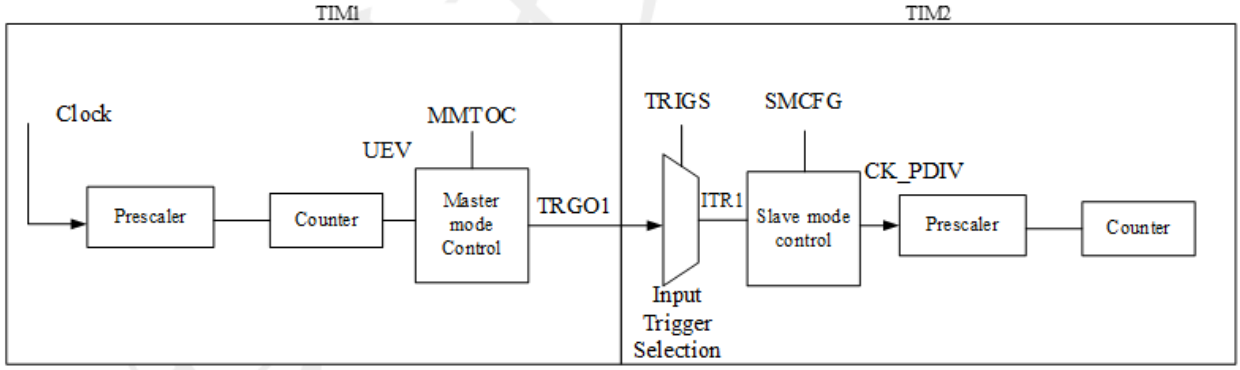

图 4.33 主从定时器示例

编程实例：使用一个计时器启用另一个计时器

- 配置定时器 1 主模式以使能从定时器（TIM1_CTR2 寄存器中的 MMTOC = 001）。
- 配置定时器 1 OC1REF 波形（TIM1_CH12CFGR 寄存器）。
- 配置定时器 2 以从定时器 1 获取输入触发（TIM2_SMCFG 寄存器中的 TRIGS = 000）。
- 在门控模式下配置定时器 2（TIM2_SMCFG 寄存器中的 SMCFG = 101）。
- 通过在 CEN 位（TIM2_CTR1 寄存器）中写入 1 来使能定时器 2。
- 通过在 CEN 位（TIM1_CTR1 寄存器）中写入 1 来启动定时器 1。

注意：计数器 2 时钟与计数器 1 不同步，此模式仅影响定时器 2 计数器使能信号。

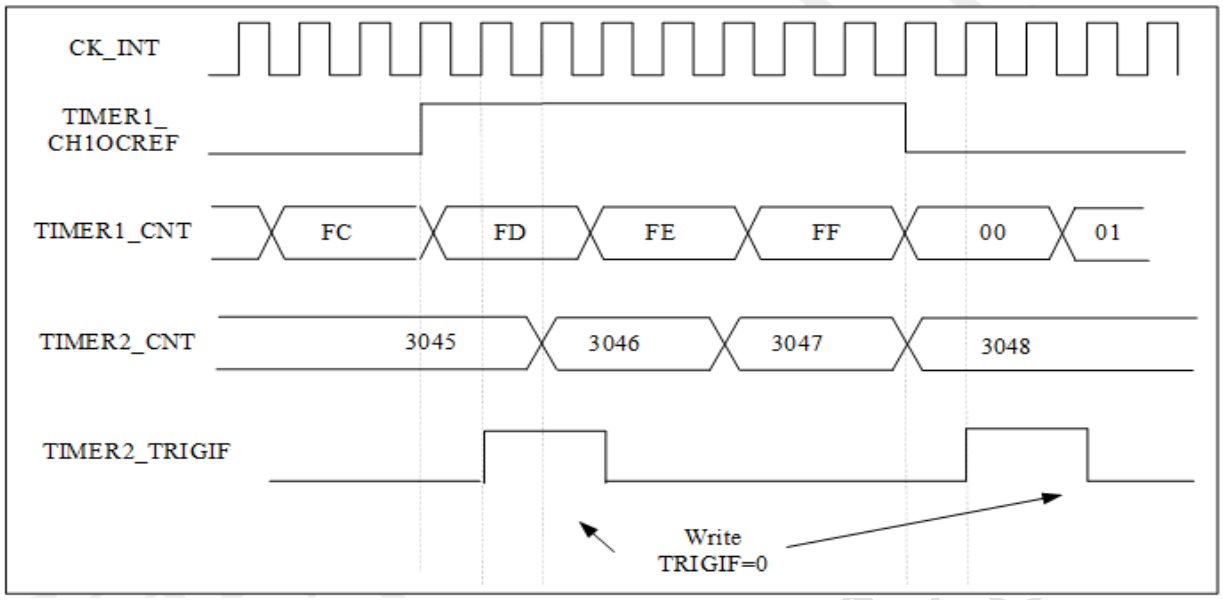

图 4.34 门控模式

在门控模式下，定时器按照选中的输入电平使能计数，计数器的启动和停止都是受控的。

编程实例：相应外部触发器同步启动 2 个定时器

- 配置定时器 1 主模式以将其启用作为触发输出（TIM1_CTR2 寄存器中的 MMTOC = 001）。
- 配置定时器 1 从机模式以从 CH1 获取输入触发（TIM1_SMCFG 寄存器中的 TRIGS = 100）。
- 将定时器 1 配置为触发模式（TIM1_SMCFG 寄存器中的 SMCFG = 110）。
- 通过写 MSM = 1（TIM1_SMCFG 寄存器）将定时器 1 配置为主/从模式。
- 配置定时器 2 以从定时器 1 获取输入触发（TIM2_SMCFG 寄存器中的 TRIGS = 000）。
- 将定时器 2 配置为触发模式（TIM2_SMCFG 寄存器中的 SMCFG = 110）。

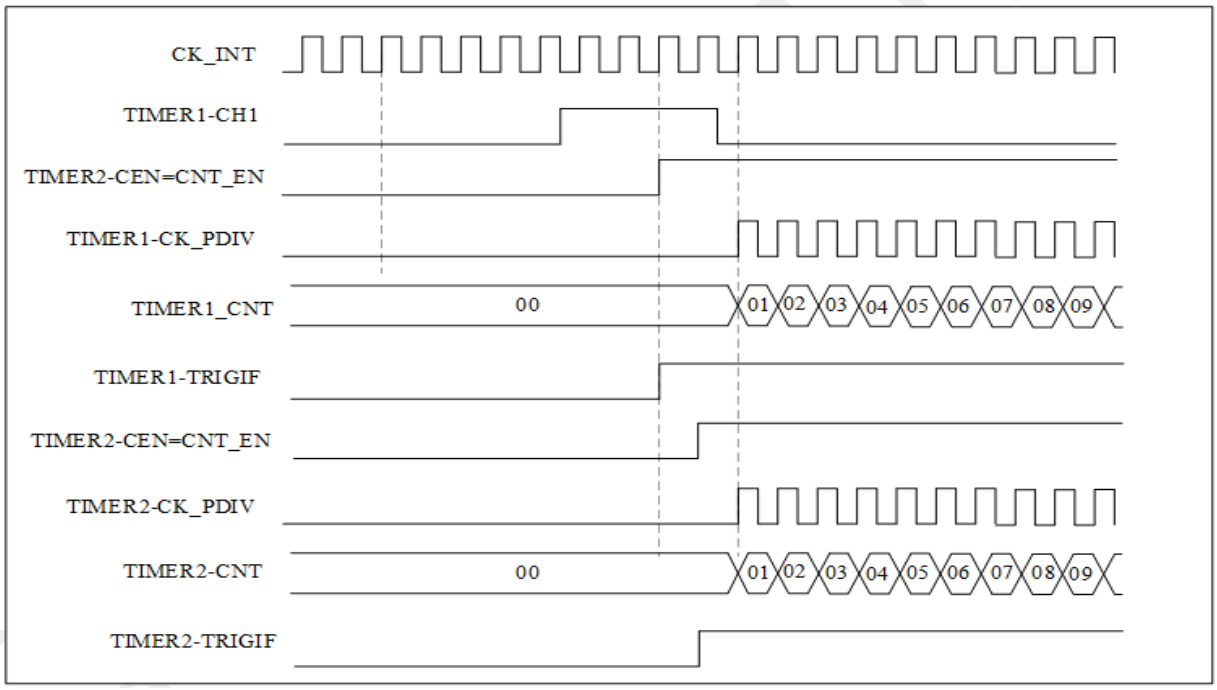

图 4.35 使用定时器 1CH1 输入触发定时器 1 和 2

5 特定场景应用示例

5.1 输出比较中断模式

5.1.1 编程步骤

①TIM3 时钟使能

这里通过 APB1ENR 的第一位设置 TIM3 时钟，程序中 TIM3 时钟->PCLK1->SystemCoreClock=48MHz

②设置 TIM3_UVAL 和 TIM3_PDIV

通过这两个寄存器，设置自动重装载值，以及分频系数

③设置 TIM3 的 PWM 为冻结模式

TIM3 的 PWM 模式默认为冻结模式

④装载输出比较寄存器的值

通过 TIM3_CH1CCVAL 寄存器装载捕获比较寄存器的值

⑤设置 TIM3_DIEN 允许更新中断

使用四个通道的更新中断，所以设置 DIEN 的 UPINTEN 位为 1，使能更新中断

⑥允许 TIM3 工作

配置完定时器，通过 TIM3_CTR1 的 CEN 位来开启定时器

⑦编写中断服务函数

通过中断函数 4 个通道的更新中断，翻转 GPIO 口。以占空比为 50%，频率分别为 585.9 Hz，1171.8 Hz，2343.75 Hz，4687.5 Hz，翻转 PA6,PA7,PB0,PB1

5.1.2 硬件设计

- 利用翻转 PA0,PA1,PA2,PA3GPIO 口观察 PWM 波形
- 定时器 TIM3

本例程中通过 TIM3 中断来控制翻转 PA6,PA7,PB0,PB1； GPIO 口观察 PWM 波形。TIM3 属于 CS32 内部资源，只需要软件设置即可。

注：软件代码见附录 1

5.2 外部同步触发产生 6-step PWM

5.2.1 编程步骤

①TIM1 时钟使能

这里通过 APB1ENR 的第一位设置 TIM1 时钟，程序中 TIM1 时钟->PCLK1->SystemCoreClock=48MHz

②设置 TIM1_UVAL 和 TIM1_ PDIV

通过这两个寄存器，设置自动重装载值，以及分频系数

③设置 TIM1 的 PWM 为 PWM1 模式

TIM1 的 PWM 模式为 PWM1 模式

④装载输出比较寄存器的值

通过 TIM1_ CH1CCVAL、TIM1_ CH2CCVAL、TIM1_ CH3CCVAL 寄存器装载捕获比较寄存器的值，但是高级定时器要输出 PWM，必须还要设置一个 CHOPEN 位（TIMx_ CHOPR 的第 15 位），以使能主输出，否则不会输出 PWM

⑤使能 SysTick100ms 中断

在 SysTick 中断使能产生 TIM1COM 同步事件

⑥编写 TIM1COM 同步事件中断服务函数

在中断服务函数中修改 6 个通道的通断，从而产生 6 步 PWM 信号

5.2.2 硬件设计

- GPIO 口

利用 GPIO 口复用 PA8、PA9、PA10 分别为通道 1、通道 2、通道 3，复用 PA7、PB0、PB1 分别作为通道 1 互补通道、通道 2 互补通道、通道 3 互补通道，从而观察相应的 6-step PWM 波形，复用 PB12 作为 TIM1 的刹车引脚，高电平有效

- 定时器 TIM1

本例程中通过 TIM1 更新 COM 同步中断事件来控制翻转 PA80，PA9，PA10，PA7，PB0，PB1GPIO 口观察 PWM 波形。TIM1 属于 CS32 内部资源，只需要软件设置即可。

注：软件代码见附录 2

5.3 TIM1 PWM 7 通道输出模式

5.3.1 编程步骤

①TIM1 时钟使能

这里通过 APB1ENR 的第一位设置 TIM1 时钟，程序中 TIM1 时钟->PCLK1->SystemCoreClock=48MHz

②设置 TIM1_UVAL 和 TIM1_PSC

通过这两个寄存器，设置自动重装载值，以及分频系数

③设置 TIM1 的 PWM 为 PWM2 模式

TIM1 的 PWM 模式为 PWM2 模式

④装载输出比较寄存器的值

通过 TIM1_ CH1CCVAL、TIM1_ CH2CCVAL、TIM1_ CH3CCVAL、TIM1_ CH4CCVAL 寄存器装载捕获比较寄存器的值，但是高级定时器要输出 PWM，必须还要设置一个 CHOPEN 位（TIMx_ CHOPR 的第 15 位），以使能主输出，否则不会输出 PWM

5.3.2 硬件设计

- GPIO 口

利用 GPIO 口复用 PA8、PA9、PA10、PA11 分别为通道 1、通道 2、通道 3、通道 4，复用 PA7、PB0、PB1 分别作为通道 1 互补通道、通道 2 互补通道、通道 3 互补通道，从而观察相应的 7 通道 PWM 波形

- 定时器 TIM1

本例程中通过 TIM1 的 PWM2 模式来控制翻转 PA80，PA9，PA10，PA7，PB0，PB1GPIO 口观察 PWM 波形。TIM1 属于 CS32 内部资源，只需要软件设置即可。

注：软件代码见附录 3

5.4 PWM 触发 ADC 转换模式

5.4.1 编程步骤

①TIM1 时钟使能

这里通过 APB1ENR 的第一位设置 TIM1 时钟，程序中 TIM1 时钟->PCLK1->SystemCoreClock=48MHz

②设置 TIM1_UVAL 和 TIM1_PDIV

通过这两个寄存器，设置自动重装载值，以及分频系数

③设置 TIM1 的 PWM 为 PWM1 模式

TIM1 的 PWM 模式为 PWM1 模式

④装载输出比较寄存器的值

通过 TIM1_ CH4CCVAL 寄存器装载捕获比较寄存器的值，但是高级定时器要输出 PWM，必须还要设置一个 CHOPEN 位（TIMx_ CHOPR 的第 15 位），以使能主输出，否则不会输出 PWM

⑤初始化 ADC

初始化 PA0 作为电压采样输入端口，设置 ADC 为持续转换，触发方式为 TIM1 CH4 的上升沿，并使能 ADC

⑥初始化串口

初始化 PA9、PA10 分别对应 USART_Tx、USART_Rx，设置波特率为 115200，并使能 USART1

5.4.2 硬件设计

- GPIO 口

利用 GPIO 口复用 PA0 作为外部电压采样输入端口

- 定时器 TIM1

本例程中通过 TIM1 的 PWM1 模式来控制通道 4 输出波形，并通过上升沿触发 ADC 采样

- ADC

利用 ADC 的通道 0 采样外部电压值

- USART

利用串口打印采样的电压值

注：软件代码见附录 4

附录 1 TIM_OCToggle

```
/**
  * @file    TIM/TIM_OCToggle/main.c
  * @brief   Main program body
  * @author  ChipSea MCU Group
  * @version V1.0.0
  * @date    2018.11.01
  * @copyright CHIPSEA TECHNOLOGIES (SHENZHEN) CORP.
  * @note
  * <h2><center>&copy; COPYRIGHT 2018 ChipSea</center></h2>
  *
  */


#include "main.h"


/** @addtogroup CS32F0xx_HAL_Examples
  * @{
  */

/** @addtogroup TIM_OCToggle
  * @{
  */

uint16_t ch1_pulse = 40961;
uint16_t ch2_pulse = 20480;
uint16_t ch3_pulse = 10240;
uint16_t ch4_pulse = 5120;



/**
  * @fn  void time_config(void)
  * @brief  Configure the TIM3 Pins.
  * @param  None
  * @return None
  */
void time_config(void)
{
      timer_config_t   timer_configStruct;
      timer_compare_t  timer_compareStruct;
      gpio_config_t    gpio_configStruct;
      nvic_config_t    nvic_configStruct;
      uint16_t prescaler_value = 0;

      rcu_apb1_periph_clock_enable_ctrl(RCU_APB1_PERI_TIM3, ENABLE);
       rcu_ahb_periph_clock_enable_ctrl(RCU_AHB_PERI_PORTA |
       RCU_AHB_PERI_PORTB, ENABLE); //GPIOA,GPIOB clock enable

      //GPIOA Configuration: TIM3 CH1 (PA6) and TIM3 CH2 (PA7)
      gpio_configStruct.gpio_pin = GPIO_PIN_6 | GPIO_PIN_7;
      gpio_configStruct.gpio_mode = GPIO_MODE_MF;
      gpio_configStruct.gpio_speed = GPIO_SPEED_HIGH;
      gpio_configStruct.gpio_out_type = GPIO_OUTPUT_PP;
      gpio_configStruct.gpio_pull = GPIO_PULL_UP ;
      gpio_init(GPIOA, &gpio_configStruct);
```

```
//GPIOB Configuration: TIM3 CH3 (PB0) and TIM3 CH4 (PB1)
gpio_configStruct.gpio_pin = GPIO_PIN_0 | GPIO_PIN_1;
gpio_init(GPIOB, &gpio_configStruct);

//Connect TIM Channels to MF1
gpio_mf_config(GPIOA, GPIO_PIN_NUM6, GPIO_MF_SEL1);
gpio_mf_config(GPIOA, GPIO_PIN_NUM7, GPIO_MF_SEL1);
gpio_mf_config(GPIOB, GPIO_PIN_NUM0, GPIO_MF_SEL1);
gpio_mf_config(GPIOB, GPIO_PIN_NUM1, GPIO_MF_SEL1);


/* TIM3 Configuration Output Compare Toggle Mode:
In this example TIM3 input clock (TIM3CLK) is set to APB1 clock (PCLK1).
TIM3CLK = PCLK1 = 48 MHz

CCx update rate = TIM3 counter clock / chx_pulse;
generates a periodic signal (frequency= CCx update rate/2).*/

//Time base configuration
timer_configStruct.time_period = 65535;
timer_configStruct.time_divide = prescaler_value;
timer_configStruct.clock_divide = 0;
timer_configStruct.count_mode = TIM_COUNT_PATTERN_UP;
tim_timer_config(TIM3, &timer_configStruct);

timer_compareStruct.time_mode = TIM_CHxOCMSEL_TOGGLE;
timer_compareStruct.output_polarity = TIM_CHxCCP_POLARITY_LOW;
timer_compareStruct.output_state = TIM_CHx_OUTPUT_ENABLE;

//configuration Channel1
timer_compareStruct.timer_pulse = ch1_pulse;
tim_ch1oc_init(TIM3, &timer_compareStruct);
tim_ch1oc_preload_set(TIM3, TIM_CHxOC_PRELOAD_DISABLE);

//configuration Channel2
timer_compareStruct.timer_pulse = ch2_pulse;
tim_ch2oc_init(TIM3, &timer_compareStruct);
tim_ch2oc_preload_set(TIM3, TIM_CHxOC_PRELOAD_DISABLE);

//configuration Channel3
timer_compareStruct.timer_pulse = ch3_pulse;
tim_ch3oc_init(TIM3, &timer_compareStruct);
tim_ch3oc_preload_set(TIM3, TIM_CHxOC_PRELOAD_DISABLE);

//configuration Channel4
timer_compareStruct.timer_pulse = ch4_pulse;
tim_ch4oc_init(TIM3, &timer_compareStruct);
tim_ch4oc_preload_set(TIM3, TIM_CHxOC_PRELOAD_DISABLE);

//Enable the TIM3  Interrupt
nvic_configStruct.nvic_IRQ_channel = IRQn_TIM3;
nvic_configStruct.nvic_channel_priority = 0;
nvic_configStruct.nvic_enable_flag = ENABLE;
nvic_init(&nvic_configStruct);

//TIM INT enable
tim_interrupt_config(TIM3,TIM_INTR_CH1|TIM_INTR_CH2|TIM_INTR_CH3|
TIM_INTR_CH4, ENABLE);
tim_enable_ctrl(TIM3, ENABLE); //TIM enable counter
```

```
}


/**
  * @fn int main(void)
  * @brief  Main program.   MCU clock setting is  configured  through
SystemInit()
  *          in startup file (startup_cs32f0xx.s) before to enter to
application main.
  * @param  None
  * @return None
  */
int main(void)
{
      time_config();
      while (1)
      {

      }
}


/**
  * @}
  */

/**
  * @}
  */



/**
  * @file    TIM/TIM_OCToggle/cs32f0xx_int.c
  * @brief  Main Interrupt Service Routines. It provides template for
  *            all exceptions handler and  peripherals interrupt service
routine.
  * @author  ChipSea MCU Group
  * @version V1.0.0
  * @date   2018.11.01
  * @copyright CHIPSEA TECHNOLOGIES (SHENZHEN) CORP.
  * @note
  * <h2><center>&copy; COPYRIGHT 2018 ChipSea</center></h2>
  *
  */


#include "cs32f0xx_int.h"


/** @addtogroup CS32F0xx_HAL_Examples
  * @{
  */

/** @addtogroup TIM_OCToggle
  * @{
  */

uint16_t capture = 0;
extern uint16_t ch1_pulse;
```

```
extern uint16_t ch2_pulse;
extern uint16_t ch3_pulse;
extern uint16_t ch4_pulse;



/**
  * @fn void NMI_Handler(void)
  * @brief  This function handles NMI exception.
  * @param  None
  * @return None
  */
void NMI_Handler(void)
{
}

/**
  * @fn void HardFault_Handler(void)
  * @brief  This function handles Hard Fault exception.
  * @param  None
  * @return None
  */
void HardFault_Handler(void)
{
    /* Go to infinite loop when Hard Fault exception occurs */
    while (1)
    {
    }
}

/**
  * @fn void SVC_Handler(void)
  * @brief  This function handles SVCall exception.
  * @param  None
  * @return None
  */
void SVC_Handler(void)
{
}

/**
  * @fn void PendSV_Handler(void)
  * @brief  This function handles PendSVC exception.
  * @param  None
  * @return None
  */
void PendSV_Handler(void)
{
}

/**
  * @fn void SysTick_Handler(void)
  * @brief  This function handles SysTick Handler.
  * @param  None
  * @return None
  */
void SysTick_Handler(void)
{
}
```

```
/**
  * @fn void TIM3_IRQHandler(void)
  * @brief  This function handles TIM3 global interrupt request.
  * @param  None
  * @return None
  */
void TIM3_IRQHandler(void)
{
    /* TIM3_CH1 toggling with frequency = 585.9 Hz */
    if (tim_interrupt_status_get(TIM3, TIM_INTR_CH1) != RESET)
    {
        tim_interrupt_status_clear(TIM3, TIM_INTR_CH1 );
        capture = tim_ch1cc_value_get(TIM3);
        tim_ch1cc_value_set(TIM3, capture + ch1_pulse );
    }

    /* TIM3_CH2 toggling with frequency = 1171.8 Hz */
    if (tim_interrupt_status_get(TIM3, TIM_INTR_CH2) != RESET)
    {
        tim_interrupt_status_clear(TIM3, TIM_INTR_CH2);
        capture = tim_ch2cc_value_get(TIM3);
        tim_ch2cc_value_set(TIM3, capture + ch2_pulse);
    }

    /* TIM3_CH3 toggling with frequency = 2343.75 Hz */
    if (tim_interrupt_status_get(TIM3, TIM_INTR_CH3) != RESET)
    {
        tim_interrupt_status_clear(TIM3, TIM_INTR_CH3);
        capture = tim_ch3cc_value_get(TIM3);
        tim_ch3cc_value_set(TIM3, capture + ch3_pulse);
    }

    /* TIM3_CH4 toggling with frequency = 4687.5 Hz */
    if (tim_interrupt_status_get(TIM3, TIM_INTR_CH4) != RESET)
    {
        tim_interrupt_status_clear(TIM3, TIM_INTR_CH4);
        capture = tim_ch4cc_value_get(TIM3);
        tim_ch4cc_value_set(TIM3, capture + ch4_pulse);
    }
}


/**
  * @}
  */

/**
  * @}
  */
```

附录 2 TIM_6Steps

```
/**
  * @file    TIM/TIM_6Steps/main.c
  * @brief   Main program body
  * @author  ChipSea MCU Group
  * @version V1.0.0
  * @date    2018.11.01
  * @copyright CHIPSEA TECHNOLOGIES (SHENZHEN) CORP.
  * @note
  * <h2><center>&copy; COPYRIGHT 2018 ChipSea</center></h2>
  *
  */


#include "main.h"


/** @addtogroup CS32F0xx_HAL_Examples
  * @{
  */

/** @addtogroup TIM_6Steps
  * @{
  */

extern uint32_t SystemCoreClock;
void time_config(void);
void sys_tick_config(void);


/**
  * @fn int main(void)
  * @brief  Main program.   MCU clock setting is  configured  through
SystemInit()
  *          in startup file (startup_cs32f0xx.s) before to enter to
application main.
  * @param  None
  * @return None
  */
int main(void)
{
    sys_tick_config();
    time_config();

    while (1)
    {
    }
}


/* TIM1 Channels states:
------------------------------------------------------
                    Step1 | Step2 | Step3 | Step4 | Step5 | Step6 |
------------------------------------------------------
|Channel1    1    |   0   |   0   |   0   |   0   |   1   |
------------------------------------------------------
|Channel1N   0    |   0   |   1   |   1   |   0   |   0   |
```

```
-----------------------------------------------------
|Channel2    0    |   0   |   0   |   1   |   1   |   0   |
-----------------------------------------------------
|Channel2N   1    |   1   |   0   |   0   |   0   |   0   |
-----------------------------------------------------
|Channel3    0    |   1   |   1   |   0   |   0   |   0   |
-----------------------------------------------------
|Channel3N   0    |   0   |   0   |   0   |   1   |   1   |
-----------------------------------------------------*/

/**
  * @fn  void time_config(void)
  * @brief  configure the TIMER peripheral.
  * @param  None
  * @return None
  */
void time_config(void)
{
      timer_config_t  timer_configStruct;
      timer_compare_t  timer_compareStruct;
      timer_protect_t time_protectStruct;
      gpio_config_t gpio_configStruct;
      nvic_config_t nvic_configStruct;

      //Enable GPIOA, GPIOB clocks
      rcu_ahb_periph_clock_enable_ctrl(RCU_AHB_PERI_PORTA |
      RCU_AHB_PERI_PORTB, ENABLE);
      rcu_apb2_periph_clock_enable_ctrl(RCU_APB2_PERI_TIM1, ENABLE); //TIM1
clock enable

      //GPIOA Configuration: Channel 1, 2, 1N and 3 as  multi-function push-
pull
      gpio_configStruct.gpio_pin = GPIO_PIN_7 | GPIO_PIN_8 | GPIO_PIN_9 |
      GPIO_PIN_10;
      gpio_configStruct.gpio_mode = GPIO_MODE_MF;
      gpio_configStruct.gpio_speed = GPIO_SPEED_HIGH;
      gpio_configStruct.gpio_out_type = GPIO_OUTPUT_PP;
      gpio_configStruct.gpio_pull = GPIO_PULL_NO_PULL;
      gpio_init(GPIOA, &gpio_configStruct);

      //GPIOB Configuration: Channel 2N and 3N as  multi-function push-pull.
      gpio_configStruct.gpio_pin = GPIO_PIN_0 | GPIO_PIN_1 | GPIO_PIN_12;
      gpio_init(GPIOB, &gpio_configStruct);

      //Connect TIM pins to MF2
      gpio_mf_config(GPIOA, GPIO_PIN_NUM7, GPIO_MF_SEL2);  //TIM1_CH1N
      gpio_mf_config(GPIOA, GPIO_PIN_NUM8, GPIO_MF_SEL2);  //TIM1_CH1
      gpio_mf_config(GPIOA, GPIO_PIN_NUM9, GPIO_MF_SEL2);  //TIM1_CH2
      gpio_mf_config(GPIOA, GPIO_PIN_NUM10, GPIO_MF_SEL2); //TIM1_CH3
      gpio_mf_config(GPIOB, GPIO_PIN_NUM0, GPIO_MF_SEL2);  //TIM1_CH2N
      gpio_mf_config(GPIOB, GPIO_PIN_NUM1, GPIO_MF_SEL2);  //TIM1_CH3N
      gpio_mf_config(GPIOB, GPIO_PIN_NUM12, GPIO_MF_SEL2); //TIM1_BKIN

      // Time Base configuration
      timer_configStruct.time_period = 4799;   // (PCLK=HCLK=48MHZ)
      timer_configStruct.time_divide = 0;
      timer_configStruct.count_mode = TIM_COUNT_PATTERN_UP;
      timer_configStruct.clock_divide = 0;
      timer_configStruct.repeate_count = 0;
      tim_timer_config(TIM1, &timer_configStruct);
```

```
    //Channel 1, 2,3 and 4 Configuration in PWM mode
    timer_compareStruct.time_mode = TIM_CHxOCMSEL_TIMING;
    timer_compareStruct.output_state = TIM_CHx_OUTPUT_ENABLE;
    timer_compareStruct.output_state_n = TIM_CHxNCCEN_ENABLE;
    timer_compareStruct.output_polarity = TIM_CHxCCP_POLARITY_HIGH;
    timer_compareStruct.output_polarity_n = TIM_CHxNCCP_POLARITY_HIGH;
    timer_compareStruct.idle_state = TIM_IVOx_SET;
    timer_compareStruct.idle_state_n = TIM_IVOxN_SET;

    timer_compareStruct.timer_pulse = 2399;
    tim_ch1oc_init(TIM1, &timer_compareStruct);

    timer_compareStruct.timer_pulse = 1199;
    tim_ch2oc_init(TIM1, &timer_compareStruct);

    timer_compareStruct.timer_pulse = 599;
    tim_ch3oc_init(TIM1, &timer_compareStruct);

    //Automatic Output enable, Break, dead time and lock configuration
    time_protectStruct.run_offstate = TIM_RUNOS_ENABLE;
    time_protectStruct.idle_offstate = TIM_IDLEOS_ENABLE;
    time_protectStruct.lock_level = TIM_LOCK_LEVEL_OFF;
    time_protectStruct.dead_time = 1;
    time_protectStruct.break_flag = TIM_BREAK_EANBLE;
    time_protectStruct.break_polarity = TIM_BREAK_POLARITY_HIGH;
    time_protectStruct.auto_output = TIM_CHOPAEN_ENABLE;
    tim_chopr_register_config(TIM1, &time_protectStruct);

    tim_ch_preload_shadow_enable_ctrl(TIM1, ENABLE);
    tim_interrupt_config(TIM1, TIM_INTR_COM, ENABLE);

    tim_enable_ctrl(TIM1, ENABLE);     // TIM1 counter enable
    tim_pwm_output_set(TIM1, ENABLE); // Main Output Enable

    //Enable the TIM1 Trigger and commutation interrupt
    nvic_configStruct.nvic_IRQ_channel = IRQn_TIM1_BRK_UP_TRG_COM;
    nvic_configStruct.nvic_channel_priority = 0;
    nvic_configStruct.nvic_enable_flag = ENABLE;
    nvic_init(&nvic_configStruct);
}

/**
  * @fn  void sys_tick_config(void)
  * @brief  Configures the SysTick.
  * @param  None
  * @return None
  */
void sys_tick_config(void)
{
    /* Setup SysTick Timer for 100 msec interrupts  */
    SysTick_Config(SystemCoreClock / 10);

    NVIC_SetPriority(SysTick_IRQn, 0x0);
}

/**
  * @}
  */
```

```
/**
  * @}
  */


/**
  * @file    TIM/TIM_6Steps/cs32f0xx_int.c
  * @brief  Main Interrupt Service Routines. It provides template for
  *            all exceptions handler and  peripherals interrupt service
routine.
  * @author  ChipSea MCU Group
  * @version V1.0.0
  * @date    2018.11.01
  * @copyright CHIPSEA TECHNOLOGIES (SHENZHEN) CORP.
  * @note
  * <h2><center>&copy; COPYRIGHT 2018 ChipSea</center></h2>
  *
  */


#include "cs32f0xx_int.h"


/** @addtogroup CS32F0xx_HAL_Examples
  * @{
  */

/** @addtogroup TIM_6Steps
  * @{
  */



__IO uint32_t step = 1;



/**
  * @fn void NMI_Handler(void)
  * @brief  This function handles NMI exception.
  * @param  None
  * @return None
  */
void NMI_Handler(void)
{
}

/**
  * @fn void HardFault_Handler(void)
  * @brief  This function handles Hard Fault exception.
  * @param  None
  * @return None
  */
void HardFault_Handler(void)
{
    /* Go to infinite loop when Hard Fault exception occurs */
    while (1)
    {
    }
}
```

```
/**
  * @fn void SVC_Handler(void)
  * @brief  This function handles SVCall exception.
  * @param  None
  * @return None
  */
void SVC_Handler(void)
{
}

/**
  * @fn void PendSV_Handler(void)
  * @brief  This function handles PendSVC exception.
  * @param  None
  * @return None
  */
void PendSV_Handler(void)
{
}

/**
  * @fn void SysTick_Handler(void)
  * @brief  This function handles SysTick Handler.
  * @param  None
  * @return None
  */
void SysTick_Handler(void)
{
    /* Generate TIM1 COM event */
    tim_event_source_set(TIM1, TIM_EVENT_SOURCE_COM);
}

/**
  * @fn void TIM1_BRK_UP_TRG_COM_IRQHandler(void)
  * @brief  It handles TIM1 Break, Update, Trigger and Commutation interrupt
request.
  * @param  None
  * @return None
  */
void TIM1_BRK_UP_TRG_COM_IRQHandler(void)
{
    /* Clear TIM1 COM  bit */
    tim_interrupt_status_clear(TIM1, TIM_INTR_COM);

    if (step == 1)
    {
        /*  Channel3 configuration */
        tim_chcc_enable_ctrl(TIM1, TIM_CHANNEL_NUM_3, TIM_CHxCC_DISABLE);
        tim_chncc_enable_ctrl(TIM1, TIM_CHANNEL_NUM_3, TIM_CHxNCC_DISABLE);

        /*  Channel1 configuration */
        tim_choc_mode_select(TIM1, TIM_CHANNEL_NUM_1, TIM_CHxOCMSEL_PWM1);
        tim_chcc_enable_ctrl(TIM1, TIM_CHANNEL_NUM_1, TIM_CHxCC_ENABLE);
        tim_chncc_enable_ctrl(TIM1, TIM_CHANNEL_NUM_1, TIM_CHxNCC_DISABLE);

        /*  Channel2 configuration */
        tim_choc_mode_select(TIM1, TIM_CHANNEL_NUM_2, TIM_CHxOCMSEL_PWM1 );
        tim_chcc_enable_ctrl(TIM1, TIM_CHANNEL_NUM_2, TIM_CHxCC_DISABLE);
        tim_chncc_enable_ctrl(TIM1, TIM_CHANNEL_NUM_2, TIM_CHxNCC_ENABLE);
```

```
        step++;
    }
    else if (step == 2)
    {
        /*  Channel2 configuration */
        tim_choc_mode_select(TIM1, TIM_CHANNEL_NUM_2, TIM_CHxOCMSEL_PWM1);
        tim_chcc_enable_ctrl(TIM1, TIM_CHANNEL_NUM_2, TIM_CHxCC_DISABLE);
        tim_chncc_enable_ctrl(TIM1, TIM_CHANNEL_NUM_2, TIM_CHxNCC_ENABLE);

        /*  Channel3 configuration */
        tim_choc_mode_select(TIM1, TIM_CHANNEL_NUM_3, TIM_CHxOCMSEL_PWM1);
        tim_chcc_enable_ctrl(TIM1, TIM_CHANNEL_NUM_3, TIM_CHxCC_ENABLE);
        tim_chncc_enable_ctrl(TIM1, TIM_CHANNEL_NUM_3, TIM_CHxNCC_DISABLE);

        /*  Channel1 configuration */
        tim_chcc_enable_ctrl(TIM1, TIM_CHANNEL_NUM_1, TIM_CHxCC_DISABLE);
        tim_chncc_enable_ctrl(TIM1, TIM_CHANNEL_NUM_1, TIM_CHxNCC_DISABLE);
        step++;
    }
    else if (step == 3)
    {
        /*  Channel3 configuration */
        tim_choc_mode_select(TIM1, TIM_CHANNEL_NUM_3, TIM_CHxOCMSEL_PWM1);
        tim_chcc_enable_ctrl(TIM1, TIM_CHANNEL_NUM_3, TIM_CHxCC_ENABLE);
        tim_chncc_enable_ctrl(TIM1, TIM_CHANNEL_NUM_3, TIM_CHxNCC_DISABLE);

        /*  Channel2 configuration */
        tim_chcc_enable_ctrl(TIM1, TIM_CHANNEL_NUM_2, TIM_CHxCC_DISABLE);
        tim_chncc_enable_ctrl(TIM1, TIM_CHANNEL_NUM_2, TIM_CHxNCC_DISABLE);

        /*  Channel1 configuration */
        tim_choc_mode_select(TIM1, TIM_CHANNEL_NUM_1, TIM_CHxOCMSEL_PWM1);
        tim_chcc_enable_ctrl(TIM1, TIM_CHANNEL_NUM_1, TIM_CHxCC_DISABLE);
        tim_chncc_enable_ctrl(TIM1, TIM_CHANNEL_NUM_1, TIM_CHxNCC_ENABLE);
        step++;
    }
    else if (step == 4)
    {
        /*  Channel3 configuration */
        tim_chcc_enable_ctrl(TIM1, TIM_CHANNEL_NUM_3, TIM_CHxCC_DISABLE);
        tim_chncc_enable_ctrl(TIM1, TIM_CHANNEL_NUM_3, TIM_CHxNCC_DISABLE);

        /*  Channel1 configuration */
        tim_choc_mode_select(TIM1, TIM_CHANNEL_NUM_1, TIM_CHxOCMSEL_PWM1);
        tim_chcc_enable_ctrl(TIM1, TIM_CHANNEL_NUM_1, TIM_CHxCC_DISABLE);
        tim_chncc_enable_ctrl(TIM1, TIM_CHANNEL_NUM_1, TIM_CHxNCC_ENABLE);

        /*  Channel2 configuration */
        tim_choc_mode_select(TIM1, TIM_CHANNEL_NUM_2, TIM_CHxOCMSEL_PWM1);
        tim_chcc_enable_ctrl(TIM1, TIM_CHANNEL_NUM_2, TIM_CHxCC_ENABLE);
        tim_chncc_enable_ctrl(TIM1, TIM_CHANNEL_NUM_2, TIM_CHxNCC_DISABLE);
        step++;
    }
    else if (step == 5)
    {
        /*  Channel3 configuration */
        tim_choc_mode_select(TIM1, TIM_CHANNEL_NUM_3, TIM_CHxOCMSEL_PWM1);
        tim_chcc_enable_ctrl(TIM1, TIM_CHANNEL_NUM_3, TIM_CHxCC_DISABLE);
        tim_chncc_enable_ctrl(TIM1, TIM_CHANNEL_NUM_3, TIM_CHxNCC_ENABLE);
```

```
        /*  Channel1 configuration */
        tim_chcc_enable_ctrl(TIM1, TIM_CHANNEL_NUM_1, TIM_CHxCC_DISABLE);
        tim_chncc_enable_ctrl(TIM1, TIM_CHANNEL_NUM_1, TIM_CHxNCC_DISABLE);

        /*  Channel2 configuration */
        tim_choc_mode_select(TIM1, TIM_CHANNEL_NUM_2, TIM_CHxOCMSEL_PWM1);
        tim_chcc_enable_ctrl(TIM1, TIM_CHANNEL_NUM_2, TIM_CHxCC_ENABLE);
        tim_chncc_enable_ctrl(TIM1, TIM_CHANNEL_NUM_2, TIM_CHxNCC_DISABLE);
        step++;
    }
    else
    {
        /*  Channel1 configuration */
        tim_choc_mode_select(TIM1, TIM_CHANNEL_NUM_1, TIM_CHxOCMSEL_PWM1);
        tim_chcc_enable_ctrl(TIM1, TIM_CHANNEL_NUM_1, TIM_CHxCC_ENABLE);
        tim_chncc_enable_ctrl(TIM1, TIM_CHANNEL_NUM_2, TIM_CHxNCC_DISABLE);

        /*  Channel3 configuration */
        tim_choc_mode_select(TIM1, TIM_CHANNEL_NUM_3, TIM_CHxOCMSEL_PWM1);
        tim_chcc_enable_ctrl(TIM1, TIM_CHANNEL_NUM_3, TIM_CHxCC_DISABLE);
        tim_chncc_enable_ctrl(TIM1, TIM_CHANNEL_NUM_3, TIM_CHxNCC_ENABLE);

        /*  Channel2 configuration */
        tim_chcc_enable_ctrl(TIM1, TIM_CHANNEL_NUM_2, TIM_CHxCC_DISABLE);
        tim_chncc_enable_ctrl(TIM1, TIM_CHANNEL_NUM_2, TIM_CHxNCC_DISABLE);
        step = 1;
    }
}


/**
  * @}
  */

/**
  * @}
  */
```

附录 3 TIM_7PWMOutputs

```
/**
  * @file    TIM/TIM_7PWMOutputs/main.c
  * @brief   Main program body
  * @author  ChipSea MCU Group
  * @version V1.0.0
  * @date    2018.11.01
  * @copyright CHIPSEA TECHNOLOGIES (SHENZHEN) CORP.
  * @note
  * <h2><center>&copy; COPYRIGHT 2018 ChipSea</center></h2>
  *
  */


#include "main.h"


/** @addtogroup CS32F0xx_HAL_Examples
  * @{
  */

/** @addtogroup TIM_7PWMOutputs
  * @{
  */

extern uint32_t SystemCoreClock;
void time_config(void);


/**
  * @fn int main(void)
  * @brief  Main program.   MCU clock setting is  configured  through
SystemInit()
  *          in startup file (startup_cs32f0xx.s) before to enter to
application main.
  * @param  None
  * @return None
  */
int main(void)
{
    time_config();

    while (1)
    {
    }
}

/**
  * @fn void time_config(void)
  * @brief  Configure the TIM1.
  * @param  None
  * @return None
  */
void time_config(void)
{
      timer_config_t  timer_configStruct;
      timer_compare_t timer_compareStruct;
```

```
    gpio_config_t   gpio_configStruct;

    uint16_t timer_period = 0;
    uint16_t ch1_pulse = 0;
    uint16_t ch2_pulse = 0;
    uint16_t ch3_pulse = 0;
    uint16_t ch4_pulse = 0;

    //Enable GPIOA and GPIOB  Clocks
    rcu_ahb_periph_clock_enable_ctrl( RCU_AHB_PERI_PORTA |
    RCU_AHB_PERI_PORTB, ENABLE);
    rcu_apb2_periph_clock_enable_ctrl(RCU_APB2_PERI_TIM1 , ENABLE); //TIM1
clock enable  (PCLK=48MHZ)

    // GPIOA Config:TIM1 Channel 1, 2, 3, 4 and Channel 1N.
    gpio_configStruct.gpio_pin = GPIO_PIN_8 | GPIO_PIN_9 | GPIO_PIN_10 |
    GPIO_PIN_11 | GPIO_PIN_7;
    gpio_configStruct.gpio_mode = GPIO_MODE_MF;
    gpio_configStruct.gpio_speed = GPIO_SPEED_HIGH;
    gpio_configStruct.gpio_out_type = GPIO_OUTPUT_PP;
    gpio_configStruct.gpio_pull = GPIO_PULL_UP ;
    gpio_init(GPIOA, &gpio_configStruct);

    gpio_mf_config(GPIOA, GPIO_PIN_NUM8,  GPIO_MF_SEL2);
    gpio_mf_config(GPIOA, GPIO_PIN_NUM9,  GPIO_MF_SEL2);
    gpio_mf_config(GPIOA, GPIO_PIN_NUM10, GPIO_MF_SEL2);
    gpio_mf_config(GPIOA, GPIO_PIN_NUM11, GPIO_MF_SEL2);
    gpio_mf_config(GPIOA, GPIO_PIN_NUM7,  GPIO_MF_SEL2);

    //GPIOB Configuration: Channel 2N and 3N .
    gpio_configStruct.gpio_pin = GPIO_PIN_0 | GPIO_PIN_1;
    gpio_init(GPIOB, &gpio_configStruct);

    gpio_mf_config(GPIOB, GPIO_PIN_NUM0, GPIO_MF_SEL2);
    gpio_mf_config(GPIOB, GPIO_PIN_NUM1, GPIO_MF_SEL2);

    //generate signal frequency 10Khz.
    timer_period = (SystemCoreClock / 10000 ) - 1;  //4800 = 48MHZ/10KHZ
    ch1_pulse = (uint16_t) (((uint32_t) 5 * (timer_period - 1)) / 10);
//duty cycle at 50%(channel 1 and 1N)
    ch2_pulse = (uint16_t) (((uint32_t) 375 * (timer_period - 1)) /
1000);//duty cycle at 37.5%(channel 2 and 2N)
    ch3_pulse = (uint16_t) (((uint32_t) 25 * (timer_period - 1)) / 100);
//duty cycle at 25%(channel 3 and 3N)
    ch4_pulse = (uint16_t) (((uint32_t) 125 * (timer_period- 1)) / 1000); //
duty cycle at 12.5%

    //Time Base configuration
    timer_configStruct.time_period = timer_period;
    timer_configStruct.time_divide = 0;
    timer_configStruct.count_mode = TIM_COUNT_PATTERN_UP;
    timer_configStruct.clock_divide = 0;
    timer_configStruct.repeate_count = 0;
    tim_timer_config(TIM1, &timer_configStruct);

    //Channel 1, 2,3 and 4 Configuration in PWM mode.
    timer_compareStruct.time_mode = TIM_CHxOCMSEL_PWM2;
    timer_compareStruct.output_state = TIM_CHx_OUTPUT_ENABLE;
    timer_compareStruct.output_state_n = TIM_CHxNCCEN_ENABLE;
    timer_compareStruct.output_polarity = TIM_CHxCCP_POLARITY_LOW;
```

```
      timer_compareStruct.output_polarity_n = TIM_CHxNCCP_POLARITY_HIGH;
      timer_compareStruct.idle_state = TIM_IVOx_SET;
      timer_compareStruct.idle_state_n = TIM_IVOx_RESET;

      timer_compareStruct.timer_pulse = ch1_pulse;
      tim_ch1oc_init(TIM1, &timer_compareStruct);

      timer_compareStruct.timer_pulse = ch2_pulse;
      tim_ch2oc_init(TIM1, &timer_compareStruct);

      timer_compareStruct.timer_pulse = ch3_pulse;
      tim_ch3oc_init(TIM1, &timer_compareStruct);

      timer_compareStruct.timer_pulse = ch4_pulse;
      tim_ch4oc_init(TIM1, &timer_compareStruct);

      tim_enable_ctrl(TIM1, ENABLE);   //TIM1 counter enable
    tim_pwm_output_set(TIM1, ENABLE);  //TIM1 PWM Output Enable.
}


/**
  * @}
  */

/**
  * @}
  */
```

附录 4 TIM_ADCTrigger

```
/**
  * @file    TIM/TIM_ADCTrigger/main.c
  * @brief   Main program body
  * @author  ChipSea MCU Group
  * @version V1.0.0
  * @date    2018.11.01
  * @copyright CHIPSEA TECHNOLOGIES (SHENZHEN) CORP.
  * @note
  * <h2><center>&copy; COPYRIGHT 2018 ChipSea</center></h2>
  *
  */


#include "main.h"


/** @addtogroup CS32F0xx_HAL_Examples
  * @{
  */

/** @addtogroup TIM_ADC_Trigger
  * @{
  */

void time_config(void);
void adc_config(void);
void usart_com_init(void);
void usart_send(uint8_t * pbuf,uint8_t len);


/**
  * @fn int main(void)
  * @brief  Main program.   MCU clock setting is  configured  through
SystemInit()
  *          in startup file (startup_cs32f0xx.s) before to enter to
application main.
  * @param  None
  * @return None
  */
int main(void)
{
      uint16_t  sample_value = 0;
      uint16_t  real_value = 0;
      uint8_t  tx_buf[60];
      uint8_t print_len = 0;

      cs_eval_led2_init();
      usart_com_init();

      time_config();
      adc_config();

      while (1)
      {
            while(adc_flag_status_get(ADC1, ADC_FLAG_EOCH) == RESET); //Test
EOC flag
```

```
            cs_eval_led2_toggle();
            memset((char *)tx_buf, 0, 60);

            sample_value =adc_conversion_value_get(ADC1); // Get ADC1
converted data
            real_value = (sample_value *3300)/0xFFF; // Compute the voltage
            print_len = sprintf((char *)tx_buf,"ADC CH0 Value=%d
mV.\r\n",real_value);
            usart_send(tx_buf,print_len);
      }
}

/**
  * @fn void time_config(void)
  * @brief  ADC and TIM configuration
  * @param  None
  * @return None
  */
void time_config(void)
{
      timer_config_t  timer_configStruct;
      timer_compare_t timer_compareStruct;

      rcu_apb2_periph_clock_enable_ctrl(RCU_APB2_PERI_TIM1, ENABLE);//TIM1
clock enable.

      tim_def_init(TIM1);
      tim_config_struct_init(&timer_configStruct);
      tim_compare_struct_init(&timer_compareStruct);

      //Time base configuration.
      timer_configStruct.time_period = 0xFF;
      timer_configStruct.time_divide = 0x0;
      timer_configStruct.clock_divide = 0x0;
      timer_configStruct.count_mode = TIM_COUNT_PATTERN_UP;
      tim_timer_config(TIM1, &timer_configStruct);

      //Output Compare PWM Mode configuration.
      timer_compareStruct.time_mode = TIM_CHxOCMSEL_PWM1; //low edge by
default
      timer_compareStruct.output_state = TIM_CHx_OUTPUT_ENABLE;
      timer_compareStruct.timer_pulse = 0x01;
      tim_ch4oc_init(TIM1, &timer_compareStruct);

      tim_enable_ctrl(TIM1, ENABLE); //TIM1 enable counter.
      tim_pwm_output_set(TIM1, ENABLE);  //Main Output Enable
}

/**
  * @fn  void adc_config(void)
  * @brief  ADC and TIM configuration
  * @param  None
  * @return None
  */
void adc_config(void)
{
      adc_config_t   adc_configStruct;
      gpio_config_t  gpio_configStruct;

      rcu_ahb_periph_clock_enable_ctrl(RCU_AHB_PERI_PORTA, ENABLE); // GPIOA
```

```
Periph clock enable
      rcu_apb2_periph_clock_enable_ctrl(RCU_APB2_PERI_ADC, ENABLE); //ADC1 and
TIM1 Periph clock enable

      gpio_configStruct.gpio_pin = GPIO_PIN_0 ; // Configure ADC CH0 as analog
input
      gpio_configStruct.gpio_mode = GPIO_MODE_AN;
      gpio_configStruct.gpio_pull = GPIO_PULL_NO_PULL ;
      gpio_init(GPIOA, &gpio_configStruct);

      adc_def_init(ADC1);
      adc_config_struct_init(&adc_configStruct);

      //Configure the ADC1 in continous mode
      adc_configStruct.adc_resolution = ADC_CONV_RES_12BITS;
      adc_configStruct.conversion_mode = ENABLE;
      adc_configStruct.trigger_mode = ADC_TRIG_MODE_SEL_RISING;
      adc_configStruct.hardware_trigger =  ADC_HW_TRIG_SEL_T1_CH4CC;
//TIM1_CH4CC TRIG ADC
      adc_configStruct.data_align = ADC_DATA_ALIGN_RIGHT;
      adc_configStruct.scan_direction = ADC_CONV_SEQ_DIR_UPWARD;
      adc_init(ADC1, &adc_configStruct);

      adc_channel_config(ADC1, ADC_CONV_CHANNEL_0, ADC_SAMPLE_TIMES_239_5); //
ADC1 CH0 with 239.5 Cycles as sampling time
      adc_calibration_value_get(ADC1);  // ADC Calibration

      adc_enable_ctrl(ADC1, ENABLE);     // Enable the ADC
      while(!adc_flag_status_get(ADC1, ADC_FLAG_EOI));  // Wait the EOI flag

      adc_conversion_start(ADC1); // ADC1 regular Software Start Conv
}


/**
 * @fn void usart_com_init(void)
 * @brief  Initializes USART.
 * @param  None
 * @return None
 */
void usart_com_init(void)
{
      usart_config_t usart_configStruct;
      gpio_config_t  gpio_configStruct;

   // Clock Config
      rcu_ahb_periph_clock_enable_ctrl(RCU_AHB_PERI_PORTA, ENABLE);
      rcu_apb2_periph_clock_enable_ctrl(RCU_APB2_PERI_USART1, ENABLE);

    // GPIO MF Config
      gpio_mf_config(GPIOA, GPIO_PIN_NUM9, GPIO_MF_SEL1);
      gpio_mf_config(GPIOA, GPIO_PIN_NUM10, GPIO_MF_SEL1);

      gpio_configStruct.gpio_pin = GPIO_PIN_9 | GPIO_PIN_10;       //PA9,PA10
USART
      gpio_configStruct.gpio_mode = GPIO_MODE_MF;
      gpio_configStruct.gpio_speed = GPIO_SPEED_MEDIUM;
      gpio_configStruct.gpio_out_type = GPIO_OUTPUT_PP;
      gpio_configStruct.gpio_pull = GPIO_PULL_NO_PULL;
      gpio_init(GPIOA, &gpio_configStruct);
```

```
	// USART Config
	usart_def_init(USART1);
	usart_configStruct.usart_rate = 115200;
	usart_configStruct.data_width = USART_DATA_WIDTH_8;
	usart_configStruct.stop_bits = USART_STOP_BIT_1;
	usart_configStruct.usart_parity = USART_PARITY_NO;
	usart_configStruct.flow_control = USART_FLOW_CONTROL_NONE;
	usart_configStruct.usart_mode = USART_MODE_RX | USART_MODE_TX;
	usart_init(USART1, &usart_configStruct);

	usart_enable_ctrl(USART1,ENABLE);
}

/**
 * @fn void usart_send(uint8_t * pbuf,uint8_t len)
 * @brief  Transmits  data through the Usart peripheral.
 * @param  pbuf: The pointer to buffer being transmited data.
 * @param  len: The data length.
 * @return None
 */
void usart_send(uint8_t * pbuf,uint8_t len)
{
	uint8_t i = 0;

	for(i = 0; i < len; i++)
	{
		usart_data_send(USART1,pbuf[i]);
		while(usart_flag_status_get(USART1,USART_FLAG_TXE) == RESET);
	}
}


/**
  * @}
  */

/**
  * @}
  */
```