11. STM32的定时器及PWM

北京科技大学

计算机与通信工程学院

目录

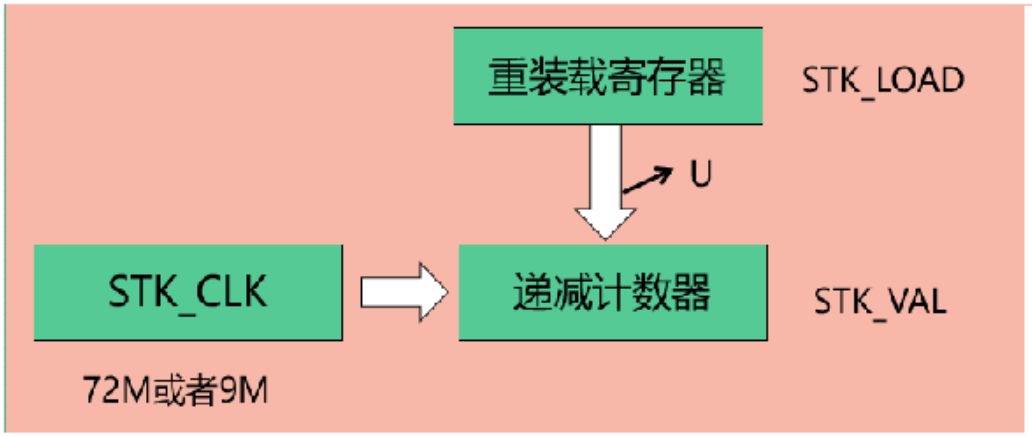
CONTENTS.

STM32-10

回顾

SysTick简介

SysTick：系统定时器，24位，只能递减，存在于内核，嵌套在NVIC中，所有的Cortex-M内核的单片机都具有这个定时器。



counter在时钟的驱动下，从reload初值开始往下递减计数到0，产生中断和置位COUNTFLAG标志。然后又从reload值开始重新递减计数，如此循环。

定时器简介

定时器功能： 定时、输出比较、输入捕获、互补输出

定时器分类： 基本定时器、通用定时器、高级定时器

定时器资源： F103系列有2个高级定时器TIM1和TIM8、4个通用定时器TIM2/3/4/5、2个基本定时器TIM6和TIM7

定时器简介

定时器分类

	定时器	计数器分辨率	计数器类型	预分频系数	产生DMA	捕获/比较通道	互补输出
高级定时器	TIM1	16位	向上/向下	1~65535	可以	4	有
	TIM8	16位	向上/向下	1~65535	可以	4	有
通用定时器	TIM2	16位	向上/向下	1~65535	可以	4	没有
	TIM3	16位	向上/向下	1~65535	可以	4	没有
	TIM4	16位	向上/向下	1~65535	可以	4	没有
	TIM5	16位	向上/向下	1~65535	可以	4	没有
基本定时器	TIM6	16位	向上	1~65535	可以	0	没有
	TIM7	16位	向上	1~65535	可以	0	没有

基本定时器功能框图讲解

基本定时器功能简介

1-计数器16bit，只能向上计数，只有TIM6和TIM7

2-没有外部的GPIO，是内部资源，只能用来定时

3-时钟来自PCLK1=36Mhz，可实现1~65536分频

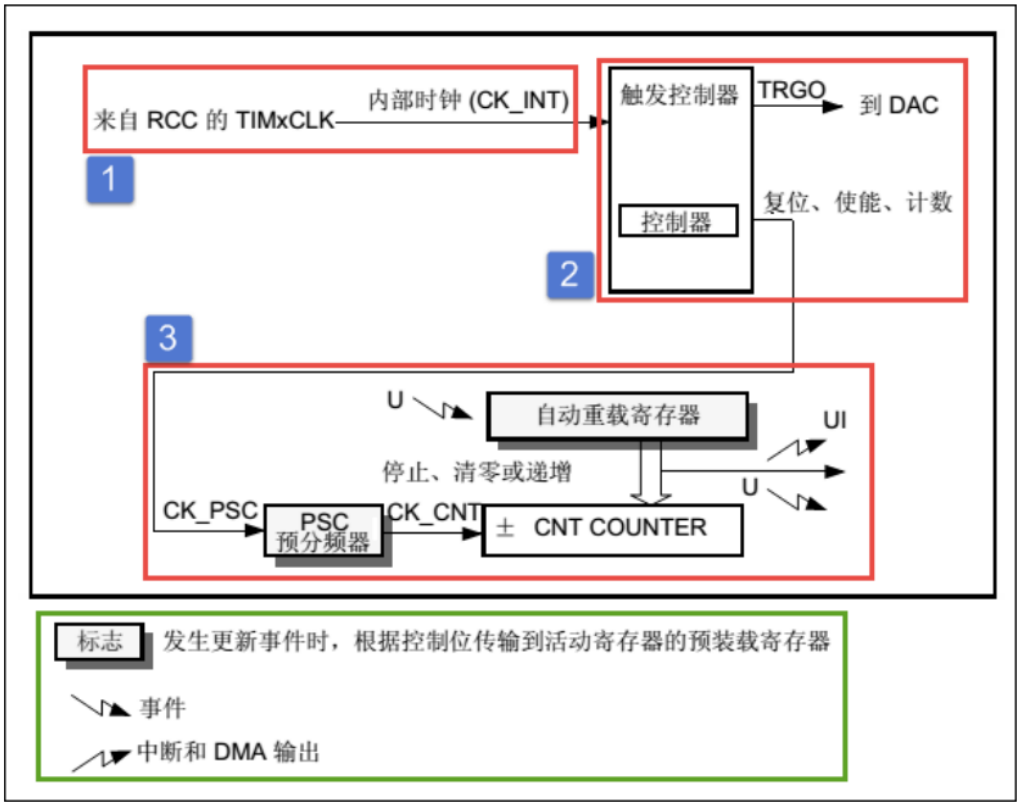
基本定时器功能框图讲解

1-时钟源

2-控制器

3-时基单元

定时周期的计算



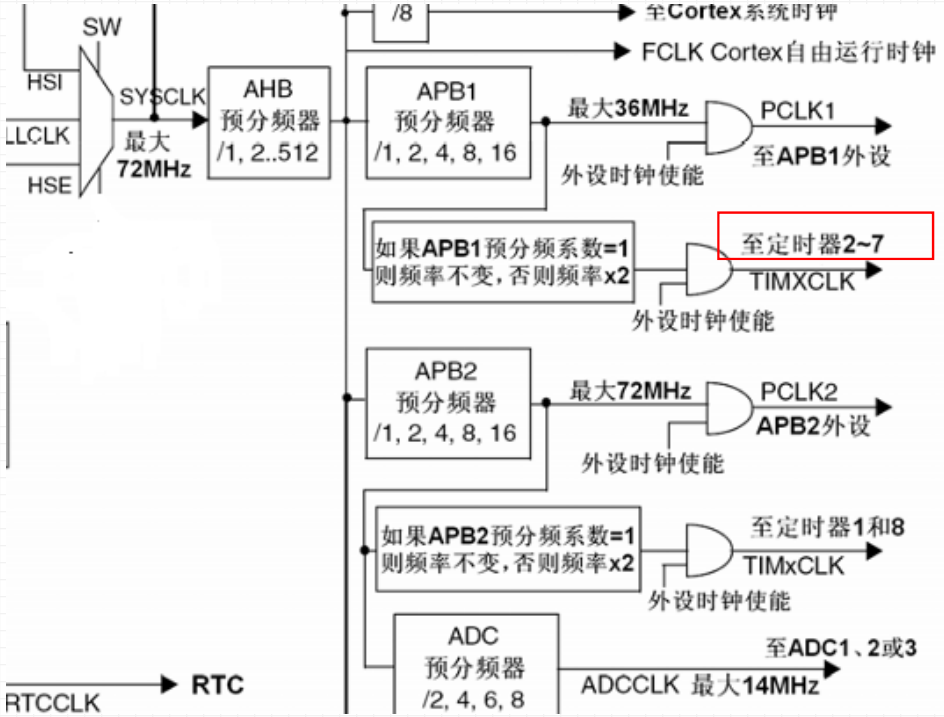
基本定时器功能框图讲解

时钟源

1-时钟源来自RCC的TIMx_CLK（属于内部的CK_INT）

2-TIMx_CLK等于多少呢？如何确定？

具体的查看：RCC时钟树框图



基本定时器功能框图讲解

控制器

1-控制器用于控制定时器的：复位、使能、计数、触发DAC

2、涉及到的寄存器为：CR1/2、DIER、EGR、SR

基本定时器功能框图讲解

时基（定时器的心脏）

定时器最主要的就是时基部分：包括 **预分频器、计数器、自动重装载寄存器。**

基本定时器功能框图讲解

预分频器

1-16位的预分频器PSC对内部时钟CK_PSC进行分频之后，得到计数器时钟CK_CNT=CK_PSC/(PSC+1)

2-计数器CNT在计数器时钟的驱动下开始计数，计数一次的时间为1/CK_CNT

基本定时器功能框图讲解

计数器、自动重装载寄存器

定时器使能(CEN 置 1)后，计数器 CNT在CK_CNT 驱动下计数，当 TCNT 值与 ARR 的设定值相等时就自动生成事件并 CNT 自动清零，然后自动重新开始计数，如此重复以上过程。

基本定时器功能框图讲解

影子寄存器

1-PSC和ARR都有影子寄存器，功能框图上有个影子

2-影子寄存器的存在起到一个缓冲的作用，用户值->寄存器->影子寄存器->起作用，如果不使用影子寄存器则用户值在写到寄存器之后则里面起作用。

ARR影子，TIMx_CR1:APRE位控制

基本定时器功能框图讲解

定时时间的计算

如何实现500mS的定时

基本定时器功能框图讲解

定时时间的计算

1、PSC = 36-1，定时器频率

=36M/(PSC+1)=1MHZ

2、ARR = 1000-1，从0计数到999，则计了1000次

3、中断周期T = 1000 *1/1000000 = 1mS

时基初始化结构体

```
typedef struct
{
  // 分频因子
  uint16_t TIM_Prescaler;

  // 计数模式，基本定时器只能向上计数
  uint16_t TIM_CounterMode;

  // 自动重装载值
  uint32_t TIM_Period;

  // 外部输入时钟分频因子，基本定时器没有
  uint16_t TIM_ClockDivision;

  // 重复计数器，基本定时器没有，高级定时器专用
  uint8_t TIM_RepetitionCounter;
} TIM_TimeBaseInitTypeDef;
```

TIM6 和 TIM7 的寄存器里面只有 TIM_Prescaler 和 TIM_Period，另外三个成员基本定时器是没有的，

基本定时器示例

任务：基本定时器 TIM6/7 定时 1s，1s 时间到 LED 翻转一次。

1. 编程要点
(1) 开定时器时钟 TIMx_CLK, x[6,7];
(2) 初始化时基初始化结构体;
(3) 使能 TIMx, x[6,7] update 中断;
(4) 打开定时器;
(5) 编写中断服务程序

```
1 /************************基本定时器 TIM 参数定义，只限 TIM6、7*************/
5 #define              BASIC_TIM                       TIM6
 6 #define              BASIC_TIM_APBxClock_FUN         RCC_APB1PeriphClockCmd
 7 #define              BASIC_TIM_CLK                   RCC_APB1Periph_TIM6
 8 #define              BASIC_TIM_IRQ                   TIM6_IRQn
 9 #define              BASIC_TIM_IRQHandler            TIM6_IRQHandler
```

基本定时器示例

基本定时器模式配置

```
1 void BASIC_TIM_Config(void)
 2 {
 3     TIM_TimeBaseInitTypeDef  TIM_TimeBaseStructure;
 4
 5     // 开启定时器时钟,即内部时钟 CK_INT=36M
 6     BASIC_TIM_APBxClock_FUN(BASIC_TIM_CLK, ENABLE);
 7
 8     // 自动重装载寄存器周的值(计数值)
 9     TIM_TimeBaseStructure.TIM_Period=1000;
10
11    // 累计 TIM_Period 个频率后产生一个更新或者中断
12 // 时钟预分频数为 35，则驱动计数器的时钟 CK_CNT = CK_INT / (35+1)=1M
13     TIM_TimeBaseStructure.TIM_Prescaler= 35;
14
15     // 时钟分频因子 ，基本定时器没有，不用管
16     //TIM_TimeBaseStructure.TIM_ClockDivision=TIM_CKD_DIV1;
17
```

基本定时器示例

```
18      // 计数器计数模式，基本定时器只能向上计数，没有计数模式的设置
19      //TIM_TimeBaseStructure.TIM_CounterMode=TIM_CounterMode_Up;
20
21      // 重复计数器的值，基本定时器没有，不用管
22      //TIM_TimeBaseStructure.TIM_RepetitionCounter=0;
23
24      // 初始化定时器
25      TIM_TimeBaseInit(BASIC_TIM, &TIM_TimeBaseStructure);
26
27      // 清除计数器中断标志位
28      TIM_ClearFlag(BASIC_TIM, TIM_FLAG_Update);
29
30      // 开启计数器中断
31      TIM_ITConfig(BASIC_TIM,TIM_IT_Update,ENABLE);
32
33      // 使能计数器
34      TIM_Cmd(BASIC_TIM, ENABLE);
35
36      // 暂时关闭定时器的时钟，等待使用
37      BASIC_TIM_APBxClock_FUN(BASIC_TIM_CLK, DISABLE);
38  }
```

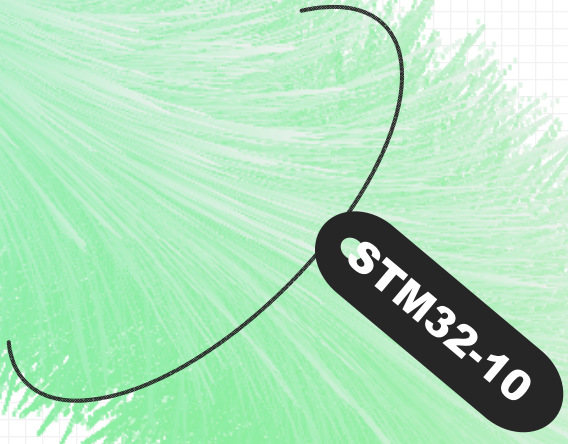
基本定时器示例

```c
void  BASIC_TIM_IRQHandler (void)
{
    if ( TIM_GetITStatus( BASIC_TIM, TIM_IT_Update) != RESET ) {
        time++;
        TIM_ClearITPendingBit(BASIC_TIM, TIM_FLAG_Update);
    }
}
```

```c
int main(void)
{
    /* led 端口配置 */
    LED_GPIO_Config();

    /* 基本定时器 TIMx,x[6,7] 定时配置 */
    BASIC_TIM_Config();

    /* 配置基本定时器 TIMx,x[6,7]的中断优先级 */
    BASIC_TIM_NVIC_Config();

    /* 基本定时器 TIMx,x[6,7] 重新开时钟，开始计时 */
    BASIC_TIM_APBxClock_FUN(BASIC_TIM_CLK, ENABLE);

    while (1) {
        if ( time == 1000 ) { /* 1000 * 1 ms = 1s 时间到 */
            time = 0;
            /* LED1 取反 */
            LED1_TOGGLE;
        }
    }
}
```

目录

CONTENTS.



定时器简介

定时器功能： 定时、输出比较、输入捕获、断路输入

定时器分类： 基本定时器、通用定时器、高级定时器

定时器资源： F103有2个高级定时器、4个通用定时器、2个基本定时器

定时器简介

定时器特性

	定时器	计数器分辨率	计数器类型	预分频系数	产生DMA	捕获/比较通道	互补输出
高级定时器	TIM1	16位	向上/向下	1~65535	可以	4	有
	TIM8	16位	向上/向下	1~65535	可以	4	有
通用定时器	TIM2	16位	向上/向下	1~65535	可以	4	没有
	TIM3	16位	向上/向下	1~65535	可以	4	没有
	TIM4	16位	向上/向下	1~65535	可以	4	没有
	TIM5	16位	向上/向下	1~65535	可以	4	没有
基本定时器	TIM6	16位	向上	1~65535	可以	0	没有
	TIM7	16位	向上	1~65535	可以	0	没有

高级定时器简介

高级定时器功能简介

1-计数器16bit，上/下/两边 计数，TIM1和TIM8，还有一个重复计数器RCR，独有。

2-有4个GPIO，其中通道1~3还有互补输出GPIO

3-时钟来自PCLK2，为72M，可实现1~65536分频

高级定时器GPIO说明

表 33-1 高级控制和通用定时器通道引脚分布

	高级定时器		通用定时器			
	TIM1	TIM8	TIM2	TIM5	TIM3	TIM4
CH1	PA8/PE9	PC6	PA0/PA15	PA0	PA6/PC6/PB4	PB6/PD12
CH1N	PB13/PA7/PE8	PA7				
CH2	PA9/PE11	PC7	PA1/PB3	PA1	PA7/PC7/PB5	PB7/PD13
CH2N	PB14/PB0/PE10	PB0				
CH3	PA10/PE13	PC8	PA2/PB10	PA2	PB0/PC8	PB8/PD14
CH3N	PB15/PB1/PE12	PB1				
CH4	PA11/PE14	PC9	PA3/PB11	PA3	PB1/PC9	PB9/PD15
ETR	PA12/PE7	PA0	PA0/PA15		PD2	PE0
BKIN	PB12/PA6/PE15	PA6				

其他型号STM32可参考数据手册的引脚说明章节

高级定时器功能框图讲解

1-时钟源

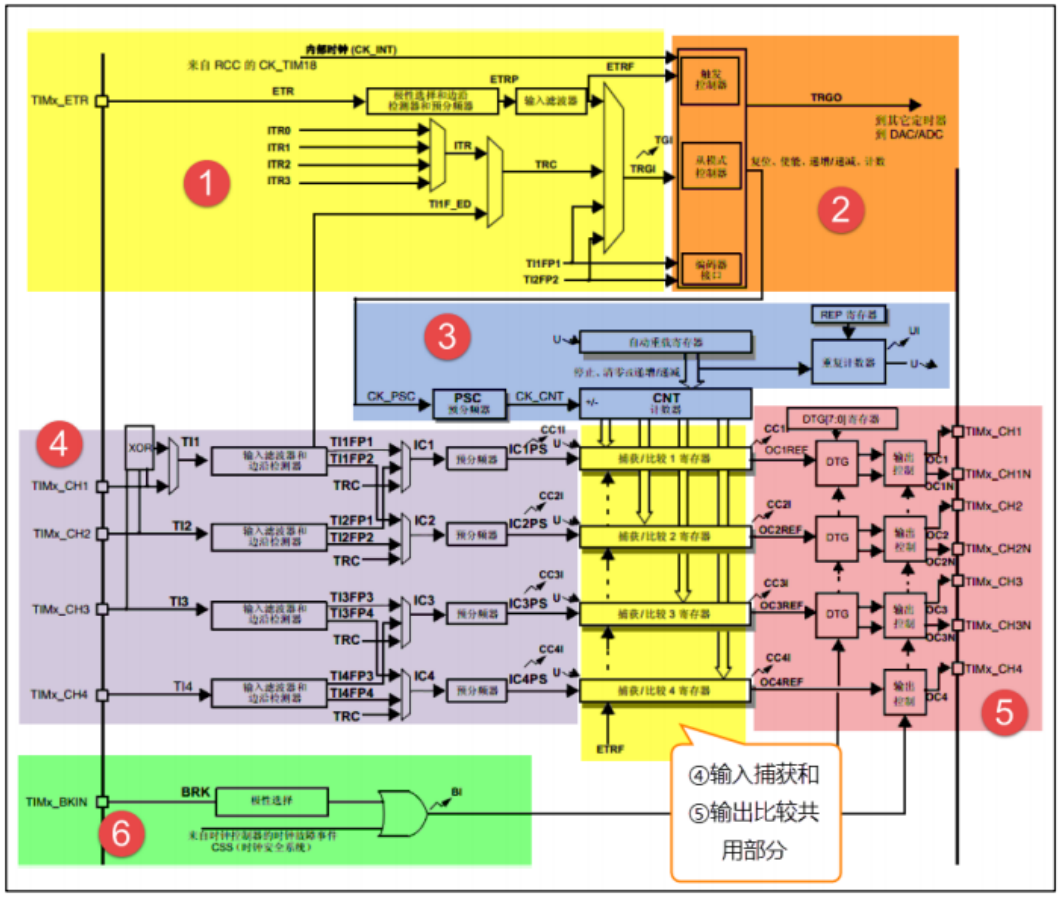
2-控制器

3-时基

4-输入捕获

5-输出比较

6-断路功能



高级定时器功能框图讲解

一、时钟源

1-内部时钟源CK_INT

2-外部时钟模式1—外部的GPIO Tix（x=1 2 3 4）

3-外部时钟模式2—外部的GPIO ETR

4-内部触发输入

高级定时器功能框图讲解

内部时钟源

1-内部时钟源来自RCC的TIMx_CLK

2-TIMx_CLK等于多少呢？如何确定？

具体的查看：RCC时钟树部分

高级定时器功能框图讲解

二、控制器

1-控制器就是用来控制的，发送命令的

2-CR1、CR2、SMCR、CCER，主要学习这几个寄存器即可。

高级定时器功能框图讲解

三、时基单元

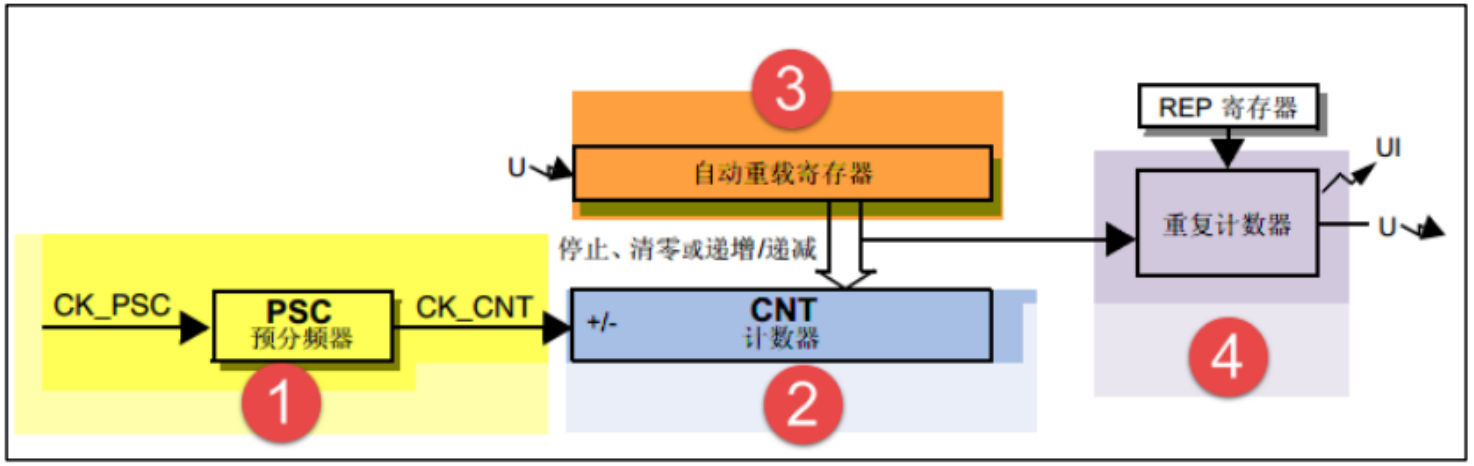


图 32-5 高级定时器时基单元

高级定时器功能框图讲解

时基单元的组成

1-16位的预分频器 PSC，PSC

2-16位的计数器CNT， CNT

3-8位的重复计数器RCR，RCR（高级定时器独有）

4-16位的自动重装载寄存器ARR，ARR

高级定时器功能框图讲解

预分频器

预分频器 PSC

预分频器 PSC，有一个输入时钟 CK_PSC 和一个输出时钟 CK_CNT。输入时钟 CK_PSC 就是上面时钟源的输出，输出 CK_CNT 则用来驱动计数器 CNT 计数。通过设置预分频器 PSC 的值可以得到不同的 CK_CNT，实际计算为：f_{CK_CNT} 等于 f_{CK_PSC}/(PSC[15:0]+1)，可以实现 1 至 65536 分频。

高级定时器功能框图讲解

计数器（上/下/两边）

(1) 递增计数模式下，计数器从 0 开始计数，每来一个 CK_CNT 脉冲计数器就增加 1，直到计数器的值与自动重载寄存器 ARR 值相等，然后计数器又从 0 开始计数并生成计数器上溢事件，计数器总是如此循环计数。如果禁用重复计数器，在计数器生成上溢事件就马上生成更新事件(UEV)；如果使能重复计数器，每生成一次上溢事件重复计数器内容就减 1，直到重复计数器内容为 0 时才会生成更新事件。

(2) 递减计数模式下，计数器从自动重载寄存器 ARR 值开始计数，每来一个 CK_CNT 脉冲计数器就减 1，直到计数器值为 0，然后计数器又从自动重载寄存器 ARR 值开始递减计数并生成计数器下溢事件，计数器总是如此循环计数。如果禁用重复计数器，在计数器生成下溢事件就马上生成更新事件；如果使能重复计数器，每生成一次下溢事件重复计数器内容就减 1，直到重复计数器内容为 0 时才会生成更新事件。

(3) 中心对齐模式下，计数器从 0 开始递增计数，直到计数值等于(ARR-1)值生成计数器上溢事件，然后从 ARR 值开始递减计数直到 1 生成计数器下溢事件。然后又从 0 开始计数，如此循环。每次发生计数器上溢和下溢事件都会生成更新事件。

高级定时器功能框图讲解

自动重装载寄存器

自动重载寄存器 ARR

自动重载寄存器 ARR 用来存放与计数器 CNT 比较的值，如果两个值相等就递减重复计数器。可以通过 TIMx_CR1 寄存器的 ARPE 位控制自动重载影子寄存器功能，如果 ARPE 位置 1，自动重载影子寄存器有效，只有在事件更新时才把 TIMx_ARR 值赋给影子寄存器。如果 ARPE 位为 0，则修改 TIMx_ARR 值马上有效。

高级定时器功能框图讲解

重复计数器

重复计数器寄存器（TIMx_RCR），核心作用是：**让定时器的“更新事件（UEV）”延迟触发**——默认情况下定时器溢出 1 次就产生 1 个更新事件，配置 RCR 后，需溢出 (RCR 值 + 1) 次才产生 1 个更新事件，进而间接控制 PWM 周期、输出比较周期或定时器中断周期“放大 (RCR+1) 倍”。

核心用途：解决“高分辨率 + 长周期”的矛盾

如果需要长周期（如低频 PWM、低频中断），常规方案有两个缺点：

增大 ARR：但 ARR 是 16 位（最大 65535），若 F_timer=72MHz、PSC=0，最大基础周期仅～910us，无法满足“秒级”长周期；

增大 PSC：会降低定时器分辨率（占空比调节精度），比如 PSC=71999 时，定时器时钟 = 1kHz，占空比最小调节步长仅 1ms，无法实现高精度控制。

而 RCR 可以**“精度不变，周期放大”**，完美解决这个矛盾：

用 ARR + PSC 保证高分辨率（如 PSC=0、ARR=65535 → 基础周期～910us，分辨率 13.89ns）；

用 RCR 放大周期（如 RCR=1094 → 1095 次基础溢出 → 周期～1s，且分辨率仍保持 13.89ns）。

高级定时器功能框图讲解

四、输入捕获

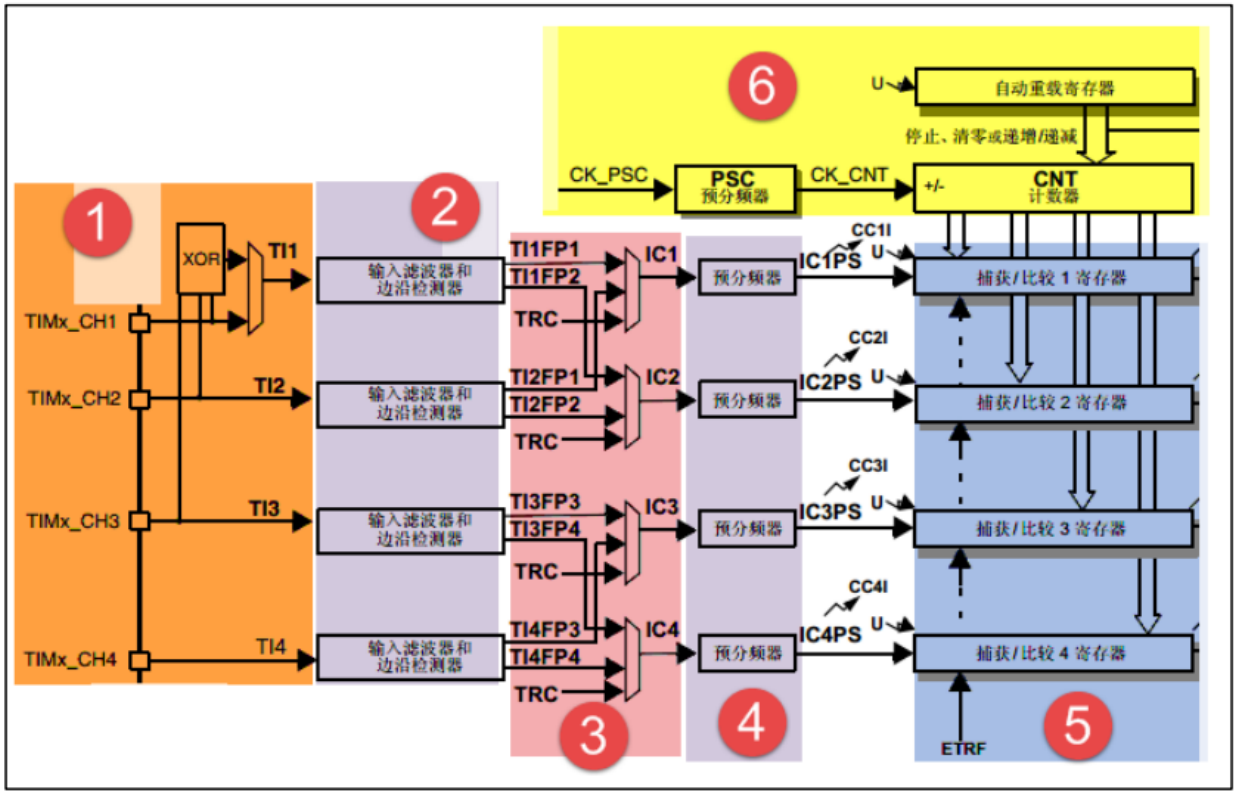


图 32-6 输入捕获功能框图

高级定时器功能框图讲解

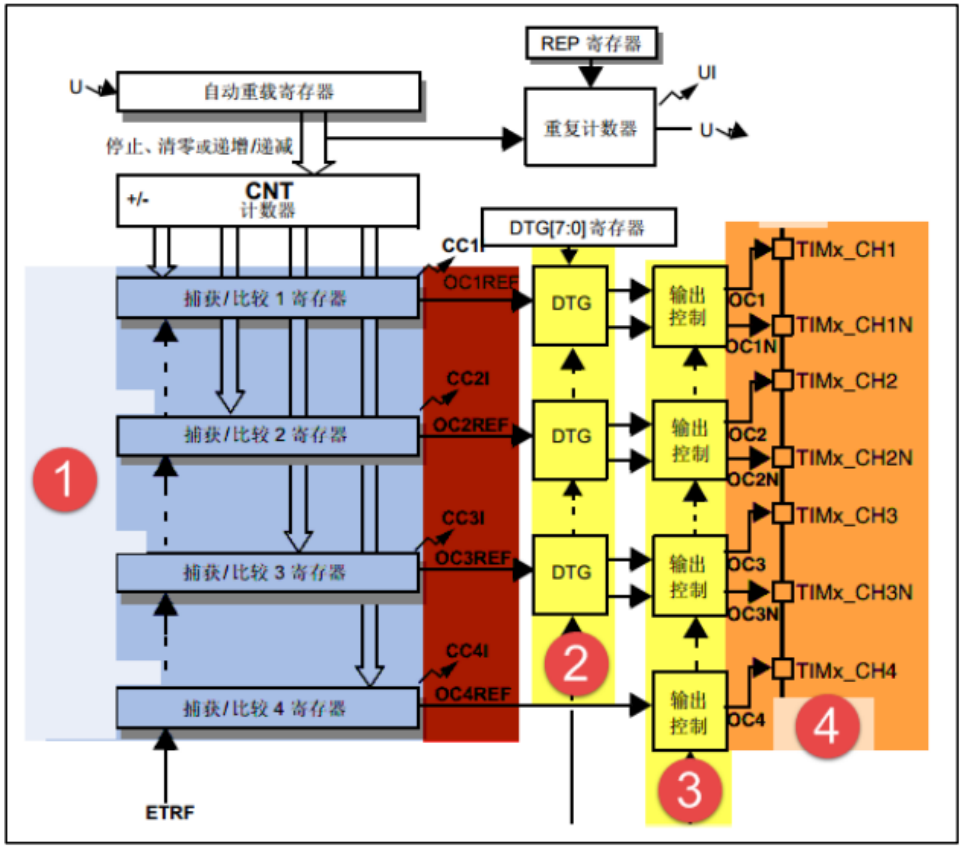
输入捕获的作用和原理

输入捕获可以对输入的信号的上升沿，下降沿或者双边沿进行捕获，常用的有测量输入信号的脉宽和测量 PWM 输入信号的频率和占空比这两种。

输入捕获的大概的原理就是，当捕获到信号的跳变沿的时候，把计数器 CNT 的值锁存到捕获寄存器 CCR 中，把前后两次捕获到的 CCR 寄存器中的值相减，就可以算出脉宽或者频率。如果捕获的脉宽的时间长度超过你的捕获定时器的周期，就会发生溢出，这个我们需要做额外的处理。

高级定时器功能框图讲解

五、输出比较



高级定时器功能框图讲解

输出比较的作用

输出比较就是通过定时器的外部引脚对外输出控制信号，有冻结、将通道 X（x=1,2,3,4）设置为匹配时输出有效电平、将通道 X 设置为匹配时输出无效电平、翻转、强制变为无效电平、强制变为有效电平、PWM1 和 PWM2 这八种模式，具体使用哪种模式由寄存器 CCMRx 的位 OCxM[2:0]配置。其中 PWM 模式是输出比较中的特例，使用的也最多。

高级定时器功能框图讲解

①输出比较寄存器

①比较寄存器

当计数器 CNT 的值跟比较寄存器 CCR 的值相等的时候，输出参考信号 OCxREF 的信号的极性就会改变，其中 OCxREF=1（高电平）称之为有效电平，OCxREF=0（低电平）称之为无效电平，并且会产生比较中断 CCxI，相应的标志位 CCxIF（SR 寄存器中）会置位。然后 OCxREF 再经过一系列的控制之后就成为真正的输出信号 OCx/OCxN。

高级定时器功能框图讲解

②死区发生器

在生成的参考波形 OCxREF 的基础上，可以插入死区时间，用于生成两路互补的输出信号 OCx 和 OCxN，死区时间的大小具体由 BDTR 寄存器的位 DTG[7:0]配置。死区时间的大小必须根据与输出信号相连接的器件及其特性来调整。下面我们简单举例说明下带死区的 PWM 信号的应用，我们以一个板桥驱动电路为例。

高级定时器功能框图讲解

带死区插入的半桥驱动电路

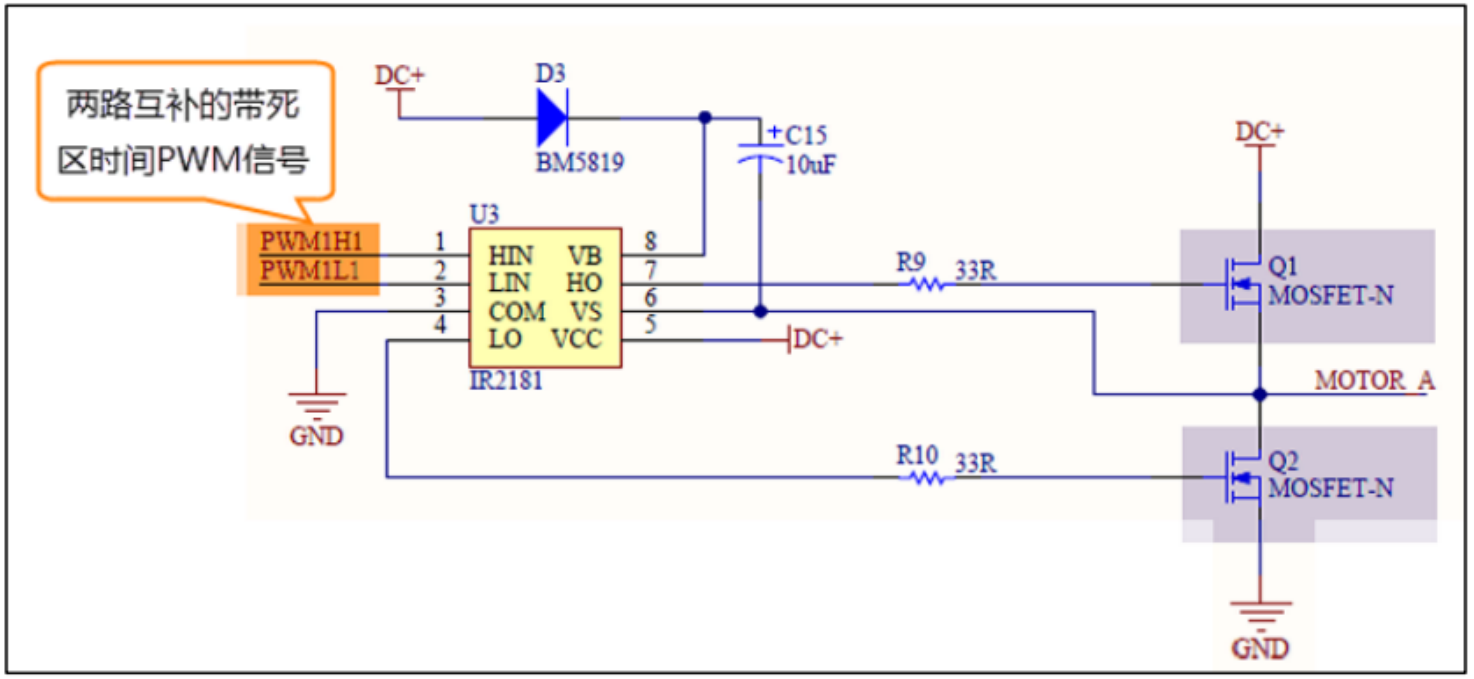


图 32-8 半桥驱动电路

高级定时器功能框图讲解

带死区插入的互补输出波形图

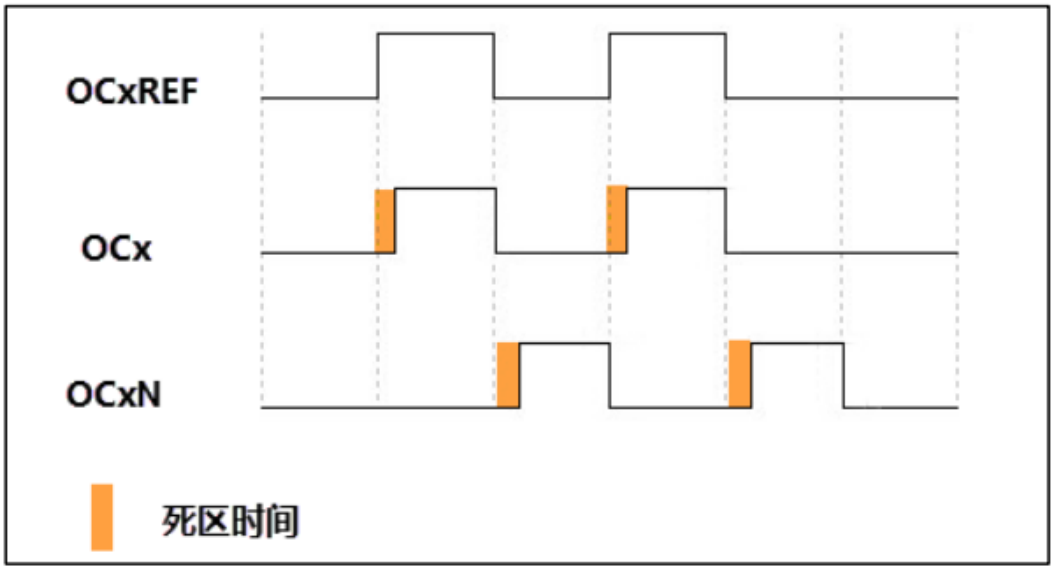
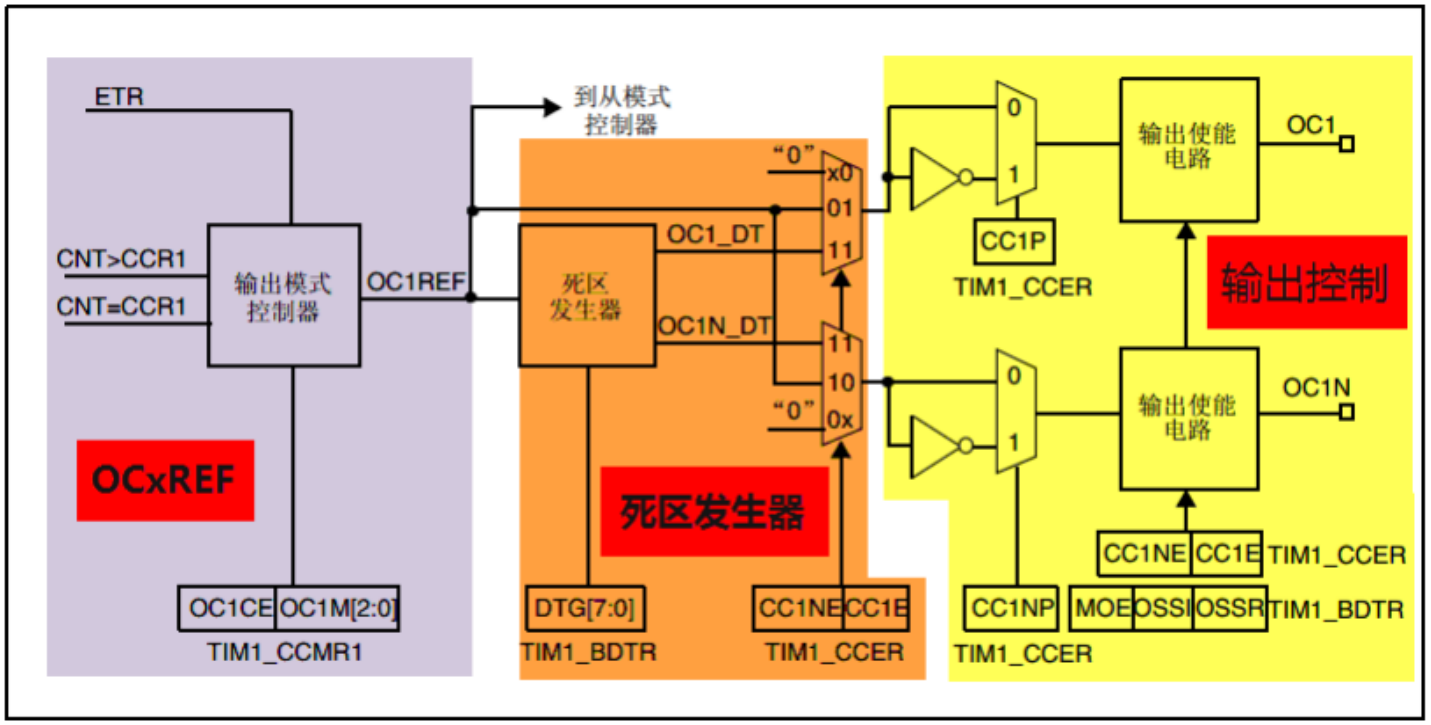


图 32-9 带死区插入的互补输出

高级定时器功能框图讲解

③输出控制



高级定时器功能框图讲解

③输出控制

在输出比较的输出控制中，参考信号 OCxREF 在经过死区发生器之后会产生两路带死区的互补信号 OCx_DT 和 OCxN_DT（通道 1~3 才有互补信号，通道 4 没有，其余跟通道 1~3 一样），这两路带死区的互补信号然后就进入输出控制电路，如果没有加入死区控制，那么进入输出控制电路的信号就直接是 OCxREF。

进入输出控制电路的信号会被分成两路，一路是原始信号，一路是被反向的信号，具体的由寄存器 CCER 的位 CCxP 和 CCxNP 控制。经过极性选择的信号是否由 OCx 引脚输出到外部引脚 CHx/CHxN 则由寄存器 CCER 的位 CxE/CxNE 配置。

如果加入了断路（刹车）功能，则断路和死区寄存器 BDTR 的 MOE、OSSI 和 OSSR 这三个位会共同影响输出的信号。

高级定时器功能框图讲解

④输出引脚

输出比较的输出信号最终是通过定时器的外部 IO 来输出的，分别为 CH1/2/3/4，其中前面三个通道还有互补的输出通道 CH1/2/3N。更加详细的 IO 说明还请查阅相关的数据手册。

输出比较的应用

PWM输出模式

PWM 输出就是对外输出脉宽（即占空比）可调的方波信号，信号频率由自动重装寄存器 ARR 的值决定，占空比由比较寄存器 CCR 的值决定。

脉冲宽度调制

脉冲宽度调制（Pulse Width Modulation，PWM）是利用微处理器的数字输出对模拟电路进行控制的一种非常有效的技术，广泛应用于从测量、通信到功率控制与变换等许多领域中。

脉冲宽度调制以其控制简单、灵活和动态响应好的优点而成为电工电子技术中广泛应用的控制方式，也是人们研究的热点。

例：实现LED亮度调节，直流电机转速调节

PWM原理

- **占空比**(正占空比，duty cycle)是指脉冲信号的通电时间与通电周期之比，即高电平的时间占整个周期的比例。

- 下图是一个周期是 10ms，即频率是 100Hz 的波形，但是每个周期内，高低电平脉冲宽度各不相同，这就是 PWM 的本质。比如第一部分波形的占空比是 40%，第二部分波形占空比是 60%，第三部分波形占空比是 80%，这就是 PWM 的解释。

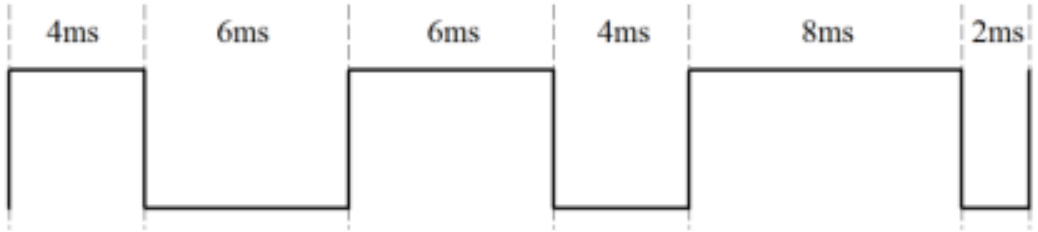


图 10-1　PWM 波形

输出比较的应用

PWM输出模式—分类

表格 32-1 PWM1 与 PWM2 模式的区别

模式	计数器 CNT 计算方式	说明
PWM1	递增	CNT<CCR，通道 CH 为有效，否则为无效
	递减	CNT>CCR，通道 CH 为无效，否则为有效
PWM2	递增	CNT<CCR，通道 CH 为无效，否则为有效
	递减	CNT>CCR，通道 CH 为有效，否则为无效

有效：高电平；无效：低电平

输出比较的应用

边沿对齐 VS 中心对齐

1-根据CNT的计数方向，PWM波形分成边沿对齐和中心对齐两种。边沿对齐主要用于直流电机，中心对齐主要用于交流电机。

2-边沿对齐时，CNT只工作在递增或者递减。

3-中心对齐时，CNT工作在递增和递减。

输出比较的应用

PWM1边沿对齐模式的波形

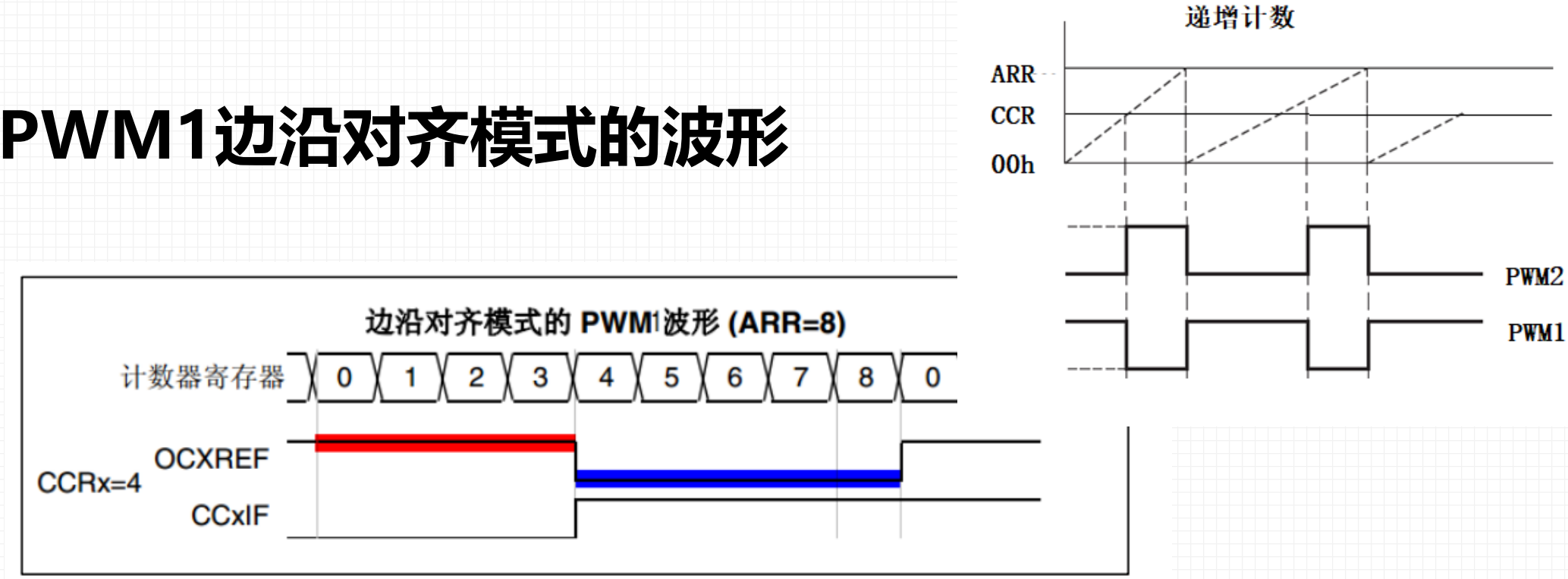


图 32-14 PWM1 模式的边沿对齐波形

输出比较的应用

2. PWM 中心对齐模式

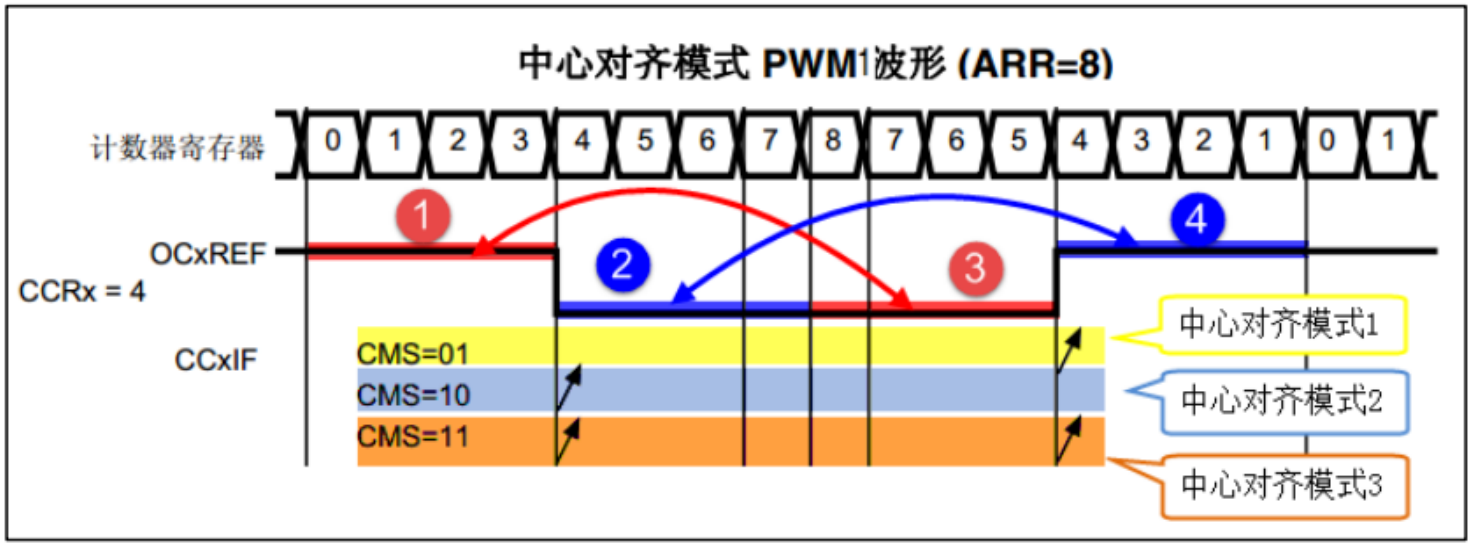
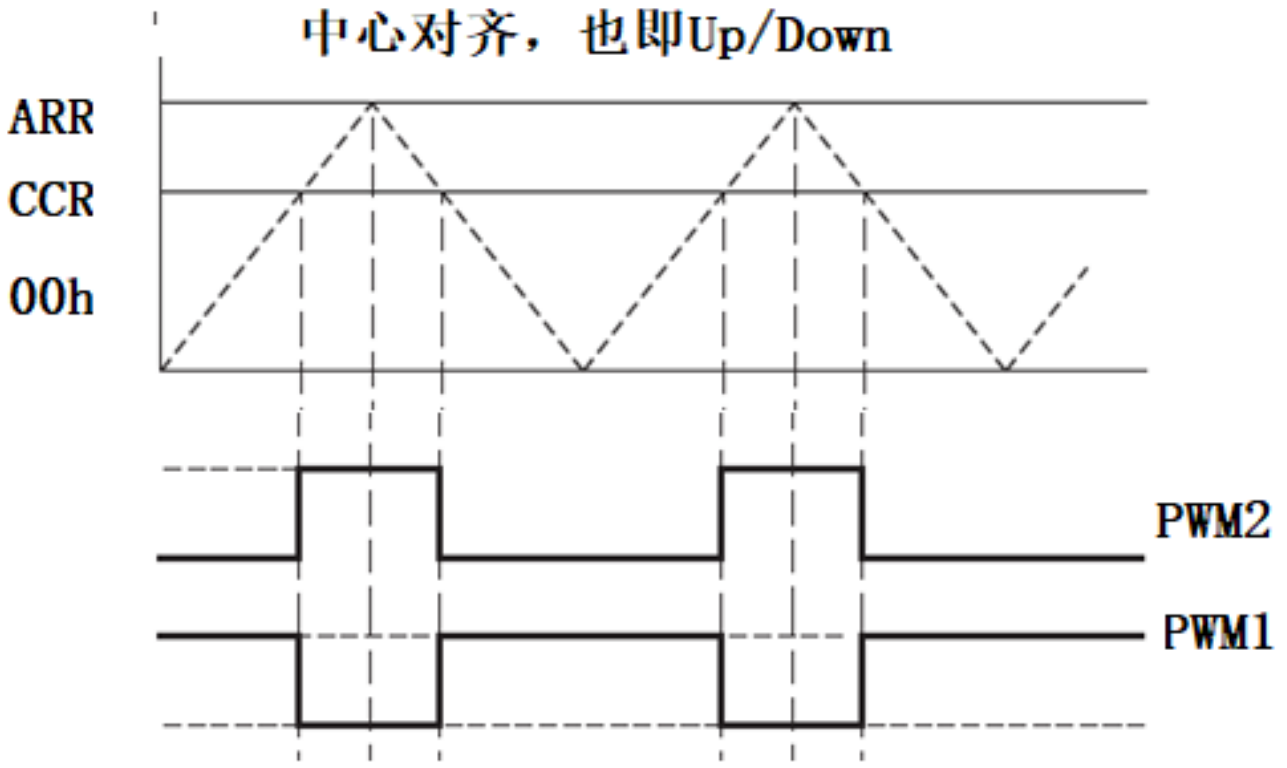


图 32-15 PWM1 模式的中心对齐波形

分为中心对齐模式 1/2/3 区别：比较中断
中断标志位CCxIF 在何时置1：
CNT递减计数时置 1
CNT递增计数时候置 1
CNT递增和递减计数时都置1

PWM1中心对齐模式的波形



目录

CONTENTS.

STM32-10

初始化结构体讲解

1-时基初始化结构体：TIM_TimeBaseInitTypeDef

代码清单 32-1 定时器基本初始化结构体

```
1 typedef struct {
2     uint16_t TIM_Prescaler;          // 预分频器
3     uint16_t TIM_CounterMode;        // 计数模式
4     uint32_t TIM_Period;             // 定时器周期
5     uint16_t TIM_ClockDivision;      // 时钟分频
6     uint8_t TIM_RepetitionCounter;   // 重复计算器
7 } TIM_TimeBaseInitTypeDef;
```

初始化结构体讲解

1-TIM_Prescaler：定时器预分频器设置，时钟源经该预分频器才是定时器计数时钟CK_CNT，它设定 PSC 寄存器的值。计算公式为： 计数器时钟频率 (fCK_CNT) 等于fCK_PSC / (PSC[15:0] + 1)，可实现 1 至 65536 分频。

初始化结构体讲解

2-TIM_CounterMode：定时器计数方式，可设置为向上计数、向下计数以及中心对齐。高级控制定时器允许选择任意一种。

3-TIM_Period：定时器周期，实际就是设定自动重载寄存器 ARR 的值， ARR 为要装载到实际自动重载寄存器（即影子寄存器） 的值， 可设置范围为 0 至 65535。

初始化结构体讲解

4-TIM_ClockDivision：时钟分频，设置定时器时钟 CK_INT 频率与死区发生器以及数字滤波器采样时钟频率分频比。可以选择 1、 2、 4 分频。

5-TIM_RepetitionCounter：重复计数器，只有 8 位，只存在于高级定时器。

初始化结构体讲解

2-输出比较结构体：TIM_OCInitTypeDef

代码清单 32-2 定时器比较输出初始化结构体

```
typedef struct {
    uint16_t TIM_OCMode;         // 比较输出模式
    uint16_t TIM_OutputState;    // 比较输出使能
    uint16_t TIM_OutputNState;   // 比较互补输出使能
    uint32_t TIM_Pulse;          // 脉冲宽度
    uint16_t TIM_OCPolarity;     // 输出极性
    uint16_t TIM_OCNPolarity;    // 互补输出极性
    uint16_t TIM_OCIdleState;    // 空闲状态下比较输出状态
    uint16_t TIM_OCNIdleState;   // 空闲状态下比较互补输出状态
} TIM_OCInitTypeDef;
```

初始化结构体讲解

1-**TIM_OCMode**：比较输出模式选择，总共有八种，常用的为 PWM1/PWM2。它设定CCMRx 寄存器 OCxM[2:0]位的值。

2-TIM_OutputState：比较输出使能，决定最终的输出比较信号 OCx 是否通过外部引脚输出。它设定 TIMx_CCER 寄存器 CCxE 位的值。

初始化结构体讲解

3-TIM_OutputNState:比较互补输出使能，决定 OCx 的互补信号 OCxN 是否通过外部引脚输出。它设定 CCER 寄存器 CCxNE 位的值。

4-TIM_Pulse：比较输出脉冲宽度，实际设定比较寄存器 CCR 的值，决定脉冲宽度。可设置范围为 0 至 65535。

初始化结构体讲解

5-TIM_OCPolarity：比较输出极性，可选 OCx 为高电平有效或低电平有效。它决定着定时器通道有效电平。它设定 CCER 寄存器的 CCxP 位的值。

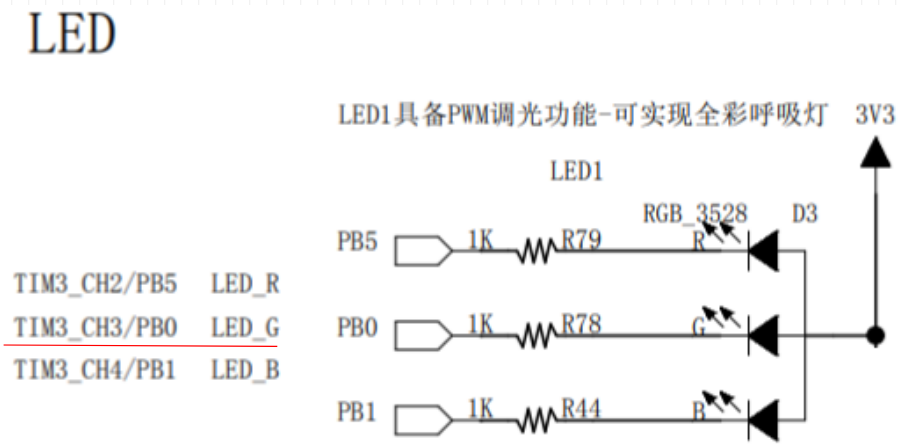
6-TIM_OCNPolarity：比较互补输出极性，可选 OCxN 为高电平有效或低电平有效。它设定 TIMx_CCER 寄存器的 CCxNP 位的值。

初始化结构体讲解

7-TIM_OCIdleState：空闲状态时通道输出电平设置，可选输出 1 或输出 0，即在空闲状态(BDTR_MOE 位为 0)时，经过死区时间后定时器通道输出高电平或低电平。它设定CR2 寄存器的 OISx 位的值。

8-TIM_OCNIdleState：空闲状态时互补通道输出电平设置，可选输出 1 或输出 0，即在空闲状态(BDTR_MOE 位为 0)时，经过死区时间后定时器互补通道输出高电平或低电平，设定值必须与 TIM_OCIdleState 相反。它设定是 CR2 寄存器的 OISxN 位的值。

PWM输出示例



main.c

```c
// TIM一通用定时器-1路PWM输出应用
#include "stm32f10x.h"
#include "bsp_led.h"
#include "bsp_GeneralTim.h"

/**
  * @brief  主函数
  * @param  无
  * @retval 无
  */
int main(void)
{
  /* led 端口配置 */
  LED_GPIO_Config();

  /* 定时器初始化 */
  GENERAL_TIM_Init();

  while(1)
  {
  }
}
```

PWM输出示例

bsp_GeneralTim.h

```c
#ifndef __BSP_GENERALTIME_H
#define __BSP_GENERALTIME_H


#include "stm32f10x.h"


/************通用定时器TIM参数定义，只限TIM2、3、4、5************/
// 当使用不同的定时器的时候，对应的GPIO是不一样的，这点要注意
// 我们这里默认使用TIM3

#define             GENERAL_TIM                   TIM3

#define             GENERAL_TIM_APBxClock_FUN     RCC_APB1PeriphClockCmd
#define             GENERAL_TIM_CLK               RCC_APB1Periph_TIM3
#define             GENERAL_TIM_Period            9
#define             GENERAL_TIM_Prescaler         71

// TIM3 输出比较通道3
#define             GENERAL_TIM_CH3_GPIO_CLK      RCC_APB2Periph_GPIOB
#define             GENERAL_TIM_CH3_PORT          GPIOB
#define             GENERAL_TIM_CH3_PIN           GPIO_Pin_0

/*************************函数声明*********************************/

void GENERAL_TIM_Init(void);


#endif  /* __BSP_GENERALTIME_H */
```

bsp_GeneralTim.c

```c
#include "bsp_GeneralTim.h"

static void GENERAL_TIM_GPIO_Config(void)
{
  GPIO_InitTypeDef GPIO_InitStructure;

  // 输出比较通道3 GPIO 初始化
  RCC_APB2PeriphClockCmd(GENERAL_TIM_CH3_GPIO_CLK, ENABLE);
  GPIO_InitStructure.GPIO_Pin =  GENERAL_TIM_CH3_PIN;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_Init(GENERAL_TIM_CH3_PORT, &GPIO_InitStructure);

}
```

PWM输出示例

```c
/* ----------------   PWM信号 周期和占空比的计算--------------- */
// ARR ：自动重装载寄存器的值
// CLK_cnt：计数器的时钟，等于 Fck_int / (psc+1) = 72M/(psc+1)
// PWM 信号的周期 T = ARR * (1/CLK_cnt) = ARR*(PSC+1) / 72M
// 占空比P=CCR/(ARR+1)

static void GENERAL_TIM_Mode_Config(void)
{
  // 开启定时器时钟,即内部时钟CK_INT=72M
  GENERAL_TIM_APBxClock_FUN(GENERAL_TIM_CLK,ENABLE);

/*--------------------时基结构体初始化-------------------------*/
  // 配置周期，这里配置为100K

  TIM_TimeBaseInitTypeDef  TIM_TimeBaseStructure;
  // 自动重装载寄存器的值，累计TIM_Period+1个频率后产生一个更新或者中断
  TIM_TimeBaseStructure.TIM_Period=GENERAL_TIM_Period;
  // 驱动CNT计数器的时钟 = Fck_int/(psc+1)
  TIM_TimeBaseStructure.TIM_Prescaler= GENERAL_TIM_Prescaler;
  // 时钟分频因子 ，配置死区时间时需要用到
  TIM_TimeBaseStructure.TIM_ClockDivision=TIM_CKD_DIV1;
  // 计数器计数模式，设置为向上计数
  TIM_TimeBaseStructure.TIM_CounterMode=TIM_CounterMode_Up;
  // 重复计数器的值，没用到不用管
  TIM_TimeBaseStructure.TIM_RepetitionCounter=0;
  // 初始化定时器
  TIM_TimeBaseInit(GENERAL_TIM, &TIM_TimeBaseStructure);

  /*--------------------输出比较结构体初始化-------------------*/
  // 占空比配置
  uint16_t CCR3_Val = 3;

  TIM_OCInitTypeDef  TIM_OCInitStructure;
  // 配置为PWM模式1
  TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;
  // 输出使能
  TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
  // 输出通道电平极性配置
  TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;

  // 输出比较通道 3
  TIM_OCInitStructure.TIM_Pulse = CCR3_Val;
  TIM_OC3Init(GENERAL_TIM, &TIM_OCInitStructure);
  TIM_OC3PreloadConfig(GENERAL_TIM, TIM_OCPreload_Enable);

  // 使能计数器
  TIM_Cmd(GENERAL_TIM, ENABLE);
}

void GENERAL_TIM_Init(void)
{
  GENERAL_TIM_GPIO_Config();
  GENERAL_TIM_Mode_Config();
}
```