第1章　TIM—高级定时器

本章参考资料：《STM32F10X-中文参考手册》高级定时器章节。学习本章时，配合参考资料一起阅读，效果会更佳，特别是涉及到寄存器说明的部分。

1.1 高级控制定时器

高级控制定时器(TIM1 和 TIM8)和通用定时器在基本定时器的基础上引入了外部引脚，可以实现输入捕获和输出比较功能。高级控制定时器比通用定时器增加了可编程死区互补输出、重复计数器、带刹车(断路)功能，这些功能都是针对工业电机控制方面。这几个功能在本书不做详细的介绍，主要介绍常用的输入捕获和输出比较功能。

高级控制定时器时基单元包含一个 16 位自动重装载寄存器 ARR，一个 16 位的计数器 CNT，可向上/下计数，一个 16 位可编程预分频器 PSC，预分频器时钟源有多种可选，有内部的时钟、外部时钟。还有一个 8 位的重复计数器 RCR，这样最高可实现 40 位的可编程定时。

STM32F103ZET6 的高级/通用定时器的 IO 分配具体见表 1-1。配套开发板因为 IO 资源紧缺，定时器的 IO 很多已经复用它途，故下表中的 IO 只有部分可用于定时器的实验。

表 1-1 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				

1.2 高级控制定时器功能框图

高级控制定时器功能框图包含了高级控制定时器最核心内容，掌握了功能框图，对高级控制定时器就有一个整体的把握，在编程时思路就非常清晰，见，图中有些寄存器是带影子的，表示其有影子寄存器。

www.firebbs.cn

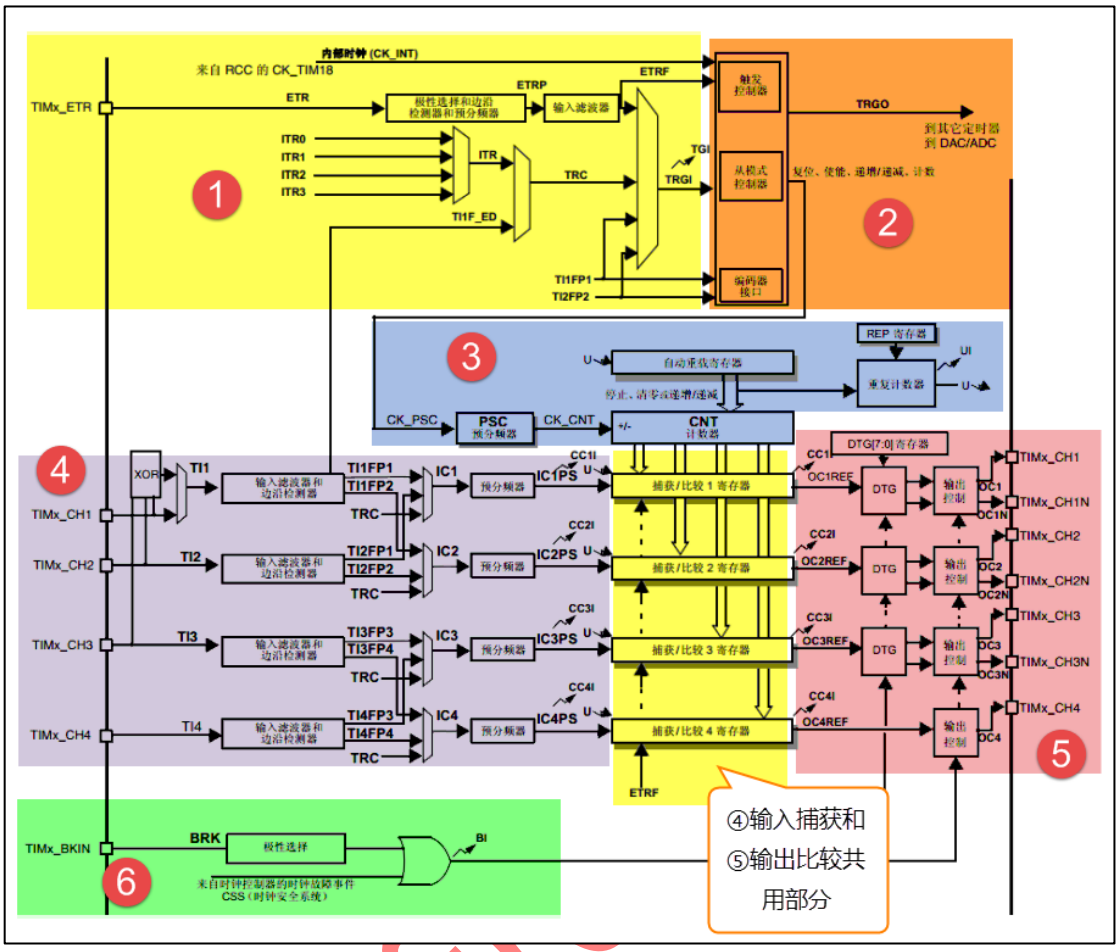


图 1-1 高级控制定时器功能框图

1. ①时钟源

高级控制定时器有四个时钟源可选：

- 内部时钟源 CK_INT
- 外部时钟模式 1：外部输入引脚 TIx（x=1,2,3,4）
- 外部时钟模式 2：外部触发输入 ETR
- 内部触发输入(ITRx)

内部时钟源(CK_INT)

内部时钟 CK_INT 即来自于芯片内部，等于 72M，一般情况下，我们都是使用内部时钟。当从模式控制寄存器 TIMx_SMCR 的 SMS 位等于 000 时，则使用内部时钟。

外部时钟模式 1

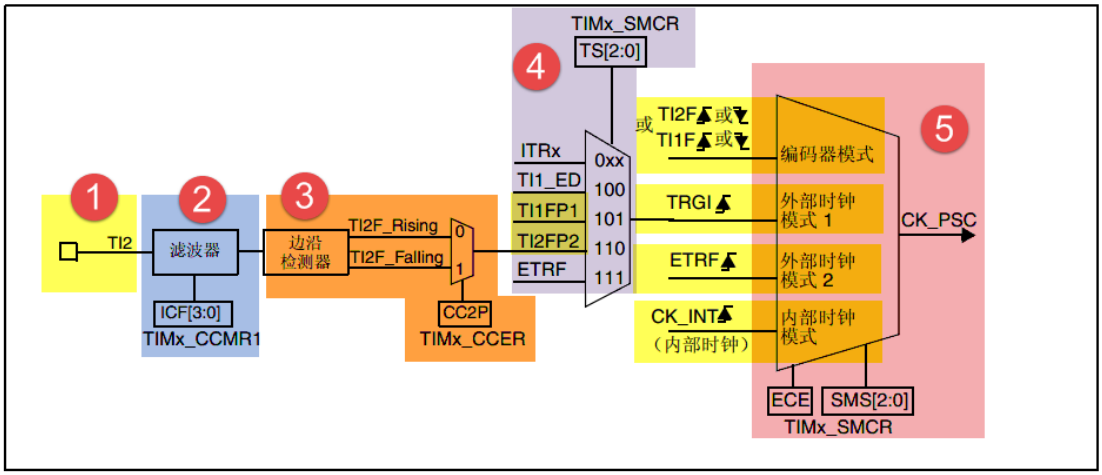


图 1-2 外部时钟模式 1 框图

①：时钟信号输入引脚

当使用外部时钟模式 1 的时候，时钟信号来自于定时器的输入通道，总共有 4 个，分别为 TI1/2/3/4，即 TIMx_CH1/2/3/4。具体使用哪一路信号，由 TIM_CCMRx 的位 CCxS[1:0]配置，其中 CCMR1 控制 TI1/2，CCMR2 控制 TI3/4。

②：滤波器

如果来自外部的时钟信号的频率过高或者混杂有高频干扰信号的话，我们就需要使用滤波器对信号重新采样，来达到降频或者去除高频干扰的目的，具体的由 TIMx_CCMRx 的位 ICxF[3:0]配置。

③：边沿检测

边沿检测的信号来自于滤波器的输出，在成为触发信号之前，需要进行边沿检测，决定是上升沿有效还是下降沿有效，具体的由 TIMx_CCER 的位 CCxP 和 CCxNP 配置。

④：触发选择

当使用外部时钟模式 1 时，触发源有两个，一个是滤波后的定时器输入 1（TI1FP1）和滤波后的定时器输入 2（TI2FP2），具体的由 TIMxSMCR 的位 TS[2:0]配置。

⑤：从模式选择

选定了触发源信号后，最后我们需把信号连接到 TRGI 引脚，让触发信号成为外部时钟模式 1 的输入，最终等于 CK_PSC，然后驱动计数器 CNT 计数。具体的配置 TIMx_SMCR 的位 SMS[2:0]为 000 即可选择外部时钟模式 1。

⑥：使能计数器

经过上面的 5 个步骤之后，最后我们只需使能计数器开始计数，外部时钟模式 1 的配置就算完成。使能计数器由 TIMx_CR1 的位 CEN 配置。

www.firebs.cn

外部时钟模式 2

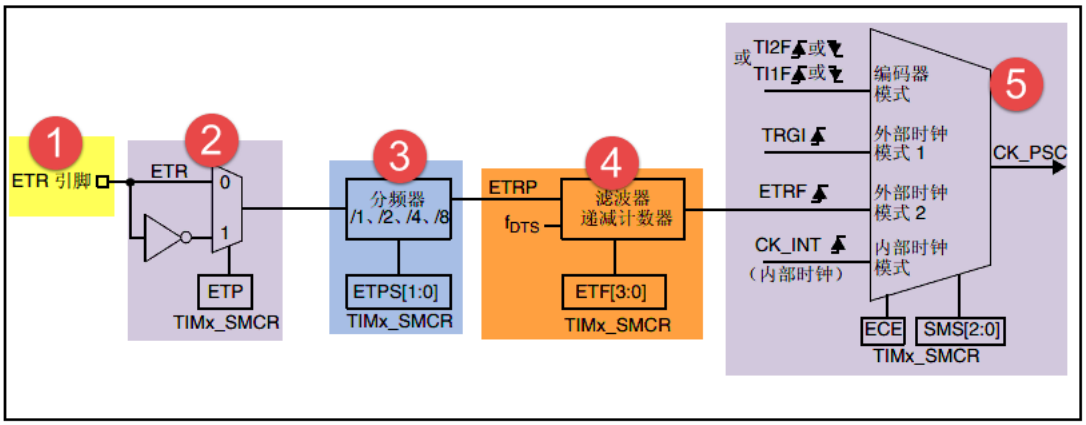


图 1-3 外部时钟模式 2 框图

①：时钟信号输入引脚

当使用外部时钟模式 2 的时候，时钟信号来自于定时器的特定输入通道 TIMx_ETR，只有 1 个。

②：外部触发极性

来自 ETR 引脚输入的信号可以选择为上升沿或者下降沿有效，具体的由 TIMx_SMCR 的位 ETP 配置。

③：外部触发预分频器

由于 ETRP 的信号的频率不能超过 TIMx_CLK（72M）的 1/4，当触发信号的频率很高的情况下，就必须使用分频器来降频，具体的由 TIMx_SMCR 的位 ETPS[1:0]配置。

④：滤波器

如果 ETRP 的信号的频率过高或者混杂有高频干扰信号的话，我们就需要使用滤波器对 ETRP 信号重新采样，来达到降频或者去除高频干扰的目的。具体的由 TIMx_SMCR 的位 ETF[3:0]配置，其中的 f_{DTS} 是由内部时钟 CK_INT 分频得到，具体的由 TIMx_CR1 的位 CKD[1:0]配置。

⑤：从模式选择

经过滤波器滤波的信号连接到 ETRF 引脚后，触发信号成为外部时钟模式 2 的输入，最终等于 CK_PSC，然后驱动计数器 CNT 计数。具体的配置 TIMx_SMCR 的位 ECE 为 1 即可选择外部时钟模式 2。

⑥：使能计数器

经过上面的 5 个步骤之后，最后我们只需使能计数器开始计数，外部时钟模式 2 的配置就算完成。使能计数器由 TIMx_CR1 的位 CEN 配置。

内部触发输入

内部触发输入是使用一个定时器作为另一个定时器的预分频器。硬件上高级控制定时器和通用定时器在内部连接在一起，可以实现定时器同步或级联。主模式的定时器可以对从模式定时器执行复位、启动、停止或提供时钟。

2. ②控制器

高级控制定时器控制器部分包括触发控制器、从模式控制器以及编码器接口。触发控制器用来针对片内外设输出触发信号，比如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/

www.firebbs.cn

递减、计数。有关控制器部分只需熟练阅读寄存器描述即可。

3. ③时基单元

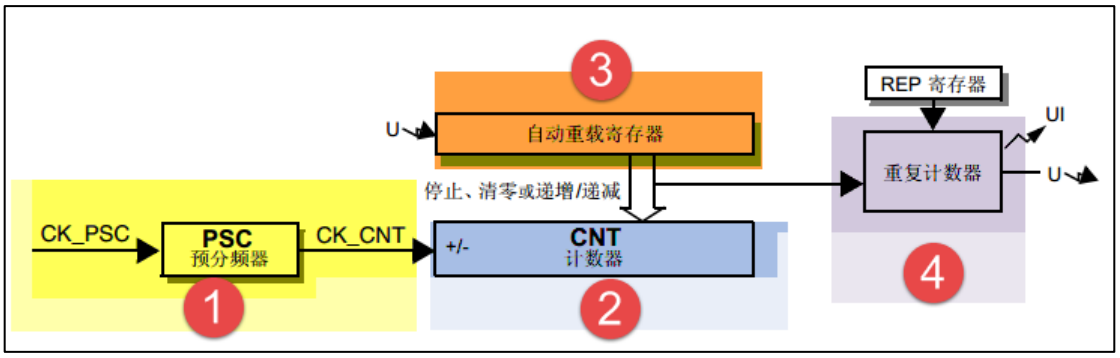


图 1-4 高级定时器时基单元

高级控制定时器时基单元功能包括四个寄存器，分别是计数器寄存器(CNT)、预分频器寄存器(PSC)、自动重载寄存器(ARR)和重复计数器寄存器(RCR)。其中重复计数器 RCR 是高级定时器独有，通用和基本定时器没有。前面三个寄存器都是 16 位有效，TIMx_RCR 寄存器是 8 位有效。

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。

计数器 CNT

高级控制定时器的计数器有三种计数模式，分别为递增计数模式、递减计数模式和递增/递减(中心对齐)计数模式。

(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。

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。

www.firebbs.cn

重复计数器 RCR

在基本/通用定时器发生上/下溢事件时直接就生成更新事件，但对于高级控制定时器却不是这样，高级控制定时器在硬件结构上多出了重复计数器，在定时器发生上溢或下溢事件是递减重复计数器的值，只有当重复计数器为 0 时才会生成更新事件。在发生 N+1 个上溢或下溢事件(N 为 RCR 的值)时产生更新事件。

4. ④输入捕获

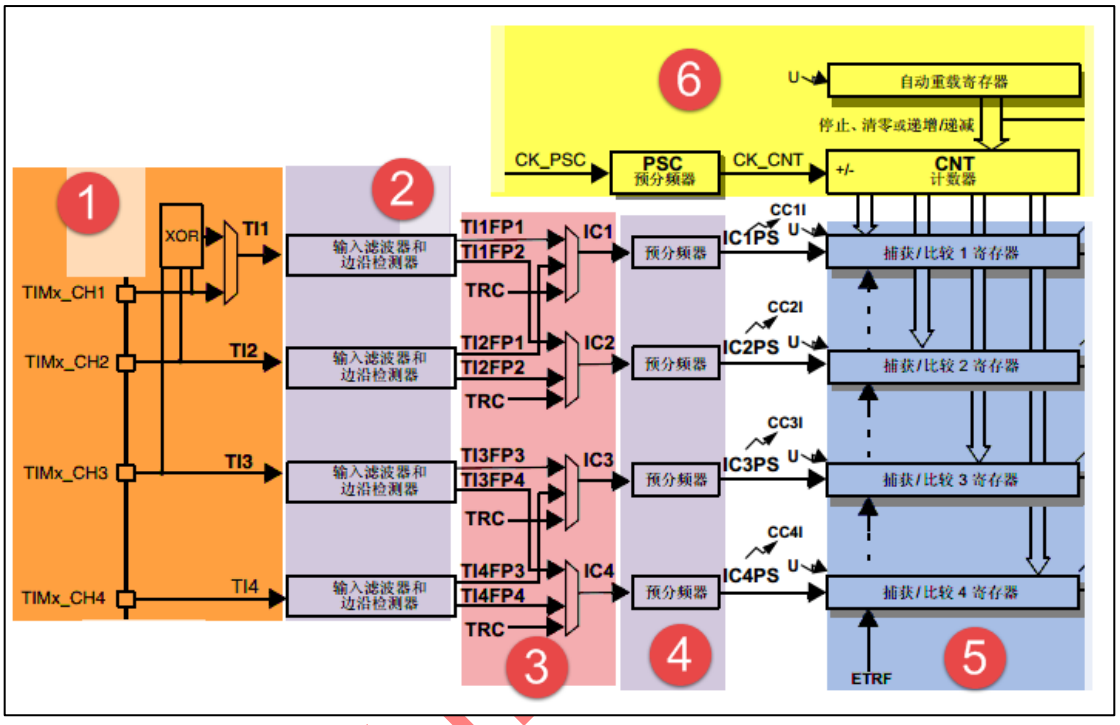


图 1-5 输入捕获功能框图

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。

①输入通道

需要被测量的信号从定时器的外部引脚 TIMx_CH1/2/3/4 进入，通常叫 TI1/2/3/4，在后面的捕获讲解中对于要被测量的信号我们都以 TIx 为标准叫法。

②输入滤波器和边沿检测器

当输入的信号存在高频干扰的时候，我们需要对输入信号进行滤波，即进行重新采样，根据采样定律，采样的频率必须大于等于两倍的输入信号。比如输入的信号为 1M，又存在高频的信号干扰，那么此时就很有必要进行滤波，我们可以设置采样频率为 2M，这样可以在保证采样到有效信号的基础上把高于 2M 的高频干扰信号过滤掉。

滤波器的配置由 CR1 寄存器的位 CKD[1:0]和 CCMR1/2 的位 ICxF[3:0]控制。从 ICxF 位的描述可知，采样频率 f_{SAMPLE} 可以由 f_{CK_INT} 和 f_{DTS} 分频后的时钟提供，其中是 f_{CK_INT} 内部时钟，f_{DTS} 是 f_{CK_INT} 经过分频后得到的频率，分频因子由 CKD[1:0]决定，可以是不分频，2 分频或者是 4 分频。

边沿检测器用来设置信号在捕获的时候是什么边沿有效，可以是上升沿，下降沿，或者是双边沿，具体的由 CCER 寄存器的位 CCxP 和 CCxNP 决定。

③捕获通道

捕获通道就是图中的 IC1/2/3/4，每个捕获通道都有相对应的捕获寄存器 CCR1/2/3/4，当发生捕获的时候，计数器 CNT 的值就会被锁存到捕获寄存器中。

这里我们要搞清楚输入通道和捕获通道的区别，输入通道是用来输入信号的，捕获通道是用来捕获输入信号的通道，一个输入通道的信号可以同时输入给两个捕获通道。比如输入通道 TI1 的信号经过滤波边沿检测器之后的 TI1FP1 和 TI1FP2 可以进入到捕获通道 IC1 和 IC2，其实这就是我们后面要讲的 PWM 输入捕获，只有一路输入信号（TI1）却占用了两个捕获通道（IC1 和 IC2）。当只需要测量输入信号的脉宽时候，用一个捕获通道即可。输入通道和捕获通道的映射关系具体由寄存器 CCMRx 的位 CCxS[1:0]配置。

④预分频器

ICx 的输出信号会经过一个预分频器，用于决定发生多少个事件时进行一次捕获。具体的由寄存器 CCMRx 的位 ICxPSC 配置，如果希望捕获信号的每一个边沿，则不分频。

⑤捕获寄存器

经过预分频器的信号 ICxPS 是最终被捕获的信号，当发生捕获时（第一次），计数器 CNT 的值会被锁存到捕获寄存器 CCR 中，还会产生 CCxI 中断，相应的中断位 CCxIF（在 SR 寄存器中）会被置位，通过软件或者读取 CCR 中的值可以将 CCxIF 清 0。如果发生第二次捕获（即重复捕获：CCR 寄存器中已捕获到计数器值且 CCxIF 标志已置 1），则捕获溢出标志位 CCxOF（在 SR 寄存器中）会被置位，CCxOF 只能通过软件清零。

5. ⑤输出比较

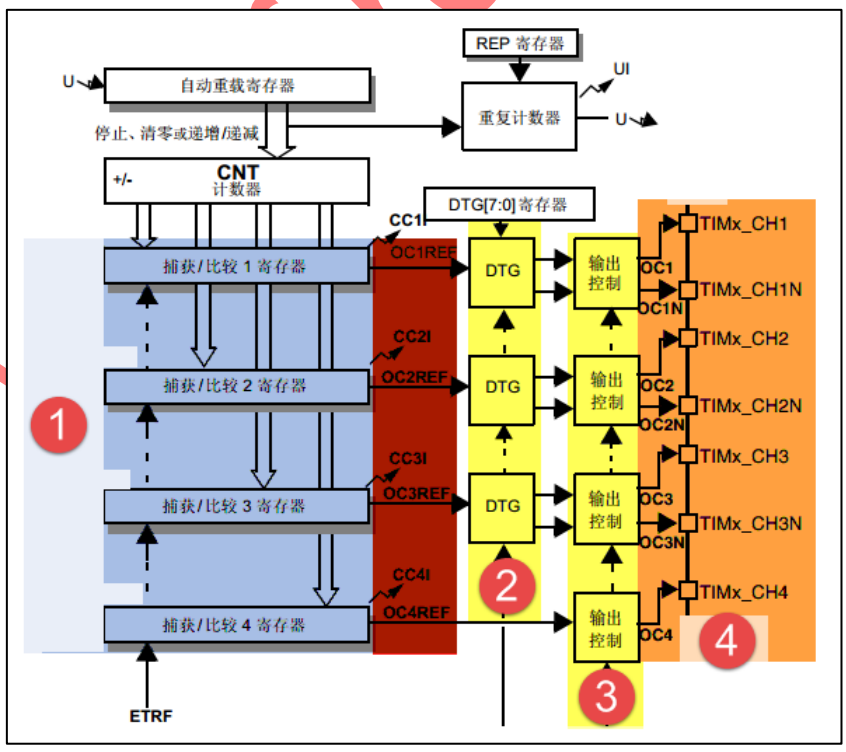


图 1-6 输出比较功能框图

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。

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。

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。

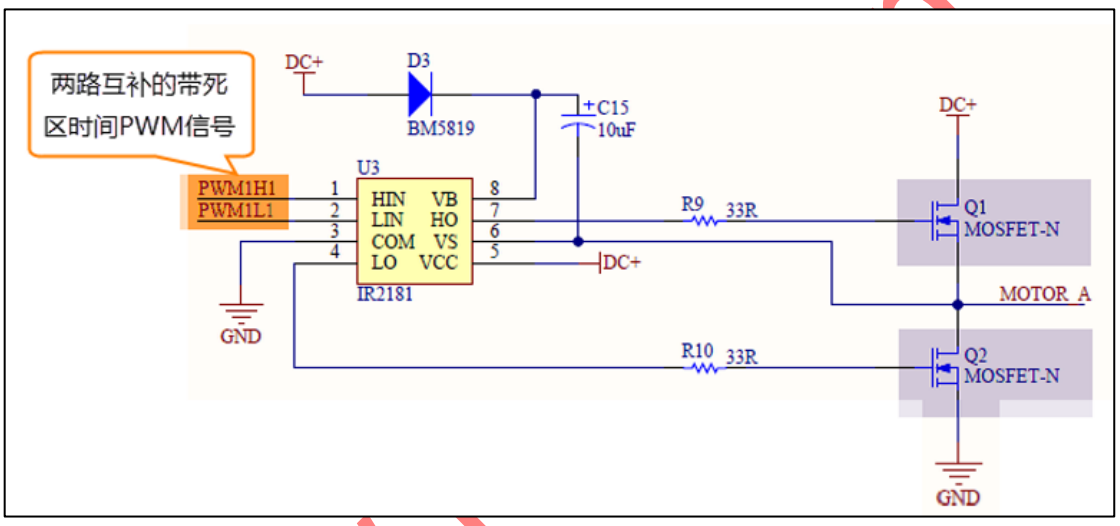


图 1-7 半桥驱动电路

在这个半桥驱动电路中，Q1 导通，Q2 截止，此时我想让 Q1 截止 Q2 导通，肯定是要先让 Q1 截止一段时间之后，再等一段时间才让 Q2 导通，那么这段等待的时间就称为死区时间，因为 Q1 关闭需要时间（由 MOS 管的工艺决定）。如果 Q1 关闭之后，马上打开 Q2，那么此时一段时间内相当于 Q1 和 Q2 都导通了，这样电路会短路。

图 1-8 是针对上面的半桥驱动电路而画的带死区插入的 PWM 信号，图中的死区时间要根据 MOS 管的工艺来调节。

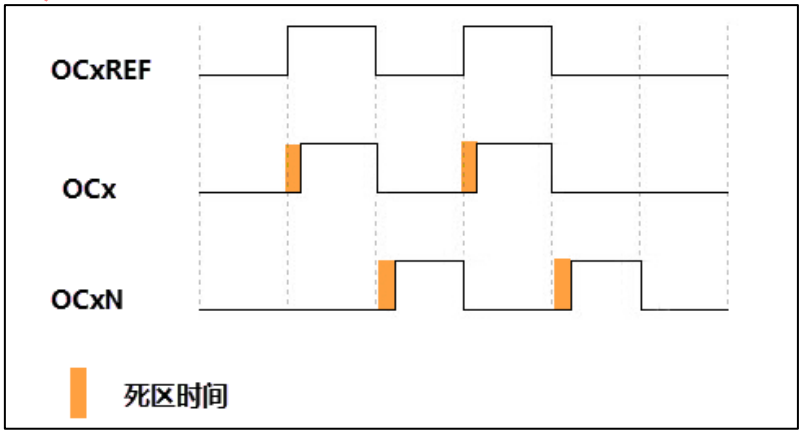


图 1-8 带死区插入的互补输出

③输出控制

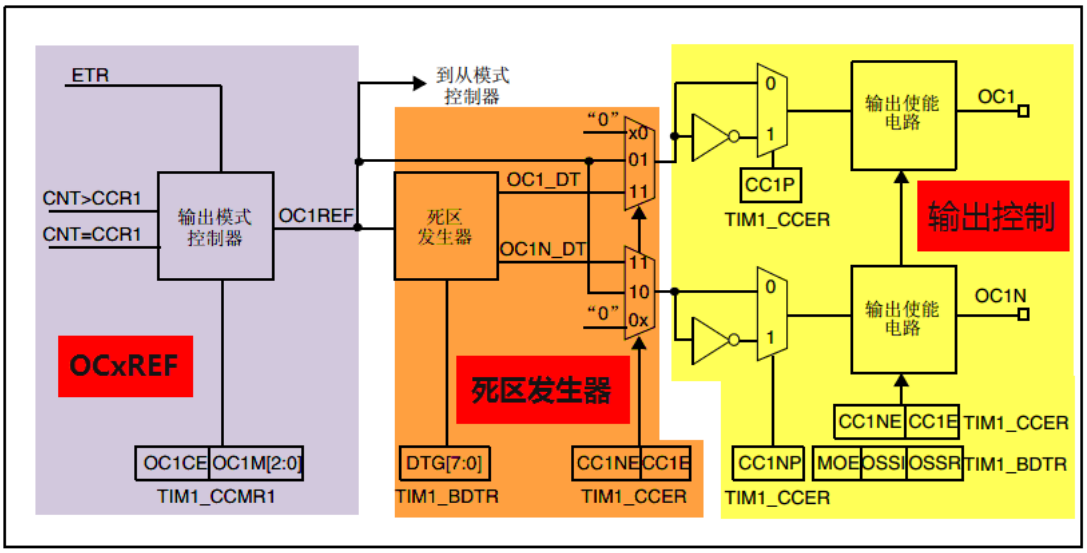


图 1-9 输出比较（通道 1~3）的输出控制框图

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。

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。

6. ⑥断路功能

断路功能就是电机控制的刹车功能，使能断路功能时，根据相关控制位状态修改输出信号电平。在任何情况下，OCx 和 OCxN 输出都不能同时为有效电平，这关系到电机控制常用的 H 桥电路结构原因。

断路源可以是时钟故障事件，由内部复位时钟控制器中的时钟安全系统(CSS)生成，也可以是外部断路输入 IO，两者是或运算关系。

系统复位启动都默认关闭断路功能，将断路和死区寄存器(TIMx_BDTR)的 BKE 为置 1，使能断路功能。可通过 TIMx_BDTR 寄存器的 BKP 位设置设置断路输入引脚的有效电平，设置为 1 时输入 BRK 为高电平有效，否则低电平有效。

发送断路时，将产生以下效果：

- TIMx_BDTR 寄存器中主输出模式使能(MOE)位被清零，输出处于无效、空闲或复位状态；
- 根据相关控制位状态控制输出通道引脚电平；当使能通道互补输出时，会根据情况自动控制输出通道电平；
- 将 TIMx_SR 寄存器中的 BIF 位置 1，并可产生中断和 DMA 传输请求。

www.ireboot.c

- 如果 TIMx_BDTR 寄存器中的 自动输出使能(AOE)位置 1，则 MOE 位会在发生下一个 UEV 事件时自动再次置 1。

1.3 输入捕获应用

输入捕获一般应用在两个方面，一个方面是脉冲跳变沿时间测量，另一方面是 PWM 输入测量。

1.3.1 测量脉宽或者频率

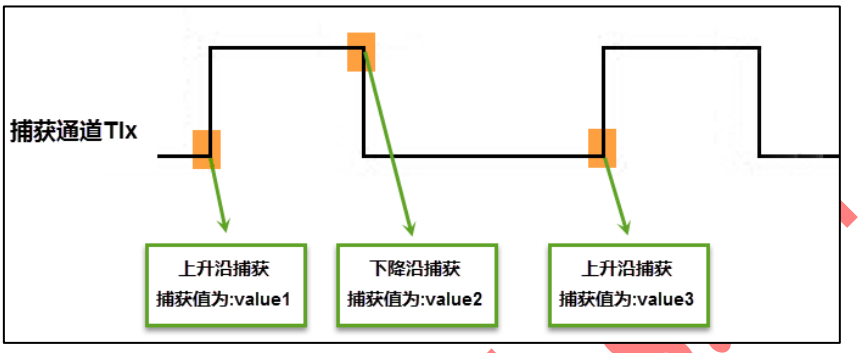


图 1-10 脉宽/频率测量示意图

1. 测量频率

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

2. 测量脉宽

当捕获通道 TIx 上出现上升沿时，发生第一次捕获，计数器 CNT 的值会被锁存到捕获寄存器 CCR 中，而且还会进入捕获中断，在中断服务程序中记录一次捕获（可以用一个标志变量来记录），并把捕获寄存器中的值读取到 value1 中。然后把捕获边沿改变为下降沿捕获，目的是捕获后面的下降沿。当下降沿到来的时候，发生第二次捕获，计数器 CNT 的值会再次被锁存到捕获寄存器 CCR 中，并再次进入捕获中断，在捕获中断中，把捕获寄存器的值读取到 value3 中，并清除捕获记录标志。然后把捕获边沿设置为上升沿捕获。

在测量脉宽过程中需要来回的切换捕获边沿的极性，如果测量的脉宽时间比较长，定时器就会发生溢出，溢出的时候会产生更新中断，我们可以在中断里面对溢出进行记录处理。

1.3.2 PWM 输入模式

测量脉宽和频率还有一个更简便的方法就是使用 PWM 输入模式，该模式是输入捕获的特例，只能使用通道 1 和通道 2，通道 3 和通道 4 使用不了。与上面那种只使用一个捕获寄存器测量脉宽和频率的方法相比，PWM 输入模式需要占用两个捕获寄存器。

www.firebbs.

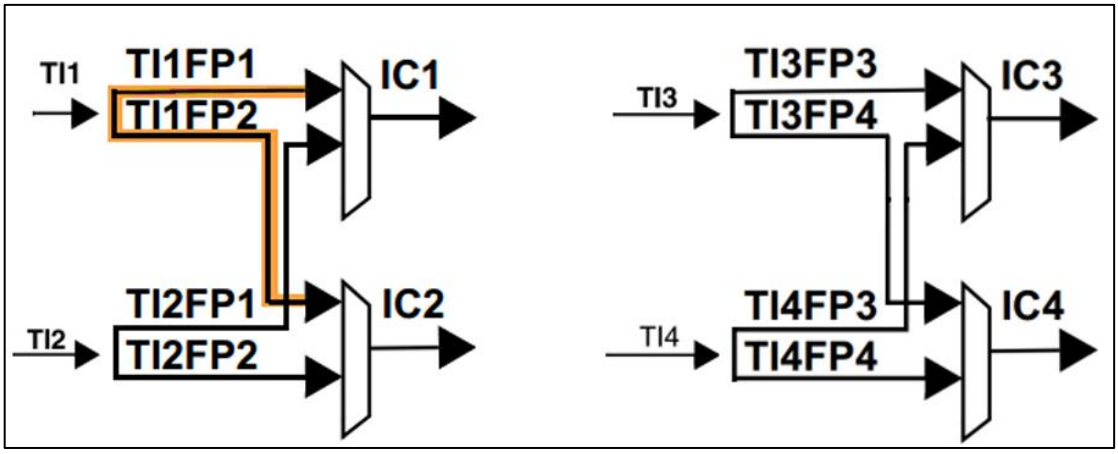


图 1-11 输入通道和捕获通道的关系映射图

当使用 PWM 输入模式的时候，因为一个输入通道(TIx)会占用两个捕获通道(ICx)，所以一个定时器在使用 PWM 输入的时候最多只能使用两个输入通道(TIx)。

我们以输入通道 TI1 工作在 PWM 输入模式为例来讲解下具体的工作原理，其他通道以此类推即可。

PWM 信号由输入通道 TI1 进入，因为是 PWM 输入模式的缘故，信号会被分为两路，一路是 TI1FP1，另外一路是 TI2FP2。其中一路是周期，另一路是占空比，具体哪一路信号对应周期还是占空比，得从程序上设置哪一路信号作为触发输入，作为触发输入的哪一路信号对应的就是周期，另一路就是对应占空比。作为触发输入的那一路信号还需要设置极性，是上升沿还是下降沿捕获，一旦设置好触发输入的极性，另外一路硬件就会自动配置为相反的极性捕获，无需软件配置。一句话概括就是：选定输入通道，确定触发信号，然后设置触发信号的极性即可，因为是 PWM 输入的缘故，另一路信号则由硬件配置，无需软件配置。

当使用 PWM 输入模式的时候必须将从模式控制器配置为复位模式（配置寄存器 SMCR 的位 SMS[2:0]来实现），即当我们启动触发信号开始进行捕获的时候，同时把计数器 CNT 复位清零。

下面我们以一个更加具体的时序图来分析下 PWM 输入模式。

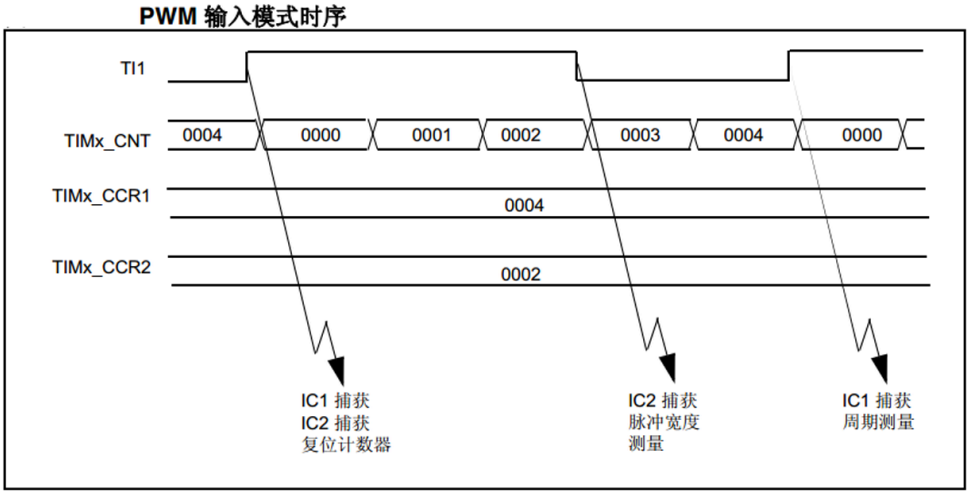


图 1-12 PWM 输入模式时序

PWM 信号由输入通道 TI1 进入，配置 TI1FP1 为触发信号，上升沿捕获。当上升沿的时候 IC1 和 IC2 同时捕获，计数器 CNT 清零，到了下降沿的时候，IC2 捕获，此时计数器 CNT 的值被锁存到捕获寄存器 CCR2 中，到了下一个上升沿的时候，IC1 捕获，计数器 CNT 的值被锁存到捕获寄存器 CCR1 中。其中 CCR2+1 测量的是脉宽，CCR1+1 测量的是周期。这

www.rebbs.cn

里要注意的是 CCR2 和 CCR1 的值在计算占空比和频率的时候都必须加 1，因为计数器是从 0 开始计数的。

从软件上来说，用 PWM 输入模式测量脉宽和周期更容易，付出的代价是需要占用两个捕获寄存器。

1.4 输出比较应用

输出比较模式总共有 8 种，具体的由寄存器 CCMRx 的位 OCxM[2:0]配置。我们这里只讲解最常用的 PWM 模式，其他几种模式具体的看数据手册即可。

1.4.1 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。

PWM 模式分为两种，PWM1 和 PWM2，总得来说是差不多，就看你怎么用而已，具体的区别见表格 1-1。

表格 1-1 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

下面我们以 PWM1 模式来讲解，以计数器 CNT 计数的方向不同还分为边沿对齐模式和中心对齐模式。PWM 信号主要都是用来控制电机，一般的电机控制用的都是边沿对齐模式，FOC 电机一般用中心对齐模式。我们这里只分析这两种模式在信号感官上（即信号波形）的区别，具体在电机控制中的区别不做讨论，到了你真正需要使用的时候就会知道了。

1. PWM 边沿对齐模式

在递增计数模式下，计数器从 0 计数到自动重载值（ TIMx_ARR 寄存器的内容），然后重新从 0 开始计数并生成计数器上溢事件

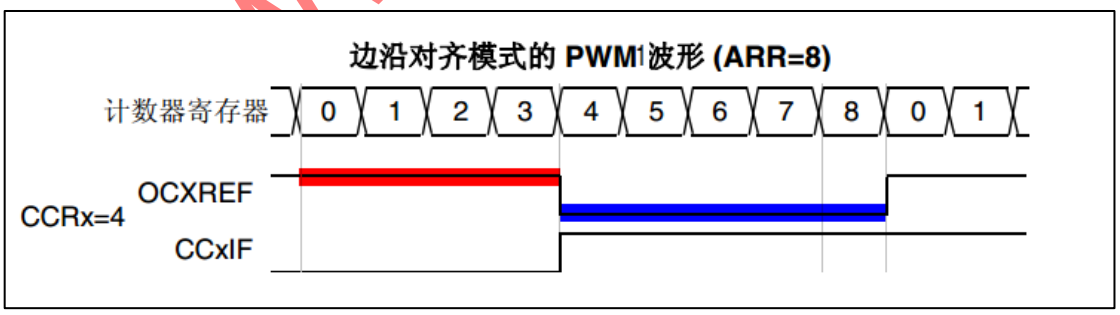


图 1-13 PWM1 模式的边沿对齐波形

在边沿对齐模式下，计数器 CNT 只工作在一种模式，递增或者递减模式。这里我们以 CNT 工作在递增模式为例，在中，ARR=8，CCR=4，CNT 从 0 开始计数，当 CNT<CCR 的值时，OCxREF 为有效的高电平，于此同时，比较中断寄存器 CCxIF 置位。当 CCR=<CNT<=ARR 时，OCxREF 为无效的低电平。然后 CNT 又从 0 开始计数并生成计数器上溢事件，以此循环往复。

www.ireboss.cn

2. PWM 中心对齐模式

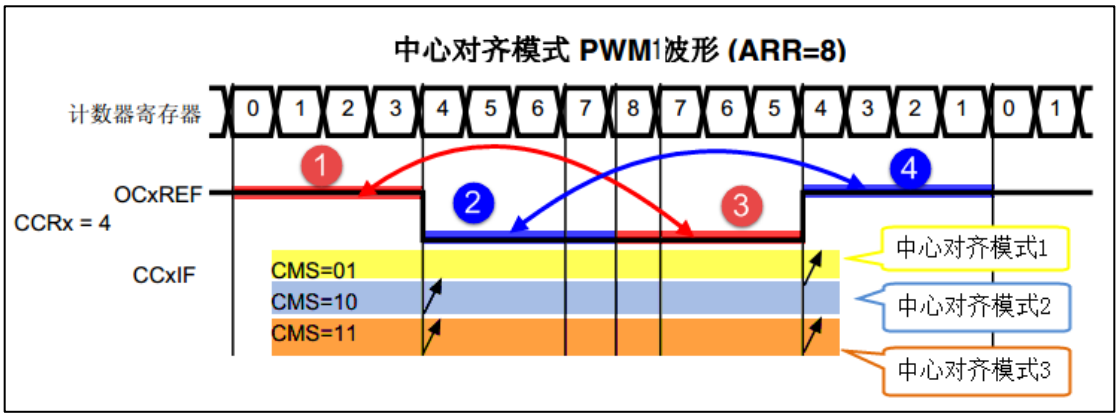


图 1-14 PWM1 模式的中心对齐波形

在中心对齐模式下，计数器 CNT 是工作做递增/递减模式下。开始的时候，计数器 CNT 从 0 开始计数到自动重载值减 1(ARR-1)，生成计数器上溢事件；然后从自动重载值开始向下计数到 1 并生成计数器下溢事件。之后从 0 开始重新计数。

图 1-14 是 PWM1 模式的中心对齐波形，ARR=8，CCR=4。第一阶段计数器 CNT 工作在递增模式下，从 0 开始计数，当 CNT<CCR 的值时，OCxREF 为有效的高电平，当 CCR=<CNT<<ARR 时，OCxREF 为无效的低电平。第二阶段计数器 CNT 工作在递减模式，从 ARR 的值开始递减，当 CNT>CCR 时，OCxREF 为无效的低电平，当 CCR=>CNT>=1 时，OCxREF 为有效的高电平。

在波形图上我们把波形分为两个阶段，第一个阶段是计数器 CNT 工作在递增模式的波形，这个阶段我们又分为①和②两个阶段，第二个阶段是计数器 CNT 工作在递减模式的波形，这个阶段我们又分为③和④两个阶段。要说中心对齐模式下的波形有什么特征的话，那就是①和③阶段的时间相等，②和④阶段的时间相等。

中心对齐模式又分为中心对齐模式 1/2/3 三种，具体由寄存器 CR1 位 CMS[1:0]配置。具体的区别就是比较中断中断标志位 CCxIF 在何时置 1：中心模式 1 在 CNT 递减计数的时候置 1，中心对齐模式 2 在 CNT 递增计数时置 1，中心模式 3 在 CNT 递增和递减计数时都置 1。

1.5 定时器初始化结构体详解

在标准库函数头文件 stm32f4xx_tim.h 中对定时器外设建立了四个初始化结构体，分别为时基初始化结构体 TIM_TimeBaseInitTypeDef、输出比较初始化结构体 TIM_OCInitTypeDef、输入捕获初始化结构体 TIM_ICInitTypeDef 和断路和死区初始化结构体 TIM_BDTRInitTypeDef，高级控制定时器可以用到所有初始化结构体，通用定时器不能使用 TIM_BDTRInitTypeDef 结构体，基本定时器只能使用时基结构体。接下来我们具体讲解下这四个结构体。

1. TIM_TimeBaseInitTypeDef

时基结构体 TIM_TimeBaseInitTypeDef 用于定时器基础参数设置，与 TIM_TimeBaseInit 函数配合使用完成配置。

代码清单 1-1 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
```

www.ireobs.cn

```
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;
```

(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 (f_{CK_CNT}) 等于 f_{CK_PSC} / (PSC[15:0] + 1)，可实现 1 至 65536 分频。

(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。

(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。

2. TIM_OCInitTypeDef

输出比较结构体 TIM_OCInitTypeDef 用于输出比较模式，与 TIM_OCxInit 函数配合使用完成指定定时器输出通道初始化配置。高级控制定时器有四个定时器通道，使用时都必须单独设置。

代码清单 1-2 定时器比较输出初始化结构体

```
1  typedef struct {
2     uint16_t TIM_OCMode;        // 比较输出模式
3     uint16_t TIM_OutputState;    // 比较输出使能
4     uint16_t TIM_OutputNState;   // 比较互补输出使能
5     uint32_t TIM_Pulse;         // 脉冲宽度
6     uint16_t TIM_OCPolarity;     // 输出极性
7     uint16_t TIM_OCNPolarity;    // 互补输出极性
8     uint16_t TIM_OCIdleState;    // 空闲状态下比较输出状态
9     uint16_t TIM_OCNIdleState;   // 空闲状态下比较互补输出状态
10 } TIM_OCInitTypeDef;
```

(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/CCxNE 位的值。

(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。

(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。

(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。

www.xxx.com

(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。

3. TIM_ICInitTypeDef

输入捕获结构体 TIM_ICInitTypeDef 用于输入捕获模式，与 TIM_ICInit 函数配合使用完成定时器输入通道初始化配置。如果使用 PWM 输入模式需要与 TIM_PWMIConfig 函数配合使用完成定时器输入通道初始化配置。

代码清单 1-3 定时器输入捕获初始化结构体

```
typedef struct {
   uint16_t TIM_Channel;      // 输入通道选择
   uint16_t TIM_ICPolarity;   // 输入捕获触发选择
   uint16_t TIM_ICSelection;  // 输入捕获选择
   uint16_t TIM_ICPrescaler;  // 输入捕获预分频器
   uint16_t TIM_ICFilter;     // 输入捕获滤波器
} TIM_ICInitTypeDef;
```

(1) TIM_Channel：捕获通道 ICx 选择，可选 TIM_Channel_1、TIM_Channel_2、TIM_Channel_3 或 TIM_Channel_4 四个通道。它设定 CCMRx 寄存器 CCxS 位 的值。

(2) TIM_ICPolarity：输入捕获边沿触发选择，可选上升沿触发、下降沿触发或边沿跳变触发。它设定 CCER 寄存器 CCxP 位和 CCxNP 位的值。

(3) TIM_ICSelection：输入通道选择，捕获通道 ICx 的信号可来自三个输入通道，分别为 TIM_ICSelection_DirectTI、TIM_ICSelection_IndirectTI 或 TIM_ICSelection_TRC，具体的区别见图 1-15。如果是普通的输入捕获，4 个通道都可以使用，如果是 PWM 输入则只能使用通道 1 和通道 2。它设定 CCRMx 寄存器的 CCxS[1:0]位的值。

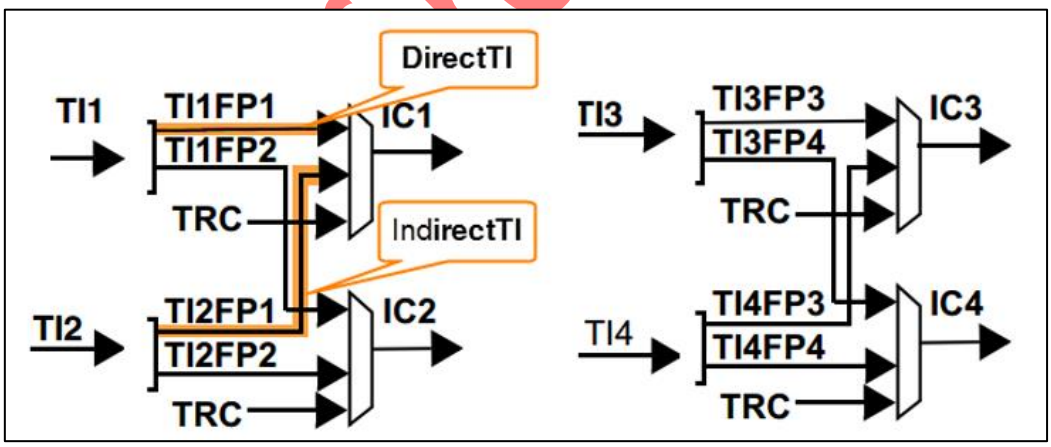


图 1-15 输入通道与捕获通道 IC 的映射图

(4) TIM_ICPrescaler：输入捕获通道预分频器，可设置 1、2、4、8 分频，它设定 CCMRx 寄存器的 ICxPSC[1:0]位的值。如果需要捕获输入信号的每个有效边沿，则设置 1 分频即可。

(5) TIM_ICFilter：输入捕获滤波器设置，可选设置 0x0 至 0x0F。它设定 CCMRx 寄存器 ICxF[3:0]位的值。一般我们不使用滤波器，即设置为 0。

4. TIM_BDTRInitTypeDef

断路和死区结构体 TIM_BDTRInitTypeDef 用于断路和死区参数的设置，属于高级定时器专用，用于配置断路时通道输出状态，以及死区时间。它与 TIM_BDTRConfig 函数配置

使用完成参数配置。这个结构体的成员只对应 BDTR 这个寄存器，有关成员的具体使用配置请参考手册 BDTR 寄存器的详细描述。

代码清单 1-4 断路和死区初始化结构体

```
typedef struct {
   uint16_t TIM_OSSRState;        // 运行模式下的关闭状态选择
   uint16_t TIM_OSSIState;        // 空闲模式下的关闭状态选择
   uint16_t TIM_LOCKLevel;        // 锁定配置
   uint16_t TIM_DeadTime;         // 死区时间
   uint16_t TIM_Break;           // 断路输入使能控制
   uint16_t TIM_BreakPolarity;    // 断路输入极性
   uint16_t TIM_AutomaticOutput;  // 自动输出使能
} TIM_BDTRInitTypeDef;
```

(1) TIM_OSSRState：运行模式下的关闭状态选择，它设定 BDTR 寄存器 OSSR 位的值。
(2) TIM_OSSIState：空闲模式下的关闭状态选择，它设定 BDTR 寄存器 OSSI 位的值。
(3) TIM_LOCKLevel：锁定级别配置， BDTR 寄存器 LOCK[1:0]位的值。
(4) TIM_DeadTime：配置死区发生器，定义死区持续时间，可选设置范围为 0x0 至 0xFF。它设定 BDTR 寄存器 DTG[7:0]位的值。
(5) TIM_Break：断路输入功能选择，可选使能或禁止。它设定 BDTR 寄存器 BKE 位的值。
(6) TIM_BreakPolarity：断路输入通道 BRK 极性选择，可选高电平有效或低电平有效。它设定 BDTR 寄存器 BKP 位的值。
(7) TIM_AutomaticOutput：自动输出使能，可选使能或禁止，它设定 BDTR 寄存器 AOE 位的值。

1.6 PWM 互补输出实验

输出比较模式比较多，这里我们以 PWM 输出为例讲解，并通过示波器来观察波形。实验中不仅在主输出通道输出波形，还在互补通道输出与主通道互补的的波形，并且添加了断路和死区功能。

1.6.1 硬件设计

根据开发板引脚使用情况，并且参考表 1-1 中定时器引脚信息 ，使用高级定时器 TIM1 的通道 1 及其互补通道作为本实验的波形输出通道，对应选择 PA8 和 PB13 引脚。将示波器的两个输入通道分别与 PA8 和 PB13 引脚连接，用于观察波形，还有注意共地。

为增加断路功能，需要用到 TIM1_BKIN 引脚，这里选择 PB12 引脚。程序我们设置该引脚为高电平有效，当 BKIN 引脚被置高低电平的时候，两路互补的 PWM 输出就被停止，就好像是刹车一样。

1.6.2 软件设计

这里只讲解核心的部分代码，有些变量的设置，头文件的包含等并没有涉及到，完整的代码请参考本章配套的工程。我们创建了两个文件：bsp_AdvanceTim.c 和 bsp_AdvanceTim.h 文件用来存定时器驱动程序及相关宏定义。

1. 编程要点

(1) 定时器用到的 GP IO 初始化
(2) 定时器时基结构体 TIM_TimeBaseInitTypeDef 初始化
(3) 定时器输出比较结构体 TIM_OCInitTypeDef 初始化

www.firebbs.cn

(4) 定时器刹车和死区结构体 TIM_BDTRInitTypeDef 初始化

2. 软件分析

宏定义

代码清单 1-5 宏定义

```
/************高级定时器 TIM 参数定义，只限 TIM1 和 TIM8************/
// 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
// 这里我们使用高级控制定时器 TIM1

#define           ADVANCE_TIM                 TIM1
#define           ADVANCE_TIM_APBxClock_FUN    RCC_APB2PeriphClockCmd
#define           ADVANCE_TIM_CLK             RCC_APB2Periph_TIM1
// PWM 信号的频率 F = TIM_CLK/{(ARR+1)*(PSC+1)}
#define           ADVANCE_TIM_PERIOD           (8-1)
#define           ADVANCE_TIM_PSC              (9-1)
#define           ADVANCE_TIM_PULSE            4

#define           ADVANCE_TIM_IRQ             TIM1_UP_IRQn
#define           ADVANCE_TIM_IRQHandler       TIM1_UP_IRQHandler

// TIM1 输出比较通道
#define           ADVANCE_TIM_CH1_GPIO_CLK     RCC_APB2Periph_GPIOA
#define           ADVANCE_TIM_CH1_PORT         GPIOA
#define           ADVANCE_TIM_CH1_PIN          GPIO_Pin_8

// TIM1 输出比较通道的互补通道
#define           ADVANCE_TIM_CH1N_GPIO_CLK     RCC_APB2Periph_GPIOB
#define           ADVANCE_TIM_CH1N_PORT         GPIOB
#define           ADVANCE_TIM_CH1N_PIN          GPIO_Pin_13

// TIM1 输出比较通道的刹车通道
#define           ADVANCE_TIM_BKIN_GPIO_CLK     RCC_APB2Periph_GPIOB
#define           ADVANCE_TIM_BKIN_PORT         GPIOB
#define           ADVANCE_TIM_BKIN_PIN          GPIO_Pin_12
```

使用宏定义非常方便程序升级、移植。有关每个宏的具体含义看程序注释即可。

定时器复用功能引脚初始化

代码清单 1-6 定时器复用功能引脚初始化

```
static void ADVANCE_TIM_GPIO_Config(void)
{
   GPIO_InitTypeDef GPIO_InitStructure;

   // 输出比较通道 GPIO 初始化
   RCC_APB2PeriphClockCmd(ADVANCE_TIM_CH1_GPIO_CLK, ENABLE);
   GPIO_InitStructure.GPIO_Pin =  ADVANCE_TIM_CH1_PIN;
   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
   GPIO_Init(ADVANCE_TIM_CH1_PORT, &GPIO_InitStructure);

   // 输出比较通道互补通道 GPIO 初始化
   RCC_APB2PeriphClockCmd(ADVANCE_TIM_CH1N_GPIO_CLK, ENABLE);
   GPIO_InitStructure.GPIO_Pin =  ADVANCE_TIM_CH1N_PIN;
   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
   GPIO_Init(ADVANCE_TIM_CH1N_PORT, &GPIO_InitStructure);

   // 输出比较通道刹车通道 GPIO 初始化
   RCC_APB2PeriphClockCmd(ADVANCE_TIM_BKIN_GPIO_CLK, ENABLE);
   GPIO_InitStructure.GPIO_Pin =  ADVANCE_TIM_BKIN_PIN;
   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
   GPIO_Init(ADVANCE_TIM_BKIN_PORT, &GPIO_InitStructure);
```

```
25    // BKIN 引脚默认先输出低电平
26    GPIO_ResetBits(ADVANCE_TIM_BKIN_PORT,ADVANCE_TIM_BKIN_PIN);
27 }
```

ADVANCE_TIM_GPIO_Config()函数初始化了定时器用到的相关的 GPIO，当使用不同的 GPIO 的时候，只需要修改头文件里面的宏定义即可，而不需要修改这个函数。

定时器模式配置

代码清单 1-7 定时器模式配置

```
1  static void ADVANCE_TIM_Mode_Config(void)
2  {
3     // 开启定时器时钟,即内部时钟 CK_INT=72M
4     ADVANCE_TIM_APBxClock_FUN(ADVANCE_TIM_CLK,ENABLE);
5
6     /*----------------------时基结构体初始化--------------------------*/
7     TIM_TimeBaseInitTypeDef  TIM_TimeBaseStructure;
8     // 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
9     TIM_TimeBaseStructure.TIM_Period=ADVANCE_TIM_PERIOD;
10    // 驱动 CNT 计数器的时钟 = Fck_int/(psc+1)
11    TIM_TimeBaseStructure.TIM_Prescaler= ADVANCE_TIM_PSC;
12    // 时钟分频因子 ，配置死区时间时需要用到
13    TIM_TimeBaseStructure.TIM_ClockDivision=TIM_CKD_DIV1;
14    // 计数器计数模式，设置为向上计数
15    TIM_TimeBaseStructure.TIM_CounterMode=TIM_CounterMode_Up;
16    // 重复计数器的值，没用到不用管
17    TIM_TimeBaseStructure.TIM_RepetitionCounter=0;
18    // 初始化定时器
19    TIM_TimeBaseInit(ADVANCE_TIM, &TIM_TimeBaseStructure);
20
21    /*----------------------输出比较结构体初始化-------------------*/
22    TIM_OCInitTypeDef  TIM_OCInitStructure;
23    // 配置为 PWM 模式 1
24    TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;
25    // 输出使能
26    TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
27    // 互补输出使能
28    TIM_OCInitStructure.TIM_OutputNState = TIM_OutputNState_Enable;
29    // 设置占空比大小
30    TIM_OCInitStructure.TIM_Pulse = ADVANCE_TIM_PULSE;
31    // 输出通道电平极性配置
32    TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;
33    // 互补输出通道电平极性配置
34    TIM_OCInitStructure.TIM_OCNPolarity = TIM_OCNPolarity_High;
35    // 输出通道空闲电平极性配置
36    TIM_OCInitStructure.TIM_OCIdleState = TIM_OCIdleState_Set;
37    // 互补输出通道空闲电平极性配置
38    TIM_OCInitStructure.TIM_OCNIdleState = TIM_OCNIdleState_Reset;
39    TIM_OC1Init(ADVANCE_TIM, &TIM_OCInitStructure);
40    TIM_OC1PreloadConfig(ADVANCE_TIM, TIM_OCPreload_Enable);
41
42    /*--------------------刹车和死区结构体初始化-------------------*/
43    // 有关刹车和死区结构体的成员具体可参考 BDTR 寄存器的描述
44    TIM_BDTRInitTypeDef TIM_BDTRInitStructure;
45    TIM_BDTRInitStructure.TIM_OSSRState = TIM_OSSRState_Enable;
46    TIM_BDTRInitStructure.TIM_OSSIState = TIM_OSSIState_Enable;
47    TIM_BDTRInitStructure.TIM_LOCKLevel = TIM_LOCKLevel_1;
48    // 输出比较信号死区时间配置，具体如何计算可参考 BDTR:UTG[7:0]的描述
49    // 这里配置的死区时间为 152ns
50    TIM_BDTRInitStructure.TIM_DeadTime = 11;
51    TIM_BDTRInitStructure.TIM_Break = TIM_Break_Enable;
52    // 当 BKIN 引脚检测到高电平的时候，输出比较信号被禁止，就好像是刹车一样
53    TIM_BDTRInitStructure.TIM_BreakPolarity = TIM_BreakPolarity_High;
54    TIM_BDTRInitStructure.TIM_AutomaticOutput = TIM_AutomaticOutput_Enable;
```

```
55      TIM_BDTRConfig(ADVANCE_TIM, &TIM_BDTRInitStructure);
56
57      // 使能计数器
58      TIM_Cmd(ADVANCE_TIM, ENABLE);
59      // 主输出使能，当使用的是通用定时器时，这句不需要
60      TIM_CtrlPWMOutputs(ADVANCE_TIM, ENABLE);
61  }
```

ADVANCE_TIM_Mode_Config()函数中初始化了三个结构体，有关这三个结构体成员的具体含义可参考“定时器初始化结构体详解”小节，剩下的程序参考注释阅读即可。如果需要修改 PWM 的周期和占空比，修改头文件里面的 ADVANCE_TIM_PERIOD、ADVANCE_TIM_PSC 和 ADVANCE_TIM_PULSE 这三个宏即可。PWM 信号的频率的计算公司为：F = TIM_CLK/{(ARR+1)*(PSC+1)}，其中 TIM_CLK 等于 72MHZ，ARR 即自动重装载寄存器的值，对应 ADVANCE_TIM_PERIOD 这个宏，PSC 即计数器时钟的分频因子，对应 ADVANCE_TIM_PSC 这个宏。

主函数

代码清单 1-8 main 函数

```
1  int main(void)
2  {
3      /* 高级定时器初始化 */
4      ADVANCE_TIM_Init();
5
6      while (1) {
7      }
8  }
```

Main 函数很简单，调用了 ADVANCE_TIM_Init()函数，该函数调用了 ADVANCE_TIM_GPIO_Config()和 ADVANCE_TIM_Mode_Config()这两个函数完成了定时器 GPIO 引脚和工作模式的初始化，这时，相应的 GPIO 引脚上就可以检测到互补输出的 PWM 信号，而且带死区时间，如果程序运行的过程中，BKIN 引脚被拉高的话，PWM 输出会被禁止，就好像是断路或者刹车一样。

1.6.3 下载验证

根据实验的硬件设计内容接好示波器输入通道和开发板引脚连接，，编译实验程序并下载到开发板上，调整示波器到合适参数，在示波器显示屏和看到一路互补的带死区时间的 PWM 波形，参考图 1-16。至于图中的信号有毛刺是因为信号的输出引脚还接了其他的芯片，受到了影响。



图 1-16 PWM 互补带死区时间波形输出

当 BKIN 引脚接高电平时，PWM 输出被禁止，就好像是刹车一样，具体见图 1-17。

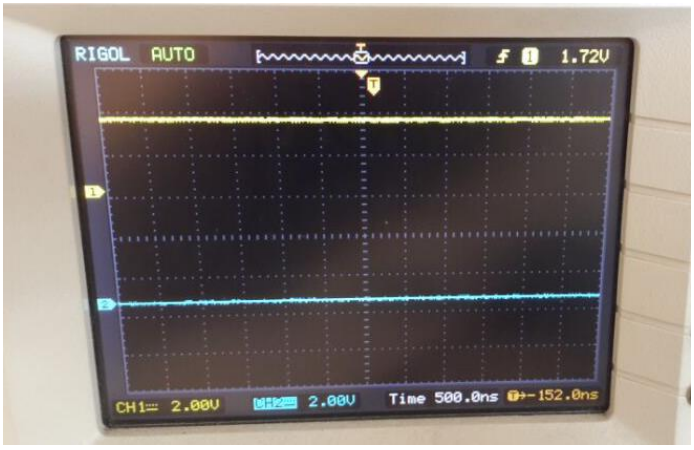


图 1-17 PWM 刹车输出

1.7 脉宽测量输入捕获实验

上一节我们讲了输出比较，这一节我们讲输入捕获。输入捕获有常见的测量脉宽和特殊的 PWM 输入，这节我们先讲测量一个信号的脉宽，下一节再讲 PWM 输入。

1.7.1 硬件设计

根据开发板引脚使用情况，我们选用通用定时器 TIM5 的 CH1，就 PA0 这个 GPIO 来测量信号的脉宽。在开发板中 PA0 接的是一个按键，默认接 GND，当按键按下的时候 IO 口会被拉高，这个时候我们可以利用定时器的输入捕获功能来测量按键按下的这段高电平的时间，按键的具体原理图见图 1-18。

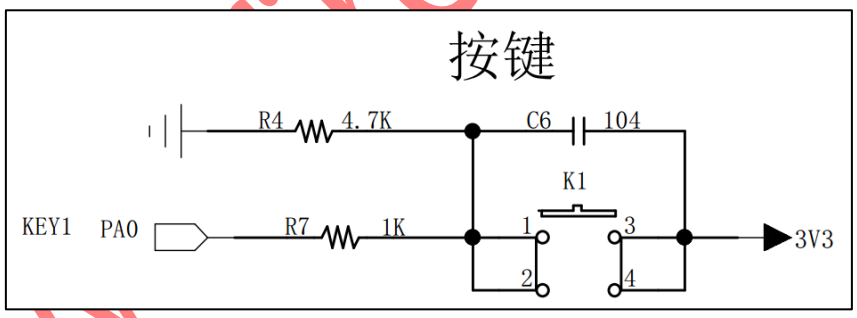


图 1-18 按键原理图。

1.7.2 软件设计

这里只讲解核心的部分代码，有些变量的设置，头文件的包含等并没有涉及到，完整的代码请参考本章配套的工程。我们创建了两个文件：bsp_GeneralTim.c 和 bsp_GeneralTim.h 文件用来存定时器驱动程序及相关宏定义。

1. 编程要点

(1) 定时器用到的 GP IO 初始化
(2) 定时器时基结构体 TIM_TimeBaseInitTypeDef 初始化
(3) 定时器输入捕获结构体 TIM_ICInitTypeDef 初始化
(4) 编写中断服务函数，读取捕获值，计算出脉宽的时间

2. 软件分析

宏定义

代码清单 1-9 宏定义

```
/************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 TIM5

#define           GENERAL_TIM                   TIM5
#define           GENERAL_TIM_APBxClock_FUN     RCC_APB1PeriphClockCmd
#define           GENERAL_TIM_CLK               RCC_APB1Periph_TIM5
#define           GENERAL_TIM_Period            0XFFFF
#define           GENERAL_TIM_Prescaler         (72-1)

// TIM 输入捕获通道 GPIO 相关宏定义
#define           GENERAL_TIM_CH1_GPIO_CLK      RCC_APB2Periph_GPIOA
#define           GENERAL_TIM_CH1_PORT          GPIOA
#define           GENERAL_TIM_CH1_PIN           GPIO_Pin_0
#define           GENERAL_TIM_CHANNEL_x         TIM_Channel_1

// 中断相关宏定义
#define           GENERAL_TIM_IT_CCx            TIM_IT_CC1
#define           GENERAL_TIM_IRQ               TIM5_IRQn
#define           GENERAL_TIM_INT_FUN           TIM5_IRQHandler

// 获取捕获寄存器值函数宏定义
#define     GENERAL_TIM_GetCapturex_FUN         TIM_GetCapture1
// 捕获信号极性函数宏定义
#define     GENERAL_TIM_OCxPolarityConfig_FUN   TIM_OC1PolarityConfig

// 测量的起始边沿
#define     GENERAL_TIM_STRAT_ICPolarity     TIM_ICPolarity_Rising
// 测量的结束边沿
#define     GENERAL_TIM_END_ICPolarity       TIM_ICPolarity_Falling
```

使用宏定义非常方便程序升级、移植。有关具体每个宏的含义看注释即可。

定时器复用功能引脚初始化

代码清单 1-10 定时器复用功能引脚初始化

```
static void GENERAL_TIM_GPIO_Config(void)
{
   GPIO_InitTypeDef GPIO_InitStructure;

   // 输入捕获通道 GPIO 初始化
   RCC_APB2PeriphClockCmd(GENERAL_TIM_CH1_GPIO_CLK, ENABLE);
   GPIO_InitStructure.GPIO_Pin =  GENERAL_TIM_CH1_PIN;
   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
   GPIO_Init(GENERAL_TIM_CH1_PORT, &GPIO_InitStructure);
}
```

GENERAL_TIM_GPIO_Config()函数初始化了定时器用到的相关的 GPIO，当使用不同的 GPIO 的时候，只需要修改头文件里面的宏定义即可，而不需要修改这个函数。

定时器模式配置

代码清单 1-11 定时器模式配置

```
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-------------------------*/
   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_PSC;
    // 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);

    /*--------------------输入捕获结构体初始化-------------------*/
    TIM_ICInitTypeDef TIM_ICInitStructure;
    // 配置输入捕获的通道，需要根据具体的 GPIO 来配置
    TIM_ICInitStructure.TIM_Channel = GENERAL_TIM_CHANNEL_x;
    // 输入捕获信号的极性配置
    TIM_ICInitStructure.TIM_ICPolarity = GENERAL_TIM_STRAT_ICPolarity;
    // 输入通道和捕获通道的映射关系，有直连和非直连两种
    TIM_ICInitStructure.TIM_ICSelection = TIM_ICSelection_DirectTI;
    // 输入的需要被捕获的信号的分频系数
    TIM_ICInitStructure.TIM_ICPrescaler = TIM_ICPSC_DIV1;
    // 输入的需要被捕获的信号的滤波系数
    TIM_ICInitStructure.TIM_ICFilter = 0;
    // 定时器输入捕获初始化
    TIM_ICInit(GENERAL_TIM, &TIM_ICInitStructure);

    // 清除更新和捕获中断标志位
    TIM_ClearFlag(GENERAL_TIM, TIM_FLAG_Update|GENERAL_TIM_IT_CCx);
    // 开启更新和捕获中断
   TIM_ITConfig (GENERAL_TIM, TIM_IT_Update | GENERAL_TIM_IT_CCx, ENABLE );

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

ADVANCE_TIM_Mode_Config()函数中初始化了两个结构体，有关这两个结构体成员的具体含义可参考“定时器初始化结构体详解”小节，剩下的程序参考注释阅读即可。

在初始化时基结构体的周期和时钟分频因子这两个成员时，我们使用了两个宏 GENERAL_TIM_PERIOD和GENERAL_TIM_PSC。GENERAL_TIM_PERIOD配置的是ARR寄存器的值，决定了计数器一个周期的计数时间，默认我们配置为 0XFFFF，即最大。GENERAL_TIM_PSC 配置的是分频因子，默认配置为 72-1，则可以计算出计数器的计数周期为(GENERAL_TIM_PSC+1)/72M = 1US。所以输入捕获能捕获的最小的时间为 1us，最长的时间为 1us*(0Xffff+1)=65536us=65.536ms，当超过这个计数周期的时候，就会产生中断，然后在中断里面做额外的处理，需要记录好产生了多少次更新中断，最后把这个更新时间加入到脉宽的时间里面。

中断优先级配置函数

```
// 中断优先级配置
static void GENERAL_TIM_NVIC_Config(void)
{
    NVIC_InitTypeDef NVIC_InitStructure;
    // 设置中断组为 0
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_0);
    // 设置中断来源
    NVIC_InitStructure.NVIC_IRQChannel = GENERAL_TIM_IRQ ;
    // 设置主优先级为 0
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
```

```
    // 设置抢占优先级为 3
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 3;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);
}
```

因为只有一个中断源，所以优先级可以随便配置。

中断服务函数

```
void GENERAL_TIM_INT_FUN(void)
{
    // 当要被捕获的信号的周期大于定时器的最长定时时，定时器就会溢出，产生更新中断
    // 这个时候我们需要把这个最长的定时周期加到捕获信号的时间里面去
    if ( TIM_GetITStatus ( GENERAL_TIM, TIM_IT_Update) != RESET ) {
        TIM_ICUserValueStructure.Capture_Period ++;
        TIM_ClearITPendingBit ( GENERAL_TIM, TIM_FLAG_Update );
    }

    // 上升沿捕获中断
    if ( TIM_GetITStatus (GENERAL_TIM, GENERAL_TIM_IT_CCx ) != RESET) {
        // 第一次捕获
        if ( TIM_ICUserValueStructure.Capture_StartFlag == 0 ) {
            // 计数器清 0
            TIM_SetCounter ( GENERAL_TIM, 0 );
            // 自动重装载寄存器更新标志清 0
            TIM_ICUserValueStructure.Capture_Period = 0;
            // 存捕获比较寄存器的值的变量的值清 0
            TIM_ICUserValueStructure.Capture_CcrValue = 0;

            // 当第一次捕获到上升沿之后，就把捕获边沿配置为下降沿
            GENERAL_TIM_OCxPolarityConfig_FUN(GENERAL_TIM, TIM_ICPolarity_Falling);
            // 开始捕获标准置 1
            TIM_ICUserValueStructure.Capture_StartFlag = 1;
        }
        // 下降沿捕获中断
        else { // 第二次捕获
            // 获取捕获比较寄存器的值，这个值就是捕获到的高电平的时间的值
            TIM_ICUserValueStructure.Capture_CcrValue =
                GENERAL_TIM_GetCapturex_FUN (GENERAL_TIM);

            // 当第二次捕获到下降沿之后，就把捕获边沿配置为上升沿，好开启新的一轮捕获
            GENERAL_TIM_OCxPolarityConfig_FUN(GENERAL_TIM, TIM_ICPolarity_Rising);
            // 开始捕获标志清 0
            TIM_ICUserValueStructure.Capture_StartFlag = 0;
            // 捕获完成标志置 1
            TIM_ICUserValueStructure.Capture_FinishFlag = 1;
        }

        TIM_ClearITPendingBit (GENERAL_TIM,GENERAL_TIM_IT_CCx);
    }
}
```

在 GENERAL_TIM_Mode_Config()函数中，我们配置输入捕获的起始边沿为 GENERAL_TIM_STRAT_ICPolarity，这是一个宏，默认配置为上升沿。我们的按键默认是接 GND，当按键按下的时候会被拉高，这个时候这个由低到高的上升沿会被捕获到，这是第一次捕获，此时我们把计数器清 0，开始计数，同时把捕获边沿改成下降沿捕获。当第二次进入中断服务函数的时候，说明捕获到下降沿，这个时候表示脉宽捕获完毕，我们读取捕获寄存器的值即可，然后我们就可以通过这个值算出脉宽的时间。最后把捕获编译配置为上升沿，为的是下一次捕获。如果脉宽的时间超过了计数器的最大计数时间，那么就会产生更新中断，我们需要做额外的处理，即产生了多少次更新中断记录下来，最后在算脉宽的时间的时候把这个更新的时间加进去即可。

中断服务函数里面用到的捕获结束标志位、捕获开始标志位、捕获寄存器的值和自动重装载更新标志这几个成员是在一个结构体里面定义，具体声明见代码清单 1-12。其中声明是在 bsp_GeneralTim.h 这个头文件中，定义和初始化则在 bsp_GeneralTim.c 文件。

代码清单 1-12 定时器输入捕获用户自定义变量结构体声明

```
// 定时器输入捕获用户自定义变量结构体声明
typedef struct {
   uint8_t   Capture_FinishFlag;   // 捕获结束标志位
   uint8_t   Capture_StartFlag;    // 捕获开始标志位
   uint16_t  Capture_CcrValue;     // 捕获寄存器的值
   uint16_t  Capture_Period;       // 自动重装载寄存器更新标志
} TIM_ICUserValueTypeDef;
```

主函数

代码清单 1-13 main 函数

```
int main(void)
{
   uint32_t time;

   // TIM 计数器的驱动时钟
   uint32_t TIM_PscCLK = 72000000 / (GENERAL_TIM_PSC+1);

   /* 串口初始化 */
   USART_Config();

   /* 定时器初始化 */
   GENERAL_TIM_Init();

   printf ( "\r\n 秉火 STM32 输入捕获实验\r\n" );
   printf ( "\r\n 按下 K1，测试 K1 按下的时间\r\n" );

   while ( 1 ) {
      if (TIM_ICUserValueStructure.Capture_FinishFlag == 1) {
         // 计算高电平时间的计数器的值
         time = TIM_ICUserValueStructure.Capture_Period * (GENERAL_TIM_PERIOD+1) +
            (TIM_ICUserValueStructure.Capture_CcrValue+1);

         // 打印高电平脉宽时间
    printf ( "\r\n 测得高电平脉宽时间：%d.%d s\r\n",time/TIM_PscCLK,time%TIM_PscCLK );

         TIM_ICUserValueStructure.Capture_FinishFlag = 0;
      }
   }
}
```

主函数很简单，主要是一些初始化，然后在一个 while 循环中打印测量的脉宽时间。在计算的时候，记得把周期 GENERAL_TIMPERIOD 和 Capture_CcrValue 的值都加 1 后再运算，因为计数器是从开始计数的。

1.7.3 下载验证

把编译好的程序烧写进开发板，用 USB 线连接好电脑与开发板的 USB 转串口接口，按下 K1 按键，电脑的串口调试助手就会打印出按键按下的时间，具体见图 1-19。



图 1-19 测试按键按下的时间

1.8 PWM 输入捕获实验

上一节我们讲了用输入捕获测量了信号的脉宽，这一节我们讲输入捕获的一个特例—PWM 输入。普通的输入捕获可以使用定时器的四个通道，一路捕获占用一个捕获寄存器，而 PWM 输入则只能使用两个通道，即通道 1 和通道 2，且一路 PWM 输入要占用两个捕获寄存器，一个用于捕获周期，一个用于捕获占空比。在本节实验中，我们用通用定时器产生一路 PWM 信号，然后用高级定时器的通道 1 或者通道 2 来捕获。

1.8.1 硬件设计

实验中用到两个引脚，一个是通用定时器 TIM3 的通道 1，即 PA6，用于输出 PWM 信号，另一个是高级控制定时器 TIM1 的通道 1，即 PA8，用于 PWM 输入捕获，实验中直接用一根杜邦线短接即可 PA6 和 PA8 即可，同时可用示波器监控 PA6 的波形，看看实验捕获的数据是否正确。

1.8.2 软件设计

这里只讲解核心的部分代码，有些变量的设置，头文件的包含等并没有涉及到，完整的代码请参考本章配套的工程。我们创建了 4 个文件：bsp_ AdvanceTim.c 和 bsp_ AdvanceTim.h 文件用来存高级定时器 PWM 输入捕获驱动程序及相关宏定义；bsp_ GeneralTim.c 和 bsp_ GeneralTim.h 文件用来存通用定时器 PWM 信号输出驱动程序及相关宏定义。

1. 编程要点

(1) 通用定时器产生 PWM 配置
(2) 高级定时器 PWM 输入配置
(3) 编写中断服务程序，计算测量的频率和占空比，并打印出来比较

www.firebo

编程的要点主要分成两部分，一个是通用定时器的 PWM 信号输出，另一个是 PWM 信号输入捕获。

2. 通用定时器 PWM 信号输出软件分析

通用定时器宏定义

代码清单 1-14 宏定义

```
/************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
// 输出 PWM 的频率为 72M/{ (ARR+1)*(PSC+1) }
#define           GENERAL_TIM_PERIOD             (10-1)
#define           GENERAL_TIM_PSC               (72-1)

#define           GENERAL_TIM_CCR1               5

// TIM3 输出比较通道 1
#define           GENERAL_TIM_CH1_GPIO_CLK       RCC_APB2Periph_GPIOA
#define           GENERAL_TIM_CH1_PORT           GPIOA
#define           GENERAL_TIM_CH1_PIN            GPIO_Pin_6
```

使用宏定义非常方便程序升级、移植。通过上面的宏，我们可以算出 PWM 信号的频率 F 为：72M/(10*72)=100KHZ，占空比为 GENERAL_TIM_CCR1/（GENERAL_TIM_PERIOD+1）= 50%。

通用定时器引脚初始化

代码清单 1-15 通用定时器引脚初始化

```
/**
  * @brief  通用定时器 PWM 输出用到的 GPIO 初始化
  * @param  无
  * @retval 无
  */
static void GENERAL_TIM_GPIO_Config(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;

    // 输出比较通道 1 GPIO 初始化
    RCC_APB2PeriphClockCmd(GENERAL_TIM_CH1_GPIO_CLK, ENABLE);
    GPIO_InitStructure.GPIO_Pin =  GENERAL_TIM_CH1_PIN;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GENERAL_TIM_CH1_PORT, &GPIO_InitStructure);
}
```

GENERAL_TIM_GPIO_Config()函数初始化了定时器用到的相关的 GPIO，当使用不同的 GPIO 的时候，只需要修改头文件里面的宏定义即可，而不需要修改这个函数。

通用定时器 PWM 输出

代码清单 1-16 通用定时器 PWM 输出

```
/**
  * @brief  通用定时器 PWM 输出初始化
  * @param  无
  * @retval 无
  * @note
  */
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_PSC;
    // 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-------------------*/
    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 1
    TIM_OCInitStructure.TIM_Pulse = GENERAL_TIM_CCR1;
    TIM_OC1Init(GENERAL_TIM, &TIM_OCInitStructure);
    TIM_OC1PreloadConfig(GENERAL_TIM, TIM_OCPreload_Enable);

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

GENERAL_TIM_Mode_Config()函数中初始化了两个结构体，有关这两个结构体成员的具体含义可参考“定时器初始化结构体详解”小节，剩下的程序参考注释阅读即可。

如果需要修改 PWM 的周期和占空比，修改头文件里面的 GENERAL_TIM_PERIOD、GENERAL_TIM_PSC 和 GENERAL_TIM_CCR1 这三个宏即可。PWM 信号的频率的计算公司为：F = TIM_CLK/{(ARR+1)*(PSC+1)}，其中 TIM_CLK 等于 72MHZ，ARR 即自动重装载寄存器的值，对应 GENERAL_TIM_PERIOD 这个宏，PSC 即计数器时钟的分频因子，对应 GENERAL_TIM_PSC 这个宏。

通用定时器初始化

```
/**
  * @brief  通用定时器 PWM 输出用到的 GPIO 和 PWM 模式初始化
  * @param  无
  * @retval 无
  */
void GENERAL_TIM_Init(void)
{
    GENERAL_TIM_GPIO_Config();
    GENERAL_TIM_Mode_Config();
}
```

当调用函数 GENERAL_TIM_Init()之后，相应的引脚就会输出 PWM 信号。

3. 高级定时器 PWM 输入捕获软件分析

输入到高级定时器捕获引脚的 PWM 信号来自通用定时器的输出。

高级定时器宏定义

```
/************高级定时器 TIM 参数定义，只限 TIM1 和 TIM8************/
// 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
// 这里我们使用高级控制定时器 TIM1

#define           ADVANCE_TIM                 TIM1
#define           ADVANCE_TIM_APBxClock_FUN     RCC_APB2PeriphClockCmd
#define           ADVANCE_TIM_CLK             RCC_APB2Periph_TIM1

// 输入捕获能捕获到的最小的频率为 72M/{ (ARR+1)*(PSC+1) }
#define           ADVANCE_TIM_Period          (1000-1)
#define           ADVANCE_TIM_Prescaler        (72-1)

// 中断相关宏定义
#define           ADVANCE_TIM_IRQ             TIM1_CC_IRQn
#define           ADVANCE_TIM_IRQHandler       TIM1_CC_IRQHandler

// TIM1 输入捕获通道 1
#define           ADVANCE_TIM_CH1_GPIO_CLK     RCC_APB2Periph_GPIOA
#define           ADVANCE_TIM_CH1_PORT         GPIOA
#define           ADVANCE_TIM_CH1_PIN          GPIO_Pin_8

#define           ADVANCE_TIM_IC1PWM_CHANNEL    TIM_Channel_1
```

在上面的宏定义里面，我们可以算出计数器的计数周期为 T=72M/(1000*72)=1MS，这个是定时器在不溢出的情况下的最大计数周期，也就是说周期小于 1ms 的 PWM 信号都可以被捕获到，转换成频率就是能捕获到的最小的频率为 1KHZ。所以我们要根据捕获的 PWM 信号来调节 ADVANCE_TIM_PERIOD 和 ADVANCE_TIM_PSC 这两个宏。

高级定时器 PWM 输入模式

代码清单 1-17 高级定时器 PWM 输入模式

```
/**
  * @brief  高级定时器 PWM 输入初始化和用到的 GPIO 初始化
  * @param  无
  * @retval 无
  */
static void ADVANCE_TIM_Mode_Config(void)
{
    // 开启定时器时钟,即内部时钟 CK_INT=72M
    ADVANCE_TIM_APBxClock_FUN(ADVANCE_TIM_CLK,ENABLE);

    /*--------------------时基结构体初始化--------------------------*/
    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=ADVANCE_TIM_PERIOD;
    // 驱动 CNT 计数器的时钟 = Fck_int/(psc+1)
    TIM_TimeBaseStructure.TIM_Prescaler= ADVANCE_TIM_PSC;
    // 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(ADVANCE_TIM, &TIM_TimeBaseStructure);

    /*--------------------输入捕获结构体初始化--------------------*/
```

```
27      // 使用 PWM 输入模式时，需要占用两个捕获寄存器，一个测周期，另外一个测占空比
28
29      TIM_ICInitTypeDef  TIM_ICInitStructure;
30      // 捕获通道 IC1 配置
31      // 选择捕获通道
32      TIM_ICInitStructure.TIM_Channel = ADVANCE_TIM_IC1PWM_CHANNEL;
33      // 设置捕获的边沿
34      TIM_ICInitStructure.TIM_ICPolarity = TIM_ICPolarity_Rising;
35      // 设置捕获通道的信号来自于哪个输入通道，有直连和非直连两种
36      TIM_ICInitStructure.TIM_ICSelection = TIM_ICSelection_DirectTI;
37      // 1 分频，即捕获信号的每个有效边沿都捕获
38      TIM_ICInitStructure.TIM_ICPrescaler = TIM_ICPSC_DIV1;
39      // 不滤波
40      TIM_ICInitStructure.TIM_ICFilter = 0x0;
41      // 初始化 PWM 输入模式
42      TIM_PWMIConfig(ADVANCE_TIM, &TIM_ICInitStructure);
43
44      // 当工作做 PWM 输入模式时,只需要设置触发信号的那一路即可（用于测量周期）
45      // 另外一路（用于测量占空比）会由硬件自带设置，不需要再配置
46
47      // 捕获通道 IC2 配置
48  //  TIM_ICInitStructure.TIM_Channel = ADVANCE_TIM_IC1PWM_CHANNEL;
49  //  TIM_ICInitStructure.TIM_ICPolarity = TIM_ICPolarity_Falling;
50  //  TIM_ICInitStructure.TIM_ICSelection = TIM_ICSelection_IndirectTI;
51  //  TIM_ICInitStructure.TIM_ICPrescaler = TIM_ICPSC_DIV1;
52  //  TIM_ICInitStructure.TIM_ICFilter = 0x0;
53  //  TIM_PWMIConfig(ADVANCE_TIM, &TIM_ICInitStructure);
54
55      // 选择输入捕获的触发信号
56      TIM_SelectInputTrigger(ADVANCE_TIM, TIM_TS_TI1FP1);
57
58      // 选择从模式：复位模式
59      // PWM 输入模式时,从模式必须工作在复位模式，当捕获开始时,计数器 CNT 会被复位
60      TIM_SelectSlaveMode(ADVANCE_TIM, TIM_SlaveMode_Reset);
61      TIM_SelectMasterSlaveMode(ADVANCE_TIM,TIM_MasterSlaveMode_Enable);
62
63      // 使能捕获中断,这个中断针对的是主捕获通道（测量周期那个）
64      TIM_ITConfig(ADVANCE_TIM, TIM_IT_CC1, ENABLE);
65      // 清除中断标志位
66      TIM_ClearITPendingBit(ADVANCE_TIM, TIM_IT_CC1);
67
68      // 使能高级控制定时器，计数器开始计数
69      TIM_Cmd(ADVANCE_TIM, ENABLE);
70  }
```

ADVANCE_TIM_Mode_Config()函数中初始化了两个结构体，有关这两个结构体成员的具体含义可参考“定时器初始化结构体详解”小节，剩下的程序参考注释阅读即可。

因为是 PWM 输入模式，只能使用通道 1 和通道 2 ，假如我们使用的是通道 1，即 TI1，输入的 PWM 信号会被分成两路，分别是 TI1FP1 和 TI1FP2，两路都可以是触发信号。如果选择 TI1FP1 为触发信号，那么 IC1 捕获到的是 PWM 信号的周期，IC2 捕获到的是占空比，这种输入通道 TI 和捕获通道 IC 的映射关系叫直连，输入捕获结构体的 TIM_ICSelection 要配置为 TIM_ICSelection_DirectTI。如果选择 TI1FP2 为触发信号，则 IC2 捕获到的是周期，IC1 捕获到的是占空比，这种输入通道 TI 和捕获通道 IC 的映射关系叫非直连，输入捕获结构体的 TIM_ICSelection 要配置为 TIM_ICSelection_IndirectTI。有关输入通道 TI 和捕获通道 IC 的具体映射关系见图 1-20，有直连和非直连两种。

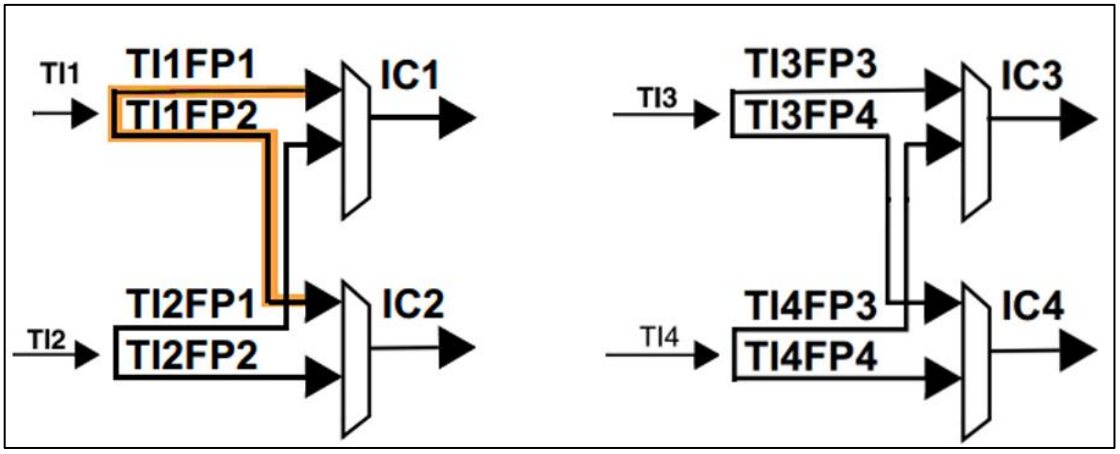


图 1-20 输入通道 TI 和捕获通道 IC 的映射图

高级定时器中断优先级

```c
/**
 * @brief  高级控制定时器 TIMx,x[1,8]中断优先级配置
 * @param  无
 * @retval 无
 */
static void ADVANCE_TIM_NVIC_Config(void)
{
    NVIC_InitTypeDef NVIC_InitStructure;
    // 设置中断组为 0
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_0);
    // 设置中断来源
    NVIC_InitStructure.NVIC_IRQChannel = ADVANCE_TIM_IRQ;
    // 设置抢占优先级
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
    // 设置子优先级
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 3;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);
}
```

因为只有一个中断源，优先级可以随便配置。

高级定时器中断服务函数

```c
void ADVANCE_TIM_IRQHandler(void)
{
    /* 清除中断标志位 */
    TIM_ClearITPendingBit(ADVANCE_TIM, TIM_IT_CC1);

    /* 获取输入捕获值 */
    IC1Value = TIM_GetCapture1(ADVANCE_TIM);
    IC2Value = TIM_GetCapture2(ADVANCE_TIM);

    // 注意：捕获寄存器 CCR1 和 CCR2 的值在计算占空比和频率的时候必须加 1
    if (IC1Value != 0) {
        /* 占空比计算 */
        DutyCycle = (float)((IC2Value+1) * 100) / (IC1Value+1);

        /* 频率计算 */
        Frequency = (72000000/(ADVANCE_TIM_PSC+1))/(float)(IC1Value+1);
        printf("占空比：%0.2f%%   频率：%0.2fHz\n",DutyCycle,Frequency);
    } else {
        DutyCycle = 0;
        Frequency = 0;
    }
```

```
22 }
```

当捕获到 PWM 信号的第一个上升沿时，产生中断，计数器被复位，锁存到捕获寄存器 IC1 和 IC2 的值都为 0。当下降沿到来时，IC2 会捕获，对应的是占空比，但是会产生中断。当捕获到第二个下降沿时，IC1 会捕获，对应的是周期，而且会再次进入中断，这个时间就可以根据 IC1 和 IC2 的值计算出频率和占空比。有关 PWM 输入的时序见图 1-21。

中断复位函数中，我们获取输入捕获寄存器 CCR1 和 CCR2 寄存器中的值，当 CCR1 的值不为 0 时，说明有效捕获到了一个周期，然后计算出频率和占空比。在计算的时候 CCR1 和 CCR2 的值都必须要加 1，因为计数器是从 0 开始计数的。

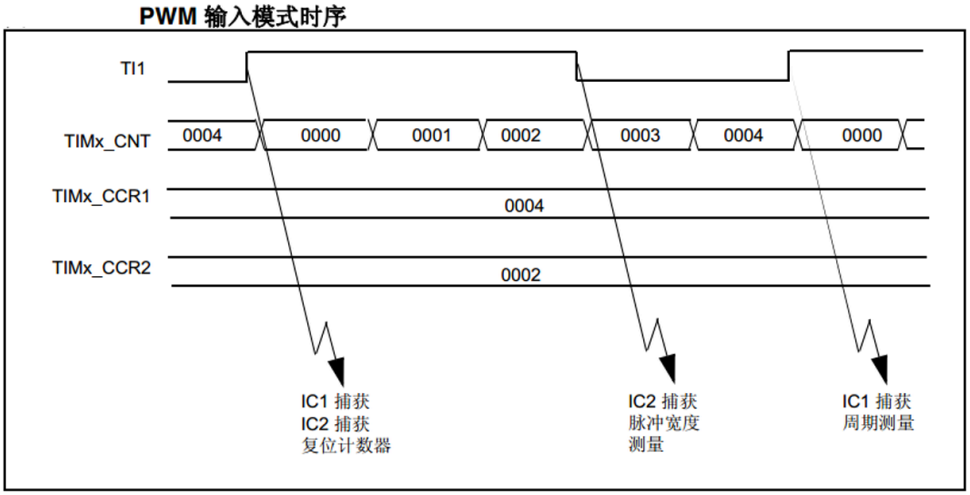


图 1-21 PWM 输入模式时序图

4. 主函数

代码清单 1-18 main 函数

```
1 /**
2   * @brief  主函数
3   * @param  无
4   * @retval 无
5   */
6 int main(void)
7 {
8    /* 串口初始化 */
9    USART_Config();
10
11   /* 通用定时器初始化，用于生成 PWM 信号 */
12   GENERAL_TIM_Init();
13
14   /* 高级定时器初始化 ，用户捕获 PWM 信号*/
15   ADVANCE_TIM_Init();
16
17   while (1) {
18   }
19 }
```

main 函数非常简单，通用定时器初始化完之后用于输出 PWM 信号，高级定时器初始化完之后用于捕获通用定时器输出的 PWM 信号。

1.8.3 下载验证

把编译好的程序烧写到开发板，用杜邦线把通用定时器的 PWM 输出引脚连接到高级定时器的 PWM 输入引脚。然后用 USB 线连接电脑与开发板的 USB 转串口，打开串口调试助

手，即可看到捕获到的 PWM 信号的频率和占空比，具体见。于此同时，可用示波器监控通用定时器输出的 PWM 信号，看下捕获到的信号是否正确，具体见。

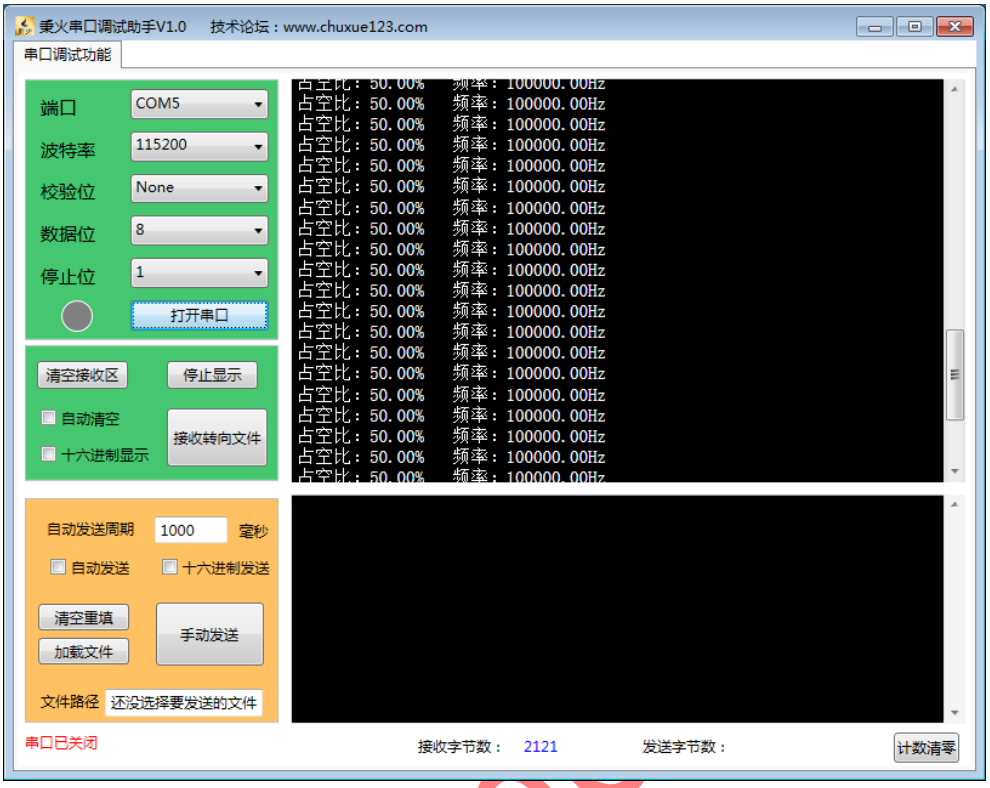


图 1-22 串口调试助手打印的捕获信息

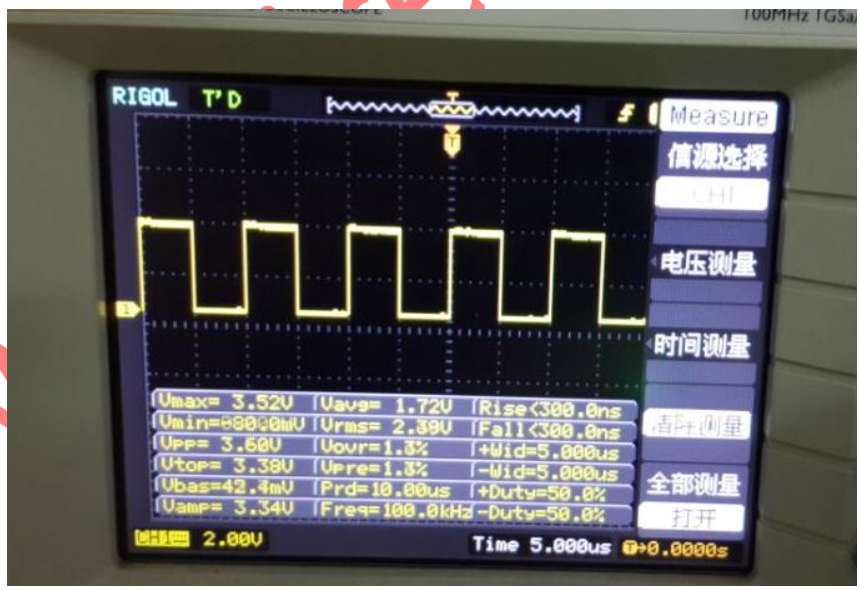


图 1-23 示波器监控的波形

从图 1-21 和图 1-23 我们可以看到，程序捕获计算出的频率和占空比和示波器监控到的波形的频率和占空比跟一致，所以我们的程序是正确的。

www.firebbs.cn