25 16 位高级 PWM 定时器，支持正交编码器

产品线	高级 PWM	PWM 输出口模式设置	
		硬件强制设置为强推挽输出模式	用户软件设置
STC8H1K08 系列	●	●	
STC8H1K28 系列	●	●	
STC8H3K64S4 系列	●	●	
STC8H3K64S2 系列	●	●	
STC8H8K64U 系列	●	●	
STC8H4K64TL 系列	●	●	
STC8H4K64TLCD 系列	●	●	
STC8H1K08T 系列	●		●
STC8H2K12U-A/B 系列	●		●
STC8H2K32U 系列	●		●
STC8G1K08-SOP8 系列			
STC8G1K08A-SOP8 系列			

STC8H 系列的单片机内部集成了 8 通道 16 位高级 PWM 定时器，分成两组周期可不同的 PWM，分别命名为 PWMA 和 PWMB（之前的数据手册曾命名为 PWM1 和 PWM2，但容易与芯片管脚名称混淆，故更改为 PWMA 和 PWMB），可分别单独设置。第一组 PWM/PWMA 可配置成 4 组互补/对称/死区控制的 PWM 或捕捉外部信号，第二组 PWM/PWMB 可配置成 4 路 PWM 输出或捕捉外部信号。

第一组 PWM/PWMA 的时钟频率可以是系统时钟经过寄存器 PWMA_PSCRH 和 PWMA_PSCRL 进行分频后的时钟，分频值可以是 1~65535 之间的任意值。第二组 PWM/PWMB 的时钟频率可以是系统时钟经过寄存器 PWMB_PSCRH 和 PWMB_PSCRL 进行分频后的时钟，分频值可以是 1~65535 之间的任意值。两组 PWM 的时钟频率可分别独立设置。

第一组 PWM 定时器/PWMA 有 4 个通道（PWM1P/PWM1N、PWM2P/PWM2N、PWM3P/PWM3N、PWM4P/PWM4N），每个通道都可独立实现 PWM 输出（可设置带死区的互补对称 PWM 输出）、捕获和比较功能；第二组 PWM 定时器/PWMB 有 4 个通道（PWM5、PWM6、PWM7、PWM8），每个通道也可独立实现 PWM 输出、捕获和比较功能。两组 PWM 定时器唯一的区别是第一组可输出带死区的互补对称 PWM，而第二组只能输出单端的 PWM，其他功能完全相同。下面关于高级 PWM 定时器的介绍只以第一组为例进行说明。

当使用第一组 PWM 定时器输出 PWM 波形时，可单独使能 PWM1P/PWM2P/PWM3P/PWM4P 输出，也可单独使能 PWM1N/PWM2N/PWM3N/PWM4N 输出。例如：若单独使能了 PWM1P 输出，则 PWM1N 就不能再独立输出，除非 PWM1P 和 PWM1N 组成一组互补对称输出。PWMA 的 4 路输出是可分别独立设置的，例如：可单独使能 PWM1P 和 PWM2N 输出，也可单独使能 PWM2N 和 PWM3N 输出。若需要使用第一组 PWM 定时器进行捕获功能或者测量脉宽时，输入信号只能从每路的正端输入，即只有 PWM1P/PWM2P/PWM3P/PWM4P 才有捕获功能和测量脉宽功能。

两组高级 PWM 定时器对外部信号进行捕获时，可选择上升沿捕获或者下降沿捕获。如果需要同时捕获上升沿和下降沿，则可将输入信号同时接入到两路 PWM，使能其中一路捕获上升沿，另外一路捕获下降沿即可。**更强悍的是，将外部输入信号同时接入到两路 PWM 时，可同时捕获信号的周期值和占空比值**。

STC 三种硬件 PWM 比较：

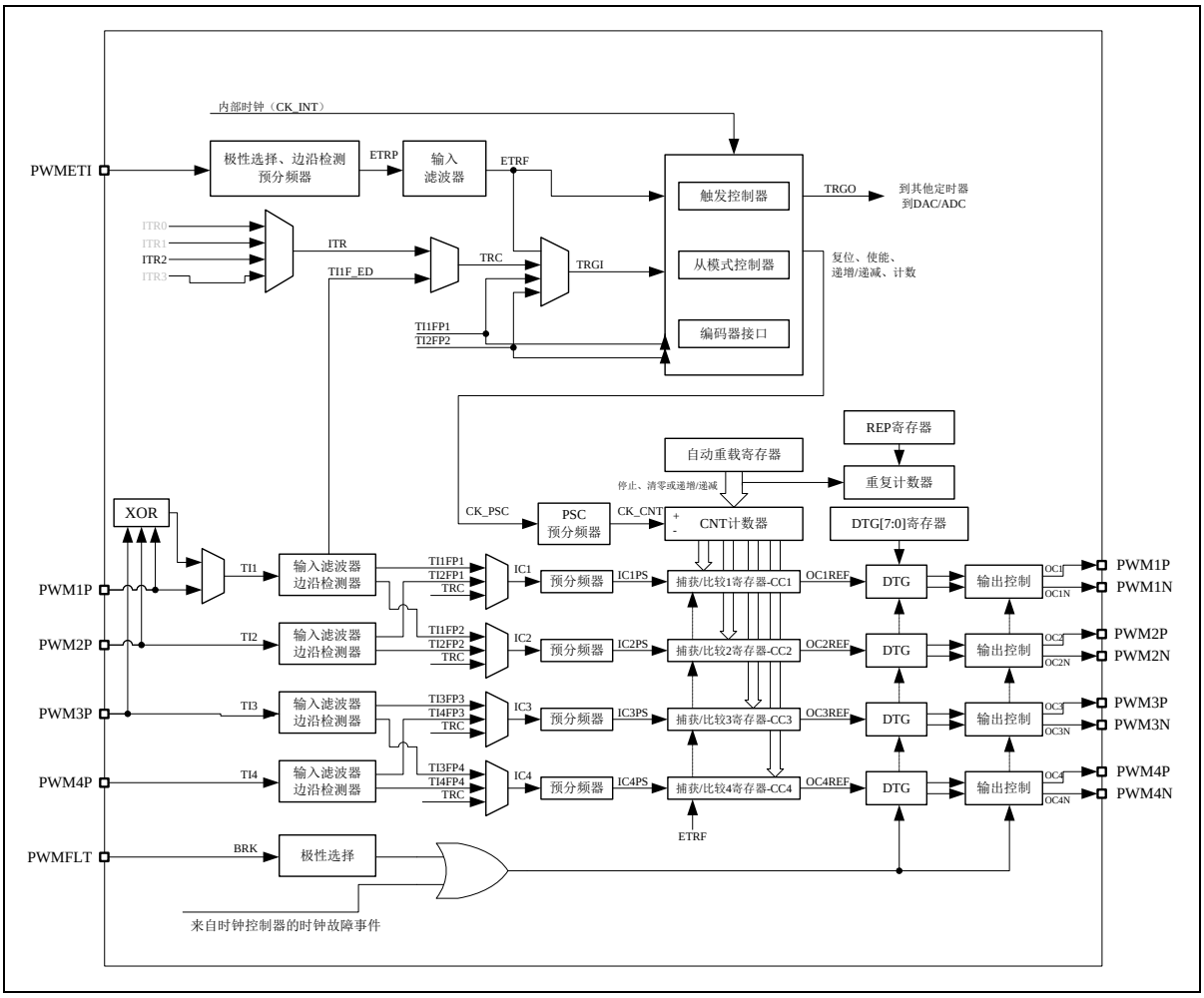
兼容传统 8051 的 PCA/CCP/PWM：可输出 PWM 波形、捕获外部输入信号以及输出高速脉冲。可对外输出 6 位/7 位/8 位/10 位的 PWM 波形，6 位 PWM 波形的频率为 PCA 模块时钟源频率/64；7 位 PWM 波形的频率为 PCA 模块时钟源频率/128；8 位 PWM 波形的频率为 PCA 模块时钟源频率/256；10 位 PWM 波形的频率为 PCA 模块时钟源频率/1024。捕获外部输入信号，可捕获上升沿、下降沿或者同时捕获上升沿和下降沿。

STC8G 系列的 15 位增强型 PWM：只能对外输出 PWM 波形，无输入捕获功能。对外输出 PWM 的频率以及占空比均可任意设置。通过软件干预，可实现多路互补/对称/带死区的 PWM 波形。有外部异常检测功能以及实时触发 ADC 转换功能。

STC8H 系列的 16 位高级 PWM 定时器：是目前 STC 功能最强的 PWM，可对外输出任意频率以及任意占空比的 PWM 波形。无需软件干预即可输出互补/对称/带死区的 PWM 波形。能捕获外部输入信号，可捕获上升沿、下降沿或者同时捕获上升沿和下降沿，测量外部波形时，可同时测量波形的周期值和占空比值。有正交编码功能、外部异常检测功能以及实时触发 ADC 转换功能。

下面的说明中，PWMA 代表第一组 PWM 定时器，PWMB 代表第二组 PWM 定时器

25.1 高级 PWM 定时器（PWMA）内部结构框图



TI1：外部时钟输入信号 1（PWM1P 管脚信号或者 PWM1P/PWM2P/PWM3P 相异或后的信号）
TI1F：经过 IC1F 数字滤波后的 TI1 信号
TI1FP：经过 CC1P/CC2P 边沿检测器后的 TI1F 信号
TI1F_ED：TI1F 的边沿信号
TI1FP1：经过 CC1P 边沿检测器后的 TI1F 信号
TI1FP2：经过 CC2P 边沿检测器后的 TI1F 信号
IC1：通过 CC1S 选择的通道 1 的捕获输入信号
OC1REF：输出通道 1 输出的参考波形（中间波形）
OC1：通道 1 的主输出信号（经过 CC1P 极性处理后的 OC1REF 信号）
OC1N：通道 1 的互补输出信号（经过 CC1NP 极性处理后的 OC1REF 信号）
CC1：通道 1 的捕获/比较寄存器

TI2：外部时钟输入信号 2（PWM2P 管脚信号）
TI2F：经过 IC2F 数字滤波后的 TI2 信号
TI2F_ED：TI2F 的边沿信号
TI2FP：经过 CC1P/CC2P 边沿检测器后的 TI2F 信号
TI2FP1：经过 CC1P 边沿检测器后的 TI2F 信号
TI2FP2：经过 CC2P 边沿检测器后的 TI2F 信号
IC2：通过 CC2S 选择的通道 2 的捕获输入信号

OC2REF：输出通道 2 输出的参考波形（中间波形）
OC2：通道 2 的主输出信号（经过 CC2P 极性处理后的 OC2REF 信号）
OC2N：通道 2 的互补输出信号（经过 CC2NP 极性处理后的 OC2REF 信号）
CC2：通道 2 的捕获/比较寄存器

TI3：外部时钟输入信号 3（PWM3P 管脚信号）
TI3F：经过 IC3F 数字滤波后的 TI3 信号
TI3F_ED：TI3F 的边沿信号
TI3FP：经过 CC3P/CC4P 边沿检测器后的 TI3F 信号
TI3FP3：经过 CC3P 边沿检测器后的 TI3F 信号
TI3FP4：经过 CC4P 边沿检测器后的 TI3F 信号
IC3：通过 CC3S 选择的通道 3 的捕获输入信号
OC3REF：输出通道 3 输出的参考波形（中间波形）
OC3：通道 3 的主输出信号（经过 CC3P 极性处理后的 OC3REF 信号）
OC3N：通道 3 的互补输出信号（经过 CC3NP 极性处理后的 OC3REF 信号）
CC3：通道 3 的捕获/比较寄存器

TI4：外部时钟输入信号 4（PWM4P 管脚信号）
TI4F：经过 IC4F 数字滤波后的 TI4 信号
TI4F_ED：TI4F 的边沿信号
TI4FP：经过 CC3P/CC4P 边沿检测器后的 TI4F 信号
TI4FP3：经过 CC3P 边沿检测器后的 TI4F 信号
TI4FP4：经过 CC4P 边沿检测器后的 TI4F 信号
IC4：通过 CC4S 选择的通道 4 的捕获输入信号
OC4REF：输出通道 4 输出的参考波形（中间波形）
OC4：通道 4 的主输出信号（经过 CC4P 极性处理后的 OC4REF 信号）
OC4N：通道 4 的互补输出信号（经过 CC4NP 极性处理后的 OC4REF 信号）
CC4：通道 4 的捕获/比较寄存器

ITR1：内部触发输入信号 1
ITR2：内部触发输入信号 2

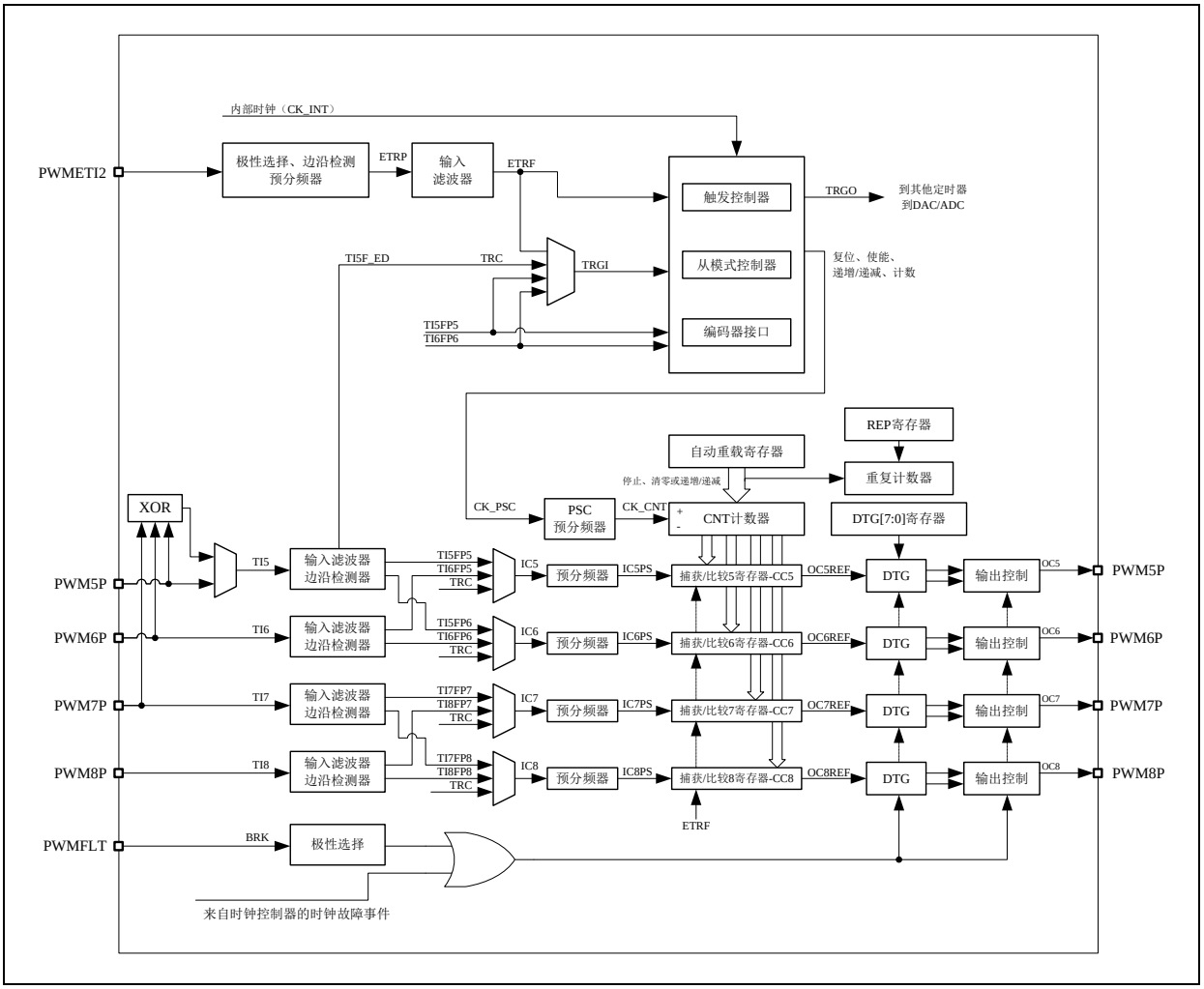
TRC：固定为 TI1_ED
TRGI：经过 TS 多路选择器后的触发输入信号
TRGO：经过 MMS 多路选择器后的触发输出信号

ETR：外部触发输入信号（PWMETI1 管脚信号）
ETRP：经过 ETP 边沿检测器以及 ETPS 分频器后的 ETR 信号
ETRF：经过 ETF 数字滤波后的 ETRP 信号

BRK：刹车输入信号（PWMFLT）

CK_PSC：预分频时钟，PWMA_PSCR 预分频器的输入时钟
CK_CNT：PWMA_PSCR 预分频器的输出时钟，PWM 定时器的时钟

25.2 高级 PWM 定时器（PWMB）内部结构框图



TI5：外部时钟输入信号 5（PWM5 管脚信号或者 PWM5/PWM6/PWM7 相异或后的信号）
TI5F：经过 IC5F 数字滤波后的 TI5 信号
TI5FP：经过 CC5P/CC6P 边沿检测器后的 TI5F 信号
TI5F_ED：TI5F 的边沿信号
TI5FP5：经过 CC5P 边沿检测器后的 TI5F 信号
TI5FP6：经过 CC6P 边沿检测器后的 TI5F 信号
IC5：通过 CC5S 选择的通道 5 的捕获输入信号
OC5REF：输出通道 5 输出的参考波形（中间波形）
OC5：通道 5 的主输出信号（经过 CC5P 极性处理后的 OC5REF 信号）
CC5：通道 5 的捕获/比较寄存器

TI6：外部时钟输入信号 6（PWM6 管脚信号）
TI6F：经过 IC6F 数字滤波后的 TI6 信号
TI6F_ED：TI6F 的边沿信号
TI6FP：经过 CC5P/CC6P 边沿检测器后的 TI6F 信号
TI6FP5：经过 CC5P 边沿检测器后的 TI6F 信号
TI6FP6：经过 CC6P 边沿检测器后的 TI6F 信号
IC6：通过 CC6S 选择的通道 6 的捕获输入信号
OC6REF：输出通道 6 输出的参考波形（中间波形）

OC6：通道 6 的主输出信号（经过 CC6P 极性处理后的 OC6REF 信号）
CC6：通道 6 的捕获/比较寄存器

TI7：外部时钟输入信号 7（PWM7 管脚信号）
TI7F：经过 IC7F 数字滤波后的 TI7 信号
TI7F_ED：TI7F 的边沿信号
TI7FP：经过 CC7P/CC8P 边沿检测器后的 TI7F 信号
TI7FP7：经过 CC7P 边沿检测器后的 TI7F 信号
TI7FP8：经过 CC8P 边沿检测器后的 TI7F 信号
IC7：通过 CC7S 选择的通道 7 的捕获输入信号
OC7REF：输出通道 7 输出的参考波形（中间波形）
OC7：通道 7 的主输出信号（经过 CC7P 极性处理后的 OC7REF 信号）
CC7：通道 7 的捕获/比较寄存器

TI8：外部时钟输入信号 8（PWM8 管脚信号）
TI8F：经过 IC8F 数字滤波后的 TI8 信号
TI8F_ED：TI8F 的边沿信号
TI8FP：经过 CC7P/CC8P 边沿检测器后的 TI8F 信号
TI8FP7：经过 CC7P 边沿检测器后的 TI8F 信号
TI8FP8：经过 CC8P 边沿检测器后的 TI8F 信号
IC8：通过 CC8S 选择的通道 8 的捕获输入信号
OC8REF：输出通道 8 输出的参考波形（中间波形）
OC8：通道 8 的主输出信号（经过 CC8P 极性处理后的 OC8REF 信号）
CC8：通道 8 的捕获/比较寄存器

25.3 简介

PWMA 由一个 16 位的自动装载计数器组成，它由一个可编程的预分频器驱动。

PWMA 适用于许多不同的用途：

- 基本的定时
- 测量输入信号的脉冲宽度（输入捕获）
- 产生输出波形（输出比较，PWM 和单脉冲模式）
- 对应与不同事件（捕获，比较，溢出，刹车，触发）的中断
- 与 PWMB 或者外部信号（外部时钟，复位信号，触发和使能信号）同步

PWMA 广泛的适用于各种控制应用中，包括那些需要中间对齐模式 PWM 的应用，该模式支持互补输出和死区时间控制。PWMA 的时钟源可以是内部时钟，也可以是外部的信号，可以通过配置寄存器来进行选择。

25.4 主要特性

PWMA 的特性包括：

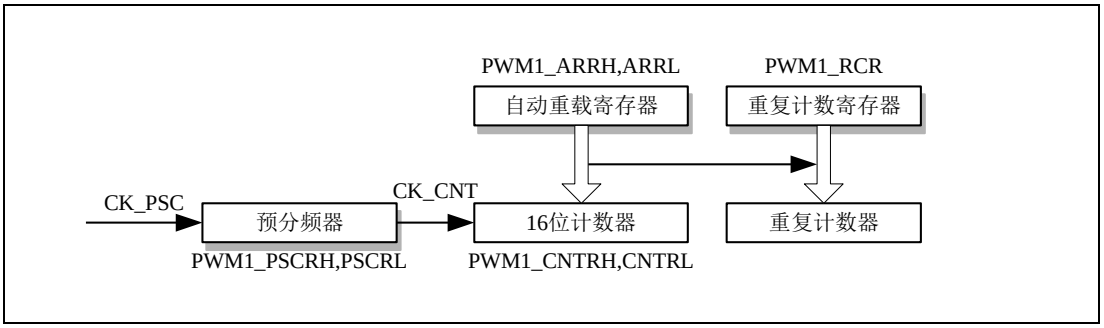
- 16 位向上、向下、向上/下自动装载计数器
- 允许在指定数目的计数器周期之后更新定时器寄存器的重复计数器
- 16 位可编程（可以实时修改）预分频器，计数器时钟频率的分频系数为 1~65535 之间的任意数值
- 同步电路，用于使用外部信号控制定时器以及定时器互联
- 多达 4 个独立通道可以配置成：
  - 输入捕获
  - 输出比较
  - PWM 输出（边缘或中间对齐模式）
  - 六步 PWM 输出
  - 单脉冲模式输出
  - 支持 4 个死区时间可编程的通道上互补输出
- 刹车输入信号（PWMFLT）可以将定时器输出信号置于复位状态或者一个确定状态
- 外部触发输入引脚（PWMETI）
- 产生中断的事件包括：
  - 更新：计数器向上溢出/向下溢出，计数器初始化（通过软件或者内部/外部触发）
  - 触发事件（计数器启动、停止、初始化或者由内部/外部触发计数）
  - 输入捕获，测量脉宽
  - 外部中断
  - 输出比较
  - 刹车信号输入

25.5 时基单元

PWMA 的时基单元包含：

- 16 位向上/向下计数器
- 16 位自动重载寄存器
- 重复计数器
- 预分频器

PWMA 时基单元



16 位计数器、预分频器、自动重载寄存器和重复计数器寄存器都可以通过软件进行读写操作。自动重载寄存器由预装载寄存器和影子寄存器组成。

可在在两种模式下写自动重载寄存器：

- 自动预装载已使能（PWMA_CR1 寄存器的 ARPE 位为 1）。在此模式下，写入自动重载寄存器的数据将被保存在预装载寄存器中，并在下一个更新事件（UEV）时传送到影子寄存器。
- 自动预装载已禁止（PWMA_CR1 寄存器的 ARPE 位为 0）。在此模式下，写入自动重载寄存器的数据将立即写入影子寄存器。

更新事件的产生条件：

- 计数器向上或向下溢出。
- 软件置位了 PWMA_EGR 寄存器的 UG 位。
- 时钟/触发控制器产生了触发事件。

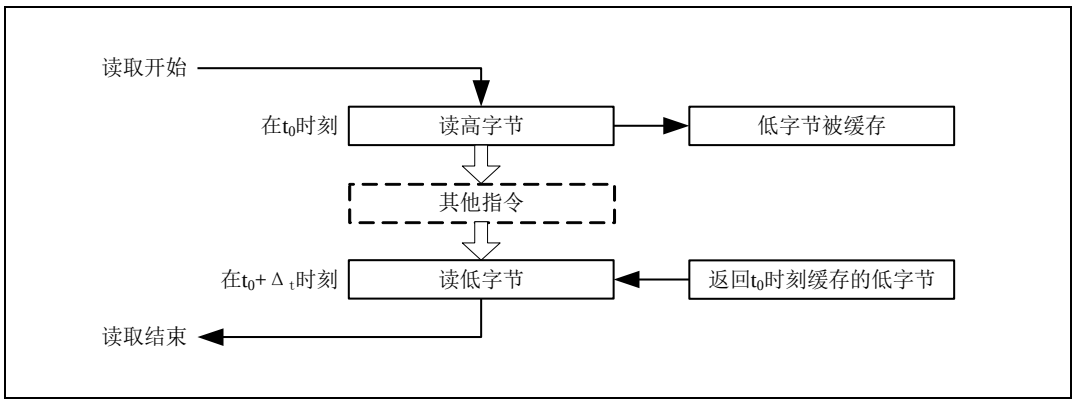
在预装载使能时（ARPE=1），如果发生了更新事件，预装载寄存器中的数值（PWMA_ARR）将写入影子寄存器中，并且 PWMA_PSCR 寄存器中的值将写入预分频器中。置位 PWMA_CR1 寄存器的 UDIS 位将禁止更新事件（UEV）。预分频器的输出 CK_CNT 驱动计数器，而 CK_CNT 仅在 PWMA_CR1 寄存器的计数器使能位（CEN）被置位时才有效。

注意：实际的计数器在 CEN 位使能的一个时钟周期后才开始计数。

25.5.1 读写 16 位计数器

写计数器的操作没有缓存，在任何时候都可以写 PWMA_CNTRH 和 PWMA_CNTRL 寄存器，因此为避免写入了错误的数值，一般建议不要在计数器运行时写入新的数值。

读计数器的操作带有 8 位的缓存。用户必须先读定时器的高字节，在用户读了高字节后，低字节将被自动缓存，缓存的数据将会一直保持直到 16 位数据的读操作完成。



25.5.2 16 位 PWMA_ARR 寄存器的写操作

预装载寄存器中的值将写入 16 位的 PWMA_ARR 寄存器中，此操作由两条指令完成，每条指令写入 1 个字节。必须先写高字节，后写低字节。

影子寄存器在写入高字节时被锁定，并保持到低字节写完。

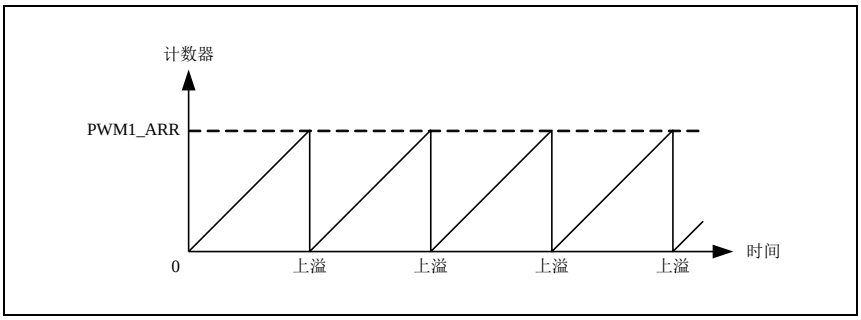
25.5.3 预分频器

PWMA 的预分频器基于一个由 16 位寄存器（PWMA_PSCR）控制的 16 位计数器。由于这个控制寄存器带有缓冲器，因此它能够在运行时被改变。预分频器可以将计数器的时钟频率按 1 到 65536 之间的任意值分频。预分频器的值由预装载寄存器写入，保存了当前使用值的影子寄存器在低字节写入时被载入。由于需两次单独的写操作来写 16 位寄存器，因此必须保证高字节先写入。新的预分频器的值在下一次更新事件到来时被采用。对 PWMA_PSCR 寄存器的读操作通过预装载寄存器完成。

计数器的频率计算公式：$f_{CK\_CNT} = f_{CK\_PSC} / (PSCR[15:0] + 1)$

25.5.4 向上计数模式

在向上计数模式中，计数器从 0 计数到用户定义的比较值（PWMA_ARR 寄存器的值），然后重新从 0 开始计数并产生一个计数器溢出事件，此时如果 PWMA_CR1 寄存器的 UDIS 位是 0，将会产生一个更新事件（UEV）。



通过软件方式或者通过使用触发控制器置位 PWMA_EGR 寄存器的 UG 位同样也可以产生一个更新事件。使用软件置位 PWMA_CR1 寄存器的 UDIS 位，可以禁止更新事件，这样可以避免在更新预装载寄存器时更新影子寄存器。在 UDIS 位被清除之前，将不产生更新事件。但是在应该产生更新事件时，计数器仍会被清 0，同时预分频器的计数也被清 0（但预分频器的数值不变）。此外，如果设置了 PWMA_CR1 寄存器中的 URS 位（选择更新请求），设置 UG 位将产生一个更新事件 UEV，但硬件不设置 UIF 标志（即不产生中断请求）。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

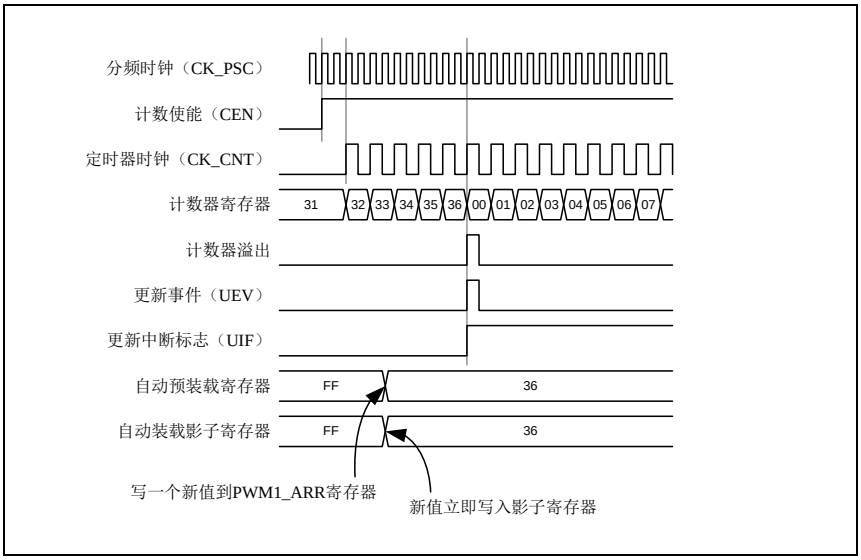
断。

当发生一个更新事件时，所有的寄存器都被更新，硬件依据 URS 位同时设置更新标志位（PWMA_SR 寄存器的 UIF 位）：

- 自动装载影子寄存器被重新置入预装载寄存器的值（PWMA_ARR）。
- 预分频器的缓存器被置入预装载寄存器的值（PWMA_PSC）。

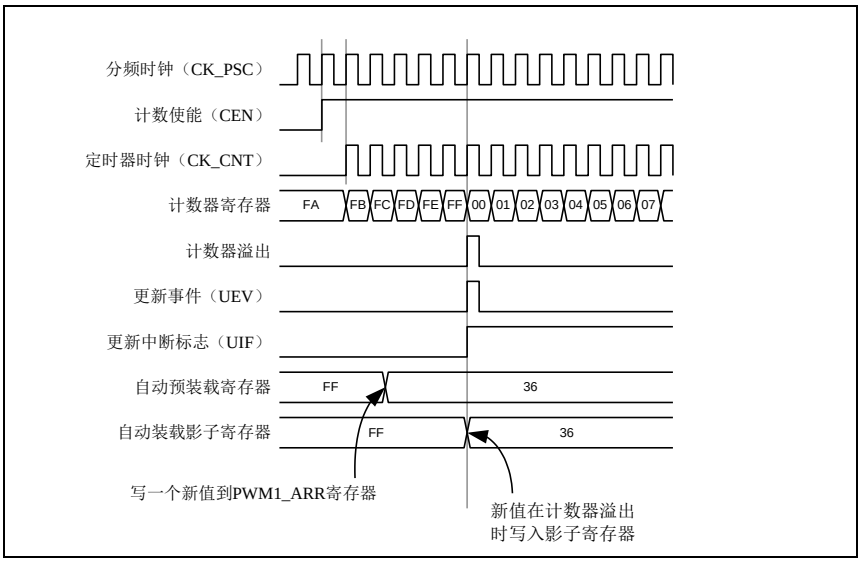
下图给出一些例子，说明当 PWMA_ARR=0x36 时，计数器在不同时钟频率下的动作。图中预分频为 2，因此计数器的时钟（CK_CNT）频率是预分频时钟（CK_PSC）频率的一半。图中禁止了自动装载功能（ARPE=0），所以在计数器达到 0x36 时，计数器溢出，影子寄存器立刻被更新，同时产生一个更新事件。

当 ARPE=0（ARR 不预装载），预分频为 2 时的计数器更新：



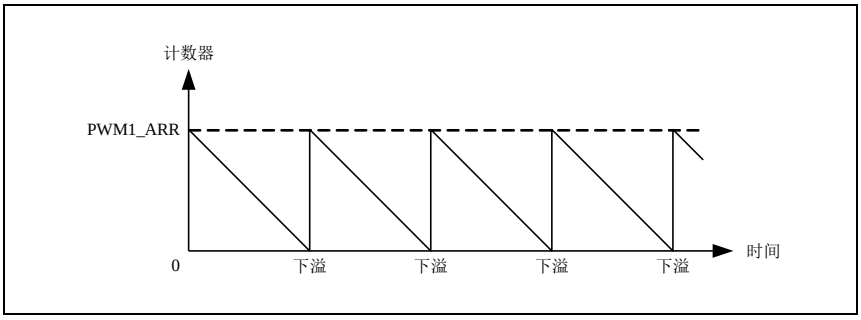
下图的预分频为 1，因此 CK_CNT 的频率与 CK_PSC 一致。图中使能了自动重载（ARPE=1），所以在计数器达到 0xFF 产生溢出。0x36 将在溢出时被写入，同时产生一个更新事件。

当 ARPE=1(PWMA_ARR 预装载)，预分频为 1 时的计数器更新：



25.5.5 向下计数模式

在向下模式中，计数器从自动装载的值（PWMA_ARR 寄存器的值）开始向下计数到 0，然后再从自动装载的值重新开始计数，并产生一个计数器向下溢出事件。如果 PWMA_CR1 寄存器的 UDIS 位被清除，还会产生一个更新事件（UEV）。



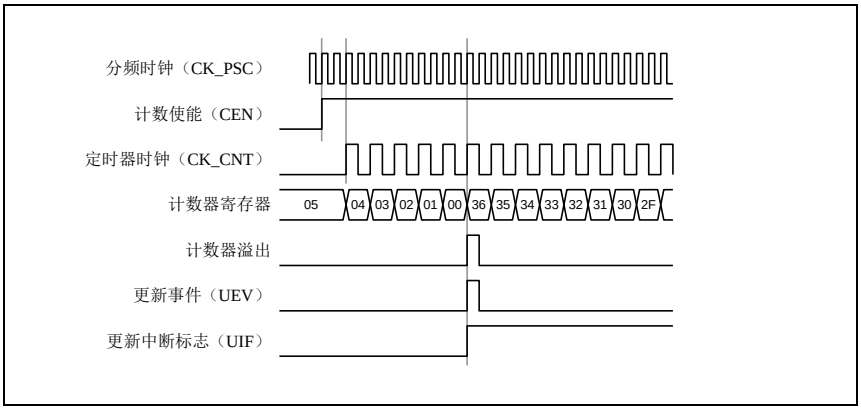
通过软件方式或者通过使用触发控制器置位 PWMA_EGR 寄存器的 UG 位同样也可以产生一个更新事件。置位 PWMA_CR1 寄存器的 UDIS 位可以禁止 UEV 事件。这样可以避免在更新预装载寄存器时更新影子寄存器。因此 UDIS 位清除之前不会产生更新事件。然而，计数器仍会从当前自动加载值重新开始计数，并且预分频器的计数器重新从 0 开始（但预分频器不能被修改）。此外，如果设置了 PWMA_CR1 寄存器中的 URS 位（选择更新请求），设置 UG 位将产生一个更新事件 UEV 但不设置 UIF 标志（因此不产生中断），这是为了避免在发生捕获事件并清除计数器时，同时产生更新和捕获中断。

当发生更新事件时，所有的寄存器都被更新，硬件依据 URS 位同时设置更新标志位（PWMA_SR 寄存器的 UIF 位）：

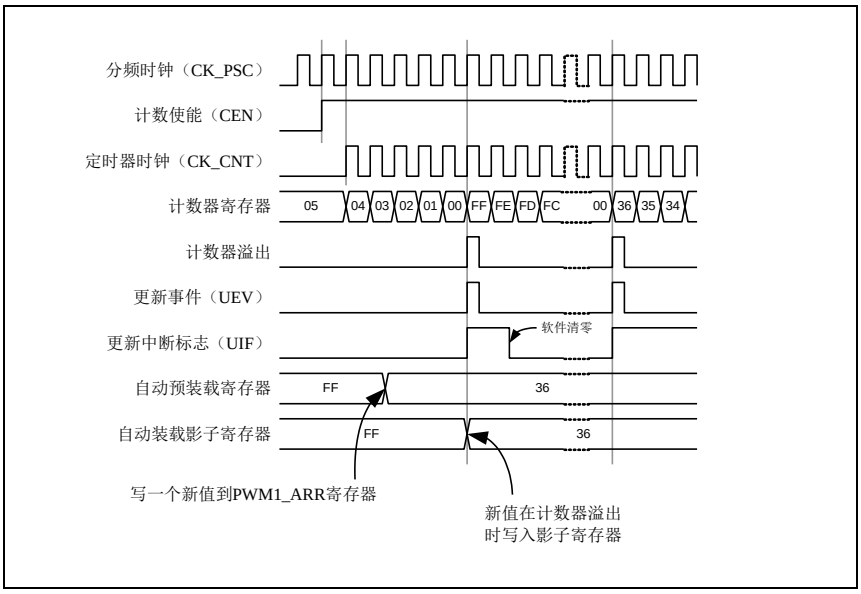
- 自动装载影子寄存器被重新置入预装载寄存器的值（PWMA_ARR）。
- 预分频器的缓存器被置入预装载寄存器的值（PWMA_PSC）。

以下是一些当 PWMA_ARR=0x36 时，计数器在不同时钟频率下的图表。下图描述了在向下计数模式下，预装载不使能时新的数值在下个周期时被写入。

当 ARPE=0（ARR 不预装载），预分频为 2 时的计数器更新：

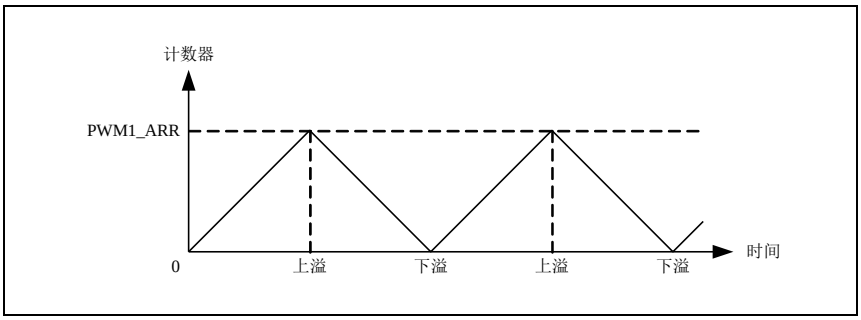


当 ARPE=1（ARR 预装载），预分频为 1 时的计数器更新



25.5.6 中间对齐模式（向上/向下计数）

在中央对齐模式，计数器从 0 开始计数到 PWMA_ARR 寄存器的值-1，产生一个计数器上溢事件，然后从 PWMA_ARR 寄存器的值向下计数到 1 并且产生一个计数器下溢事件；然后再从 0 开始重新计数。在此模式下，不能写入 PWMA_CR1 中的 DIR 方向位。它由硬件更新并指示当前的计数方向。



如果定时器带有重复计数器，在重复了指定次数（PWMA_RCR 的值）的向上和向下溢出之后会产生更新事件（UEV）。否则每一次的向上向下溢出都会产生更新事件。通过软件方式或者通过使用触发控制器置位 PWMA_EGR 寄存器的 UG 位同样也可以产生一个更新事件。此时，计数器重新从 0 开始计数，预分频器也重新从 0 开始计数。设置 PWMA_CR1 寄存器中的 UDIS 位可以禁止 UEV 事件。这样可以避免在更新预装载寄存器时更新影子寄存器。因此 UDIS 位被清为 0 之前不会产生更新事件。然而，计数器仍会根据当前自动重加载的值，继续向上或向下计数。如果定时器带有重复计数器，由于重复寄存器没有双重的缓冲，新的重复数值将立刻生效，因此在修改时需要小心。此外，如果设置了 PWMA_CR1 寄存器中的 URS 位（选择更新请求），设置 UG 位将产生一个更新事件 UEV 但不设置 UIF 标志（因此不产生中断），这是为了避免在发生捕获事件并清除计数器时，同时产生更新和捕获中断。

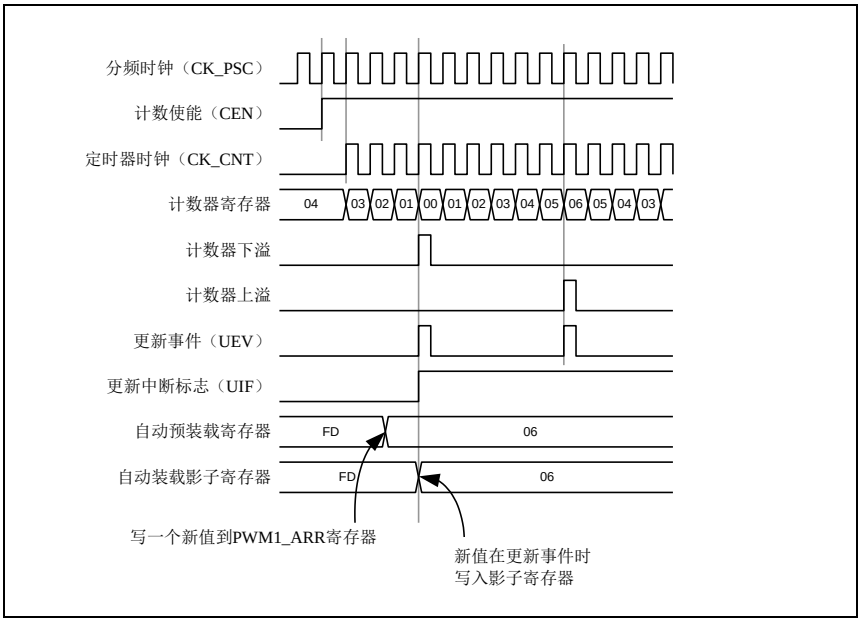
当发生更新事件时，所有的寄存器都被更新，硬件依据 URS 位更新标志位（PWMA_SR 寄存器中的 UIF 位）：

- 预分频器的缓存器被加载为预装载的值（PWMA_PSCR）。
- 当前的自动加载寄存器被更新为预装载值（PWMA_ARR）。

要注意到如果因为计数器溢出而产生更新，自动重装载寄存器将在计数器重载入之前被更新，因此下一个周期才是预期的值（计数器被装载为新的值）。

以下是一些计数器在不同时钟频率下的操作的例子：

内部时钟分频因子为 1，PWMA_ARR=0x6，ARPE＝1



使用中央对齐模式的提示：

- 启动中央对齐模式时，计数器将按照原有的向上/向下的配置计数。也就是说 PWMA_CR1 寄存器中的 DIR 位将决定计数器是向上还是向下计数。此外，软件不能同时修改 DIR 位和 CMS 位的值。
- 不推荐在中央对齐模式下，计数器正在计数时写计数器的值，这将导致不能预料的后果。具体的说：
  - 向计数器写入了比自动装载值更大的数值时（PWMA_CNT>PWMA_ARR），但计数器的计数方向不发生改变。例如计数器已经向上溢出，但计数器仍然向上计数。
  - 向计数器写入了 0 或者 PWMA_ARR 的值，但更新事件不发生。
- 安全使用中央对齐模式的计数器的方法是在启动计数器之前先用软件（置位 PWMA_EGR 寄存器的 UG 位）产生一个更新事件，并且不在计数器计数时修改计数器的值。

25.5.7 重复计数器

时基单元解释了计数器向上/向下溢出时更新事件（UEV）是如何产生的，然而事实上它只能在重复计数器的值达到 0 的时候产生。这个特性对产生 PWM 信号非常有用。

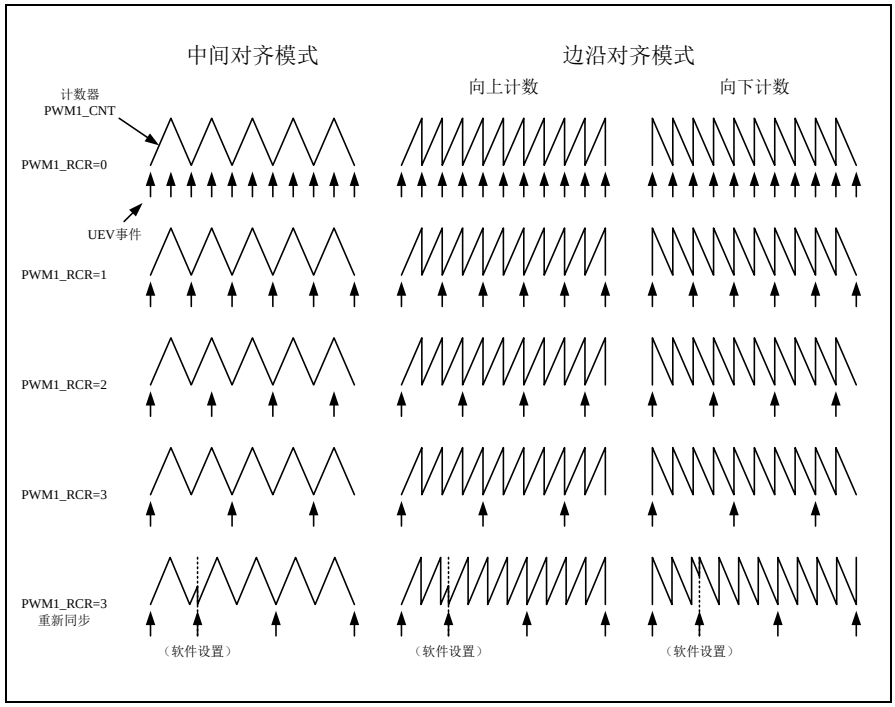
这意味着在每 N 次计数上溢或下溢时，数据从预装载寄存器传输到影子寄存器（PWMA_ARR 自动重载入寄存器，PWMA_PSCR 预装载寄存器，还有在比较模式下的捕获/比较寄存器 PWMA_CCRx），N 是 PWMA_RCR 重复计数寄存器中的值。

重复计数器在下述任一条件成立时递减：

- 向上计数模式下每次计数器向上溢出时
- 向下计数模式下每次计数器向下溢出时
- 中央对齐模式下每次上溢和每次下溢时。虽然这样限制了 PWM 的最大循环周期为 128，但它能够在每个 PWM 周期 2 次更新占空比。在中央对齐模式下，因为波形是对称的，如果每个 PWM 周期中仅刷新一次比较寄存器，则最大的分辨率为 $2*t_{CK\_PSC}$。

重复计数器是自动加载的，重复速率由 PWMA_RCR 寄存器的值定义。当更新事件由软件产生或者通过硬件的时钟/触发控制器产生，则无论重复计数器的值是多少，立即发生更新事件，并且 PWMA_RCR 寄存器中的内容被重载入到重复计数器。

不同模式下更新速率的例子，及 PWMA_RCR 的寄存器设置



25.6　时钟/触发控制器

时钟/触发控制器允许用户选择计数器的时钟源，输入触发信号和输出信号，

25.6.1　预分频时钟（CK_PSC）

时基单元的预分频时钟（CK_PSC）可以由以下源提供：

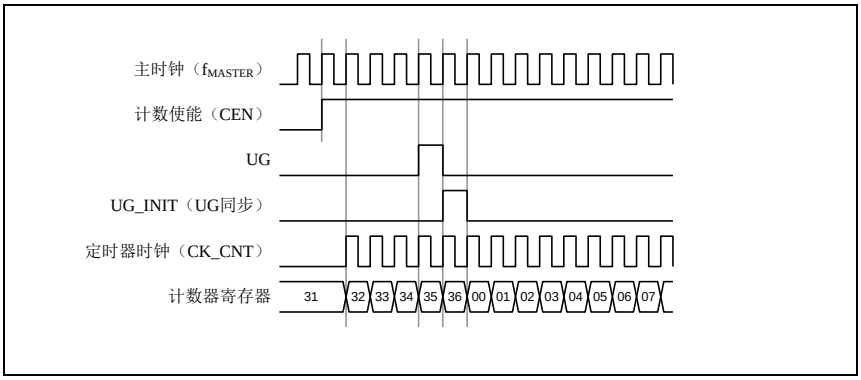
- 内部时钟（f_{MASTER}）
- 外部时钟模式 1：外部时钟输入（TIx）
- 外部时钟模式 2：外部触发输入 ETR
- 内部触发输入（ITRx）：使用一个 PWM 的 TRGO 做为另一个 PWM 的预分频时钟。

25.6.2　内部时钟源（f_{MASTER}）

如果同时禁止了时钟/触发模式控制器和外部触发输入（PWMA_SMCR 寄存器的 SMS=000，PWMA_ETR 寄存器的 ECE＝0），则 CEN、DIR 和 UG 位是实际上的控制位，并且只能被软件修改（UG 位仍被自动清除）。一旦 CEN 位被写成 1，预分频器的时钟就由内部时钟提供。

下图描述了控制电路和向上计数器在普通模式下，不带预分频器时的操作。

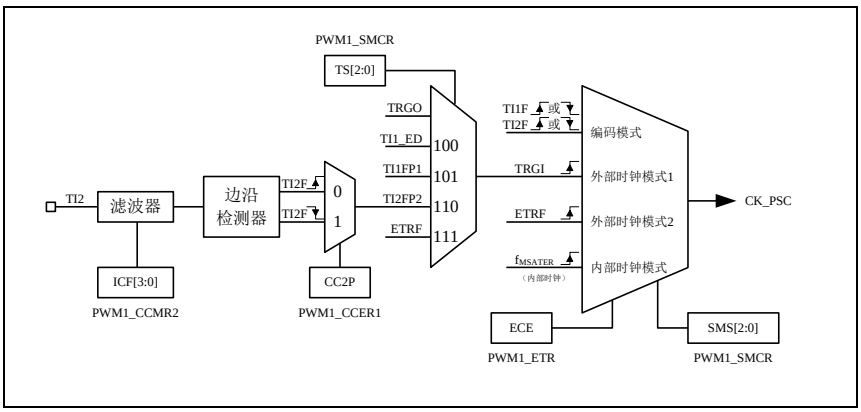
普通模式下的控制电路，f_{MASTER} 分频因子为 1



25.6.3 外部时钟源模式 1

当 PWMA_SMCR 寄存器的 SMS=111 时，此模式被选中。然后再通过 PWMA_SMCR 寄存器的 TS 选择 TRGI 的信号源。计数器可以在选定输入端的每个上升沿或下降沿计数。

下面的例子以 TI2 作为外部时钟



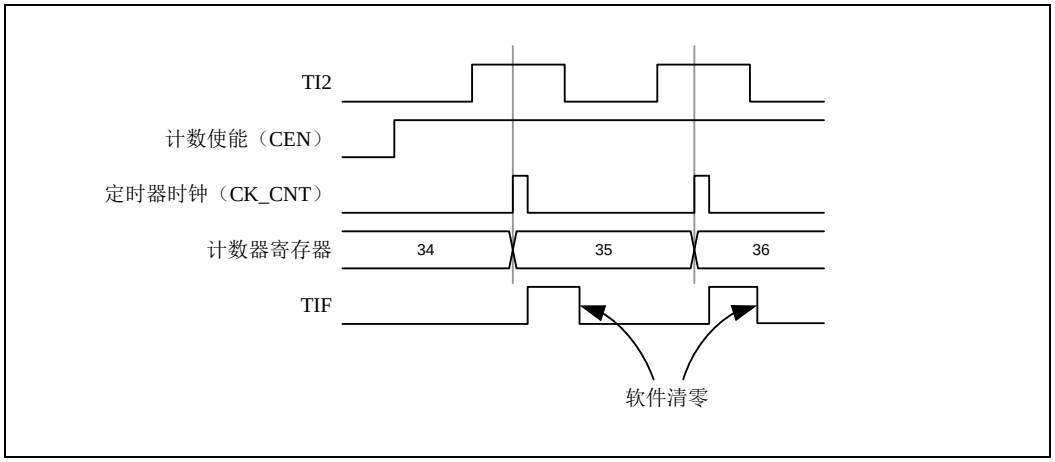
例如，要配置向上计数器在 TI2 输入端的上升沿计数，使用下列步骤：

1. 配置 PWMA_CCMR2 寄存器的 CC2S=01，使用通道 2 检测 TI2 输入
2. 配置 PWMA_CCMR2 寄存器的 IC2F[3:0]位，选择输入滤波器带宽
3. 配置 PWMA_CCER1 寄存器的 CC2P=0，选定上升沿极性
4. 配置 PWMA_SMCR 寄存器的 SMS=111，配置计数器使用外部时钟模式 1
5. 配置 PWMA_SMCR 寄存器的 TS=110，选定 TI2 作为输入源
6. 设置 PWMA_CR1 寄存器的 CEN=1，启动计数器

当上升沿出现在 TI2，计数器计数一次，且触发标识位（PWMA_SR1 寄存器的 TIF 位）被置 1，如果使能了中断（在 PWMA_IER 寄存器中配置）则会产生中断请求。

在 TI2 的上升沿和计数器实际时钟之间的延时取决于在 TI2 输入端的重新同步电路。

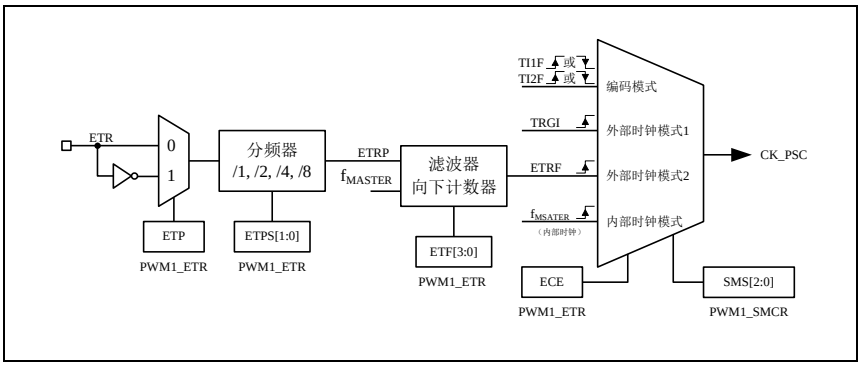
外部时钟模式 1 下的控制电路



25.6.4 外部时钟源模式 2

计数器能够在外部触发输入 ETR 信号的每一个上升沿或下降沿计数。将 PWMA_ETR 寄存器的 ECE 位写 1，即可选定此模式。（PWMA_SMCR 寄存器的 SMS=111 且 PWMA_SMCR 寄存器的 TS=111 时，也可选择此模式）

外部触发输入的总体框图：

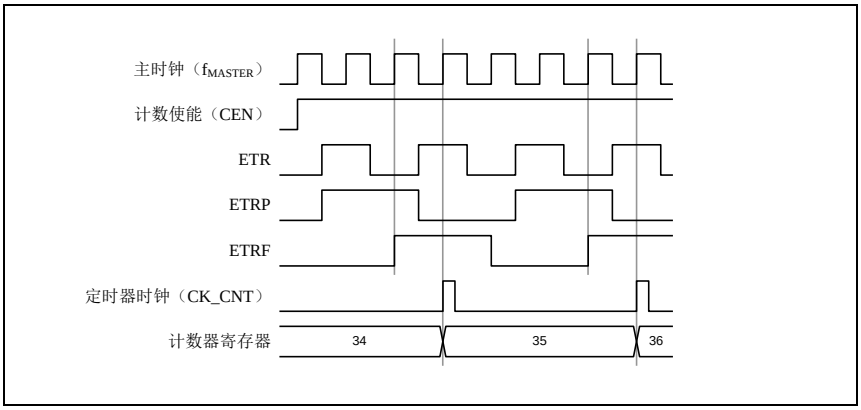


例如，要配置计数器在 ETR 信号的每 2 个上升沿时向上计数一次，需使用下列步骤：

1. 本例中不需要滤波器，配置 PWMA_ETR 寄存器的 ETF[3:0]=0000
2. 设置预分频器，配置 PWMA_ETR 寄存器的 ETPS[1:0]=01
3. 选择 ETR 的上升沿检测，配置 PWMA_ETR 寄存器的 ETP=0
4. 开启外部时钟模式 2，配置 PWMA_ETR 寄存器中的 ECE=1
5. 启动计数器，写 PWMA_CR1 寄存器的 CEN=1

计数器在每 2 个 ETR 上升沿计数一次。

外部时钟模式 2 下的控制电路



25.6.5　触发同步

PWMA 的计数器使用三种模式与外部的触发信号同步：

- 标准触发模式
- 复位触发模式
- 门控触发模式

标准触发模式

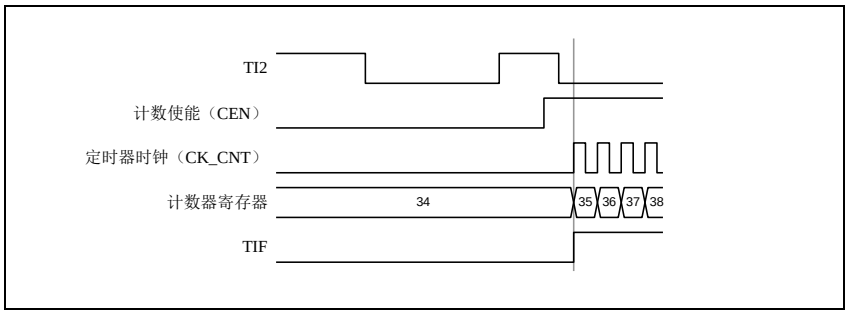
计数器的使能（CEN）依赖于选中的输入端上的事件。

在下面的例子中，计数器在 TI2 输入的上升沿开始向上计数：

1. 配置 PWMA_CCER1 寄存器的 CC2P＝0，选择 TI2 的上升沿做为触发条件。
2. 配置 PWMA_SMCR 寄存器的 SMS=110，选择计数器为触发模式。配置 PWMA_SMCR 寄存器的 TS=110，选择 TI2 作为输入源。

当 TI2 出现一个上升沿时，计数器开始在内部时钟驱动下计数，同时置位 TIF 标志。TI2 上升沿和计数器启动计数之间的延时取决于 TI2 输入端的重同步电路。

标准触发模式的控制电路



复位触发模式

在发生一个触发输入事件时，计数器和它的预分频器能够重新被初始化。同时，如果 PWMA_CR1 寄存器的 URS 位为低，还产生一个更新事件 UEV，然后所有的预装载寄存器（PWMA_ARR，PWMA_CCRx）都会被更新。

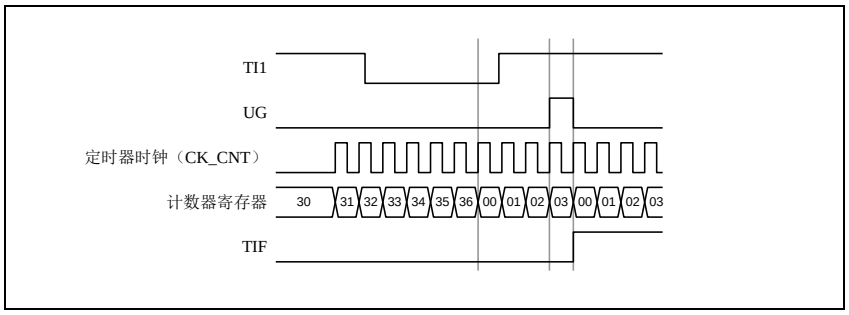
在以下的例子中，TI1 输入端的上升沿导致向上计数器被清零：

1. 配置 PWMA_CCER1 寄存器的 CC1P=0 来选择 TI1 的极性（只检测 TI1 的上升沿）。
2. 配置 PWMA_SMCR 寄存器的 SMS=100，选择定时器为复位触发模式。配置 PWMA_SMCR 寄存器的 TS=101，选择 TI1 作为输入源。
3. 配置 PWMA_CR1 寄存器的 CEN=1，启动计数器。

计数器开始依据内部时钟计数，然后正常计数直到 TI1 出现一个上升沿。此时，计数器被清零然后从 0 重新开始计数。同时，触发标志（PWMA_SR1 寄存器的 TIF 位）被置位，如果使能了中断（PWMA_IER 寄存器的 TIE 位），则产生一个中断请求。

下图显示当自动重装载寄存器 PWMA_ARR=0x36 时的动作。在 TI1 上升沿和计数器的实际复位之间的延时取决于 TI1 输入端的重同步电路。

复位触发模式下的控制电路



门控触发模式

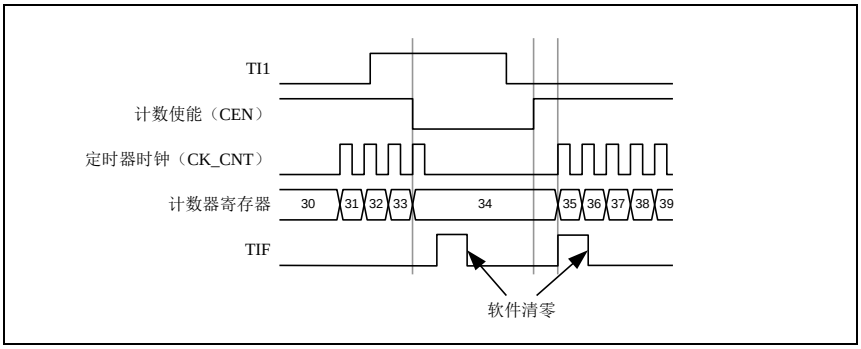
计数器由选中的输入端信号的电平使能。

在如下的例子中，计数器只在 TI1 为低时向上计数：

1. 配置 PWMA_CCER1 寄存器的 CC1P=1 来确定 TI1 的极性（只检测 TI1 上的低电平）。
2. 配置 PWMA_SMCR 寄存器的 SMS=101，选择定时器为门控触发模式，配置 PWMA_SMCR 寄存器中 TS=101，选择 TI1 作为输入源。
3. 配置 PWMA_CR1 寄存器的 CEN=1，启动计数器（在门控模式下，如果 CEN=0，则计数器不能启动，不论触发输入电平如何）。

只要 TI1 为低，计数器开始依据内部时钟计数，一旦 TI1 变高则停止计数。当计数器开始或停止时 TIF 标志位都会被置位。TI1 上升沿和计数器实际停止之间的延时取决于 TI1 输入端的重同步电路。

门控触发模式下的控制电路



外部时钟模式 2 联合触发模式

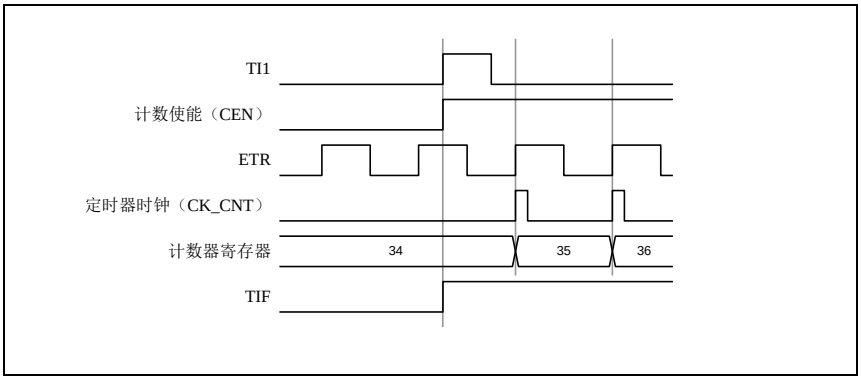
外部时钟模式 2 可以与另一个输入信号的触发模式一起使用。例如，ETR 信号被用作外部时钟的输入，另一个输入信号可用作触发输入（支持标准触发模式，复位触发模式和门控触发模式）。注意不能通过 PWMA_SMCR 寄存器的 TS 位把 ETR 配置成 TRGI。

在下面的例子中，一旦在 TI1 上出现一个上升沿，计数器即在 ETR 的每一个上升沿向上计数一次：

1. 通过 PWMA_ETR 寄存器配置外部触发输入电路。配置 ETPS=00 禁止预分频，配置 ETP=0 监测 ETR 信号的上升沿，配置 ECE=1 使能外部时钟模式 2。
2. 配置 PWMA_CCER1 寄存器的 CC1P=0 来选择 TI1 的上升沿触发。
3. 配置 PWMA_SMCR 寄存器的 SMS=110 来选择定时器为触发模式。配置 PWMA_SMCR 寄存器的 TS=101 来选择 TI1 作为输入源。

当 TI1 上出现一个上升沿时，TIF 标志被设置，计数器开始在 ETR 的上升沿计数。TI1 信号的上升沿和计数器实际时钟之间的延时取决于 TI1 输入端的重同步电路。ETR 信号的上升沿和计数器实际时钟之间的延时取决于 ETRP 输入端的重同步电路。

外部时钟模式 2+触发模式下的控制电路



25.6.6 与 PWMB 同步

在芯片中，定时器在内部互相联结，用于定时器的同步或链接。当某个定时器配置成主模式时，可以输出触发信号（TRGO）到那些配置为从模式的定时器来完成复位操作、启动操作、停止操作或者作为那些定时器的驱动时钟。

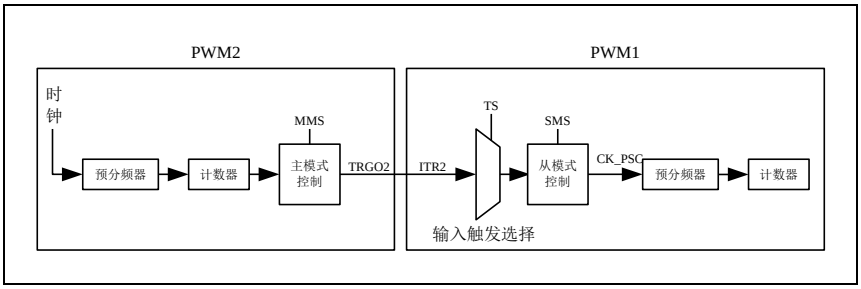
使用 PWMB 的 TRGO 作为 PWMA 的预分频时钟

例如，用户可以配置 PWMB 作为 PWMA 的预分频时钟，需进行如下配置：

1. 配置 PWMB 为主模式，使得在每个更新事件（UEV）时输出周期性的触发信号。配置 PWMB_CR2 寄存器的 MMS=010，使每个更新事件时 TRGO 能输出一个上升沿。
2. PWMB 输出的 TRGO 信号链接到 PWMA。PWMA 需要配置成触发从模式，使用 ITR2 作为输入触发信号。以上操作可以通过配置 PWMA_SMCR 寄存器的 TS=010 实现。

3. 配置 PWMA_SMCR 寄存器的 SMS=111 将时钟/触发控制器设置为外部时钟模式 1。此操作将使 PWMB 输出的周期性触发信号 TRGO 的上升沿驱动 PWMA 的时钟。
4. 最后，置位 PWMB 的 CEN 位（PWMB_CR1 寄存器中），使能两个 PWM。

主触发从模式的例子



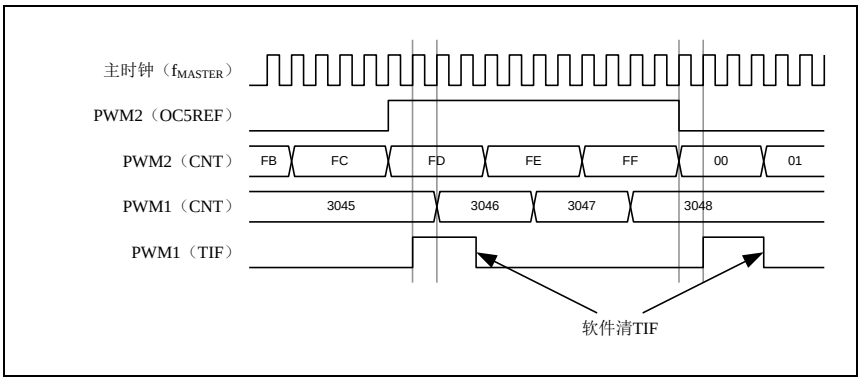
使用 PWMB 使能 PWMA

在本例中，我们用 PWMB 的比较输出使能 PWMA。PWMA 仅在 PWMB 的 OC1REF 信号为高时按照自己的驱动时钟计数。两个 PWM 都使用 4 分频的 f_{MASTER} 为时钟（$f_{CK\_CNT} = f_{MASTER}/4$）。

1. 配置 PWMB 为主模式，将比较输出信号（OC5REF）作为触发信号输出。（配置 PWMB_CR2 寄存器的 MMS=100）。
2. 配置 PWMB 的 OC5REF 信号的波形（PWMB_CCMR1 寄存器）。
3. 配置 PWMA 把 PWMB 的输出作为自己的触发输入信号（配置 PWMA_SMCR 寄存器的 TS=010）。
4. 配置 PWMA 为门控触发模式（配置 PWMA_SMCR 寄存器的 SMS=101）。
5. 置位 CEN 位（PWMA_CR1 寄存器），使能 PWMA。
6. 置位 CEN 位（PWMB_CR1 寄存器），使能 PWMB。

注意：两个 PWM 的时钟并不同步，但仅影响 PWMA 的使能信号。

PWMB 的输出门控触发 PWMA

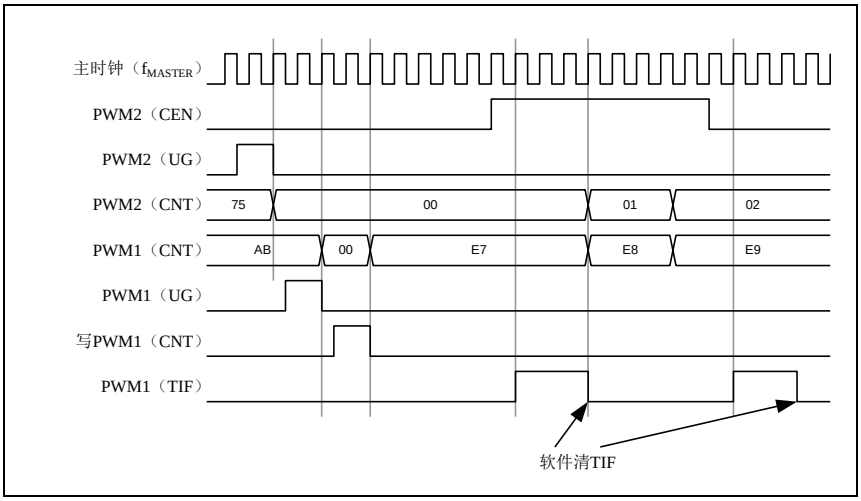


上图中，PWMA 的计数器和预分频器都没有在启动前初始化，所以都是从现有值开始计数的。如果在启动 PWMB 之前复位两个定时器，用户就可以写入期望的数值到 PWMA 的计数器，使之从指定值开始计数。对 PWMA 的复位操作可以通过软件写 PWMA_EGR 寄存器的 UG 位实现。

在下面这个例子中，我们使 PWMB 和 PWMA 同步。PWMB 为主模式并从 0 启动计数。PWMA 为触发从模式，并从 0xE7 启动计数。两个 PWM 采用相同的分频系数。当清除 PWMB_CR1 寄存器的 CEN 位时，PWMB 被禁止，同时 PWMA 停止计数。

1. 配置 PWMB 为主模式，将比较输出信号（OC5REF）作为触发信号输出。（配置 PWMB_CR2 寄存器的 MMS=100）。
2. 配置 PWMB 的 OC5REF 信号的波形（PWMB_CCMR1 寄存器）。
3. 配置 PWMA 把 PWMB 的输出作为自己的触发输入信号（配置 PWMA_SMCR 寄存器的 TS=010）。
4. 配置 PWMA 为门控触发模式（配置 PWMA_SMCR 寄存器的 SMS=101）。

5. 通过对 UG 位（PWMB_EGR 寄存器）写 1，复位 PWMB。
6. 通过对 UG 位（PWMA_EGR 寄存器）写 1，复位 PWMA。
7. 将 0xE7 写入 PWMA 的计数器中（PWMA_CNTRL），初始化 PWMA。
8. 通过对 CEN 位（PWMA_CR1 寄存器）写 1，使能 PWMA。
9. 通过对 CEN 位（PWMB_CR1 寄存器）写 1，启动 PWMB。
10. 通过对 CEN 位（PWMB_CR1 寄存器）写 0，停止 PWMB。



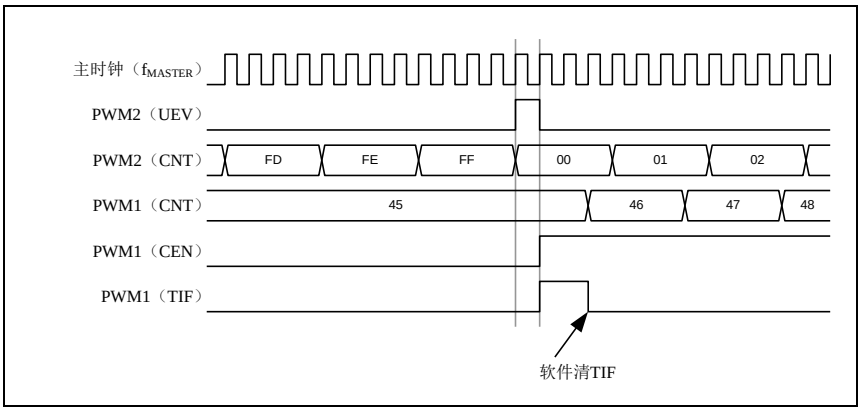
使用 PWMB 启动 PWMA

在本例中，我们用 PWMB 的更新事件来启动 PWMA。

PWMA 在 PWMB 发生更新事件时按照 PWMA 自己的驱动时钟从它的现有值开始计数（可以是非 0 值）。PWMA 在收到触发信号后自动使能 CEN 位，并开始计数，一直持续到用户向 PWMA_CR1 寄存器的 CEN 位写 0。两个 PWM 都使用 4 分频的 f_{MASTER} 作为驱动时钟（$f_{CK\_CNT} = f_{MASTER}/4$）。

1. 配置 PWMB 为主模式，输出更新信号（UEV）。（配置 PWMB_CR2 寄存器的 MMS=010）。
2. 配置 PWMB 的周期（PWMB_ARR 寄存器）。
3. 配置 PWMA 用 PWMB 的输出作为输入的触发信号（配置 PWMA_SMCR 寄存器的 TS=010）。
4. 配置 PWMA 为触发模式（配置 PWMA_SMCR 寄存器的 SMS=110）。
5. 置位 CEN 位（PWMB_CR1 寄存器）启动 PWMB。

PWMB 的更新事件（PWMB-UEV）触发 PWMA



如同前面的例子，用户也可以在启动计数器前对它们初始化。

用外部信号同步的触发两个 PWM

在本例中，使用 TI1 的上升沿使能 PWMB，并同时使能 PWMA。为了保持定时器的对齐，PWMB 需要配置成主/从模式（对于 TI1 信号为从模式，对于 PWMA 为主模式）。

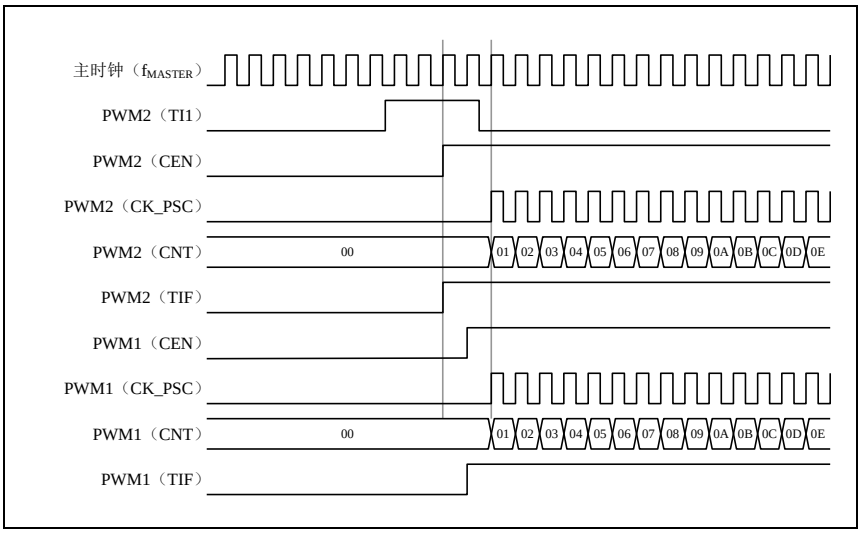
1. 配置 PWMB 为主模式，以输出使能信号作为 PWMA 的触发（配置 PWMB_CR2 寄存器的 MMS=001）。

2. 配置 PWMB 为从模式，把 TI1 信号作为输入的触发信号（配置 PWMB_SMCR 寄存器的 TS=100）。
3. 配置 PWMB 的触发模式（配置 PWMB_SMCR 寄存器的 SMS=110）。
4. 配置 PWMB 为主/从模式（配置 PWMB_SMCR 寄存器的 MSM=1）。
5. 配置 PWMA 以 PWMB 的输出为输入触发信号（配置 PWMA_SMCR 寄存器的 TS=010）。
6. 配置 PWMA 的触发模式（配置 PWMA_SMCR 寄存器的 SMS=110）。

当 TI1 上出现上升沿时，两个定时器同步的开始计数，并且 TIF 位都被置起。

注意：在本例中，两个定时器在启动前都进行了初始化（设置 UG 位），所以它们都从 0 开始计数，但是用户也可以通过修改计数器寄存器（PWMA_CNT）来插入一个偏移量，这样的话，在 PWMB 的 CK_PSC 信号和 CNT_EN 信号间会插入延时。

PWMB 的 TI1 信号触发 PWMB 和 PWMA

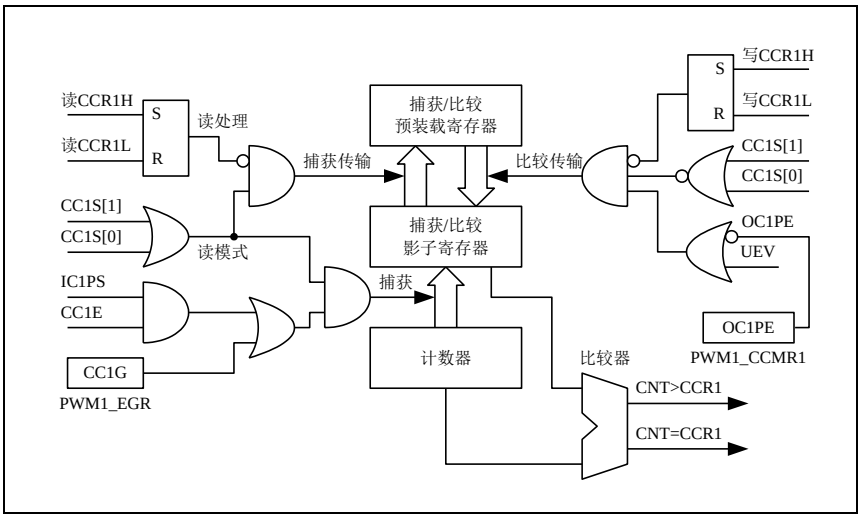


25.7 捕获/比较通道

PWM1P、PWM2P、PWM3P、PWM4P 可以用作输入捕获，PWM1P/PWM1N、PWM2P/PWM2N、PWM3P/PWM3N、PWM4P/PWM4N 可以输出比较，这个功能可以通过配置捕获/比较通道模式寄存器（PWMA_CCMRi）的 CCiS 通道选择位来实现，此处的 i 代表 1~4 的通道数。

每一个捕获/比较通道都是围绕着一个捕获/比较寄存器（包含影子寄存器）来构建的，包括捕获的输入部分（数字滤波、多路复用和预分频器）和输出部分（比较器和输出控制）。

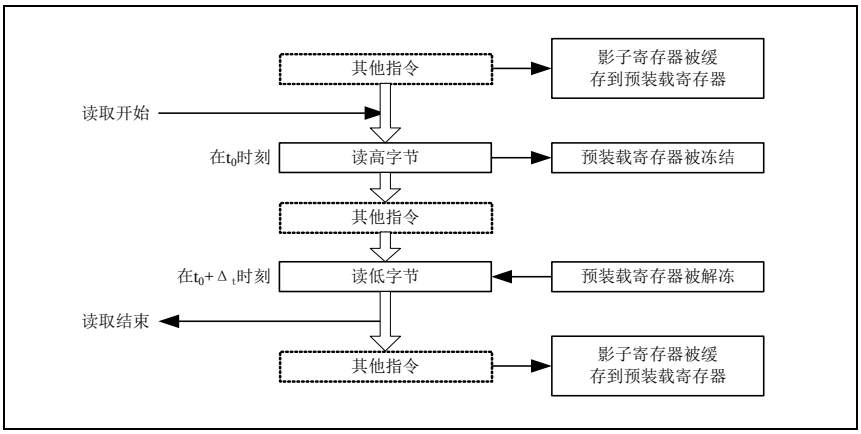
捕获/比较通道 1 的主要电路（其他通道与此类似）



捕获/比较模块由一个预装载寄存器和一个影子寄存器组成。读写过程仅操作预装载寄存器。在捕获模式下，捕获发生在影子寄存器上，然后再复制到预装载寄存器中。在比较模式下，预装载寄存器的内容被复制到影子寄存器中，然后影子寄存器的内容和计数器进行比较。

当通道被配置成输出模式时，可以随时访问 PWMA_CCRi 寄存器。

当通道被配置成输入模式时，对 PWMA_CCRi 寄存器的读操作类似于计数器的读操作。当捕获发生时，计数器的内容被捕获到 PWMA_CCRi 影子寄存器，随后再复制到预装载寄存器中。在读操作进行中，预装载寄存器是被冻结的。



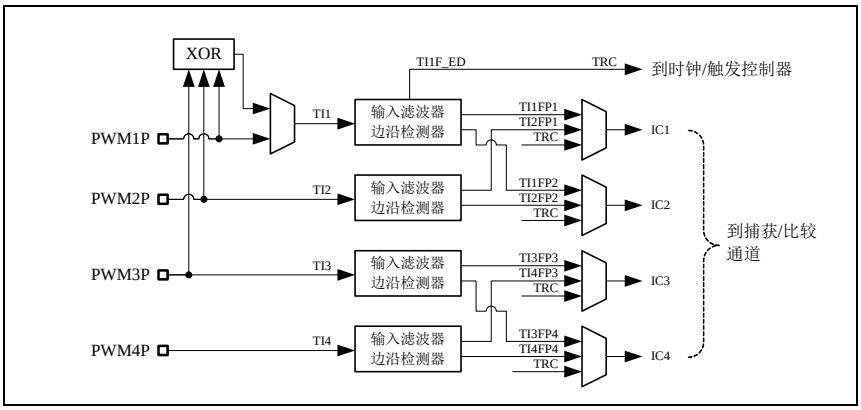
上图描述了 16 位的 CCRi 寄存器的读操作流程，被缓存的数据将保持不变直到读流程结束。在整个读流程结束后，如果仅仅读了 PWMA_CCRiL 寄存器，返回计数器数值的低位。如果在读了低位数据以后再读高位数据，将不再返回同样的低位数据。

25.7.1 16 位 PWMA_CCRi 寄存器的写流程

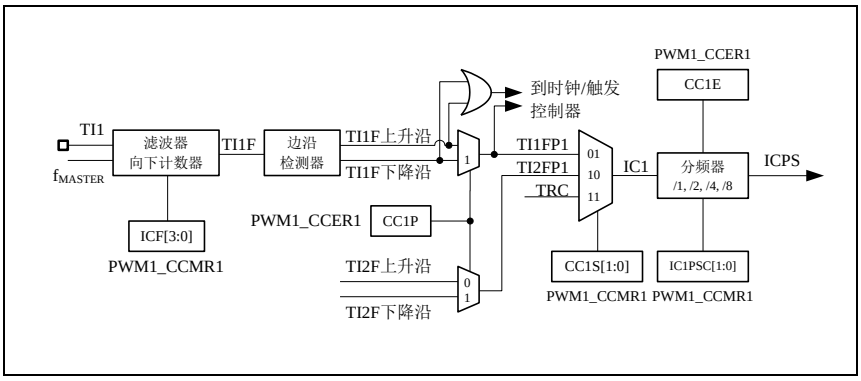
16 位 PWMA_CCRi 寄存器的写操作通过预装载寄存器完成。必需使用两条指令来完成整个流程，一条指令对应一个字节。必需先写高位字节。在写高位字节时，影子寄存器的更新被禁止直到低位字节的写操作完成。

25.7.2 输入模块

输入模块的框图



如图，输入部分对相应的 TIx 输入信号采样，并产生一个滤波后的信号 TIxF。然后，一个带极性选择的边缘监测器产生一个信号（TIxFPx），它可以作为触发模式控制器的输入触发或者作为捕获控制。该信号通过预分频后进入捕获寄存器（ICxPS）。



25.7.3 输入捕获模式

在输入捕获模式下，当检测到 ICi 信号上相应的边沿后，计数器的当前值被锁存到捕获/比较寄存器（PWMA_CCRx）中。当发生捕获事件时，相应的 CCiIF 标志（PWMA_SR 寄存器）被置 1。如果 PWMA_IER 寄存器的 CCiIE 位被置位，也就是使能了中断，则将产生中断请求。如果发生捕获事件时 CCiIF 标志已经为高，那么重复捕获标志 CCiOF（PWMA_SR2 寄存器）被置 1。写 CCiIF=0 或读取存储在 PWMA_CCRiL 寄存器中的捕获数据都可清除 CCiIF。写 CCiOF=0 可清除 CCiOF。

PWM 输入信号上升沿时捕获

以下例子说明如何在 TI1 输入的上升沿时捕获计数器的值到 PWMA_CCR1 寄存器中，步骤如下：

1. 选择有效输入端，设置 PWMA_CCMR1 寄存器中的 CC1S=01，此时通道被配置为输入，并且 PWMA_CCR1 寄存器变为只读。
2. 根据输入信号 TIi 的特点，可通过配置 PWMA_CCMR1 寄存器中的 IC1F 位来设置相应的输入滤波器的滤波时间。假设输入信号在最多 5 个时钟周期的时间内抖动，我们须配置滤波器的带宽长于 5 个时钟周期；因此我们可以连续采样 8 次，以确认在 TI1 上一次真实的边沿变换，即在 PWMA_CCMR1 寄存器中写入 IC1F=0011，此时，只有连续采样到 8 个相同的 TI1 信号，信号才为有效（采样频率为 f_{MASTER}）。
3. 选择 TI1 通道的有效转换边沿，在 PWMA_CCER1 寄存器中写入 CC1P=0（上升沿）。
4. 配置输入预分频器。在本例中，我们希望捕获发生在每一个有效的电平转换时刻，因此预分频器被禁止（写 PWMA_CCMR1 寄存器的 IC1PS=00）。
5. 设置 PWMA_CCER1 寄存器的 CC1E=1，允许捕获计数器的值到捕获寄存器中。
6. 如果需要，通过设置 PWMA_IER 寄存器中的 CC1IE 位允许相关中断请求。

当发生一个输入捕获时：

- 当产生有效的电平转换时，计数器的值被传送到 PWMA_CCR1 寄存器。
- CC1IF 标志被设置。当发生至少 2 个连续的捕获时，而 CC1IF 未曾被清除时，CC1OF 也被置 1。
- 如设置了 CC1IE 位，则会产生一个中断。

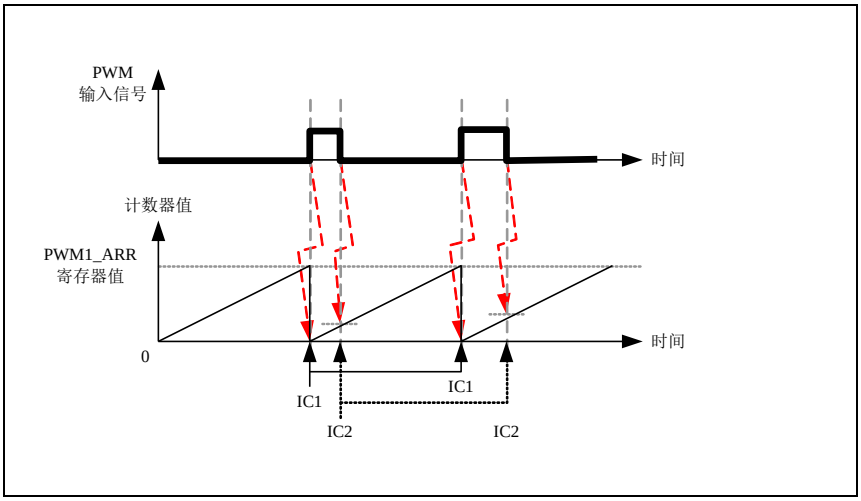
为了处理捕获溢出事件（CC1OF 位），建议在读出重复捕获标志之前读取数据，这是为了避免丢失在读出捕获溢出标志之后和读取数据之前可能产生的重复捕获信息。

注意：设置 PWMA_EGR 寄存器中相应的 CCiG 位，可以通过软件产生输入捕获中断。

PWM 输入信号测量

该模式是输入捕获模式的一个特例，除下列区别外，操作与输入捕获模式相同：

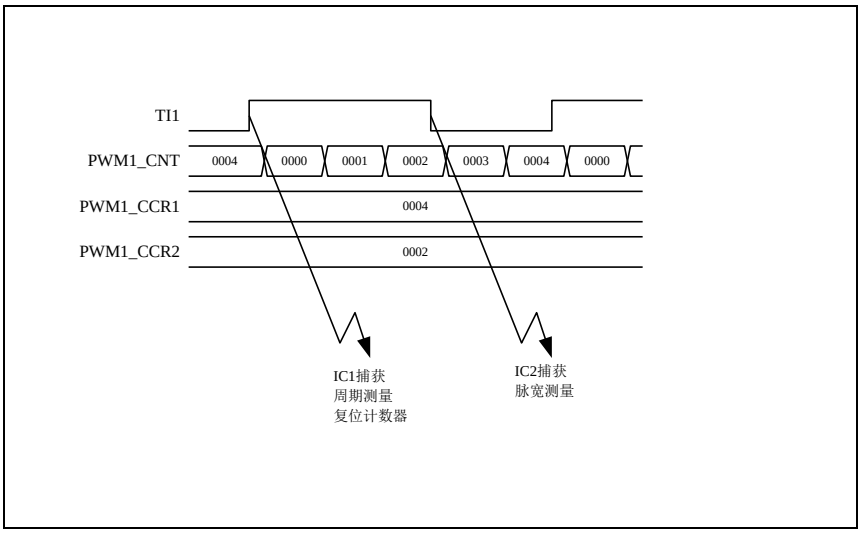
- 两个 ICi 信号被映射至同一个 TIi 输入。
- 这两个 ICi 信号的有效边沿的极性相反。
- 其中一个 TIiFP 信号被作为触发输入信号，而触发模式控制器被配置成复位触发模式。



例如，你可以用以下方式测量 TI1 上输入的 PWM 信号的周期（PWMA_CCR1 寄存器）和占空比（PWMA_CCR2 寄存器）。

1. 选择 PWMA_CCR1 的有效输入：置 PWMA_CCMR1 寄存器的 CC1S=01（选中 TI1FP1）。
2. 选择 TI1FP1 的有效极性：置 CC1P=0（上升沿有效）。
3. 选择 PWMA_CCR2 的有效输入：置 PWMA_CCMR2 寄存器的 CC2S=10（选中 TI1FP2）。
4. 选择 TI1FP2 的有效极性（捕获数据到 PWMA_CCR2）：置 CC2P=1（下降沿有效）。
5. 选择有效的触发输入信号：置 PWMA_SMCR 寄存器中的 TS=101（选择 TI1FP1）。
6. 配置触发模式控制器为复位触发模式：置 PWMA_SMCR 中的 SMS=100。
7. 使能捕获：置 PWMA_CCER1 寄存器中 CC1E=1，CC2E=1。

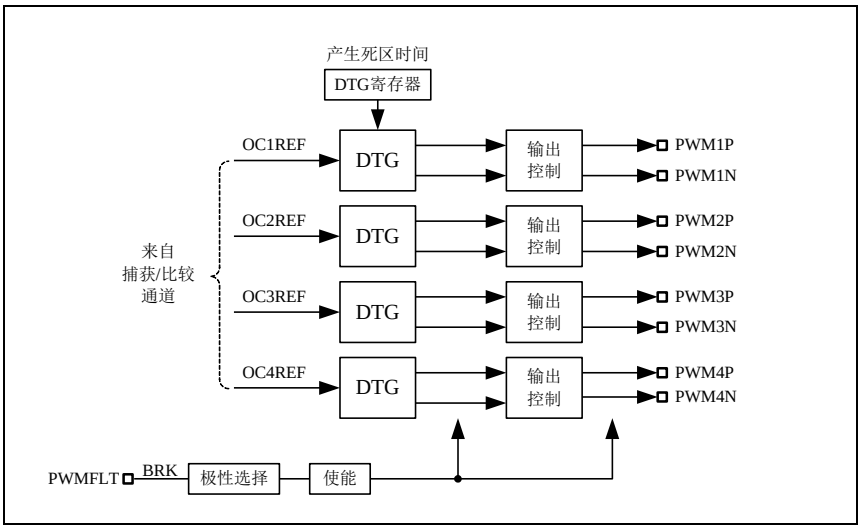
PWM 输入信号测量实例



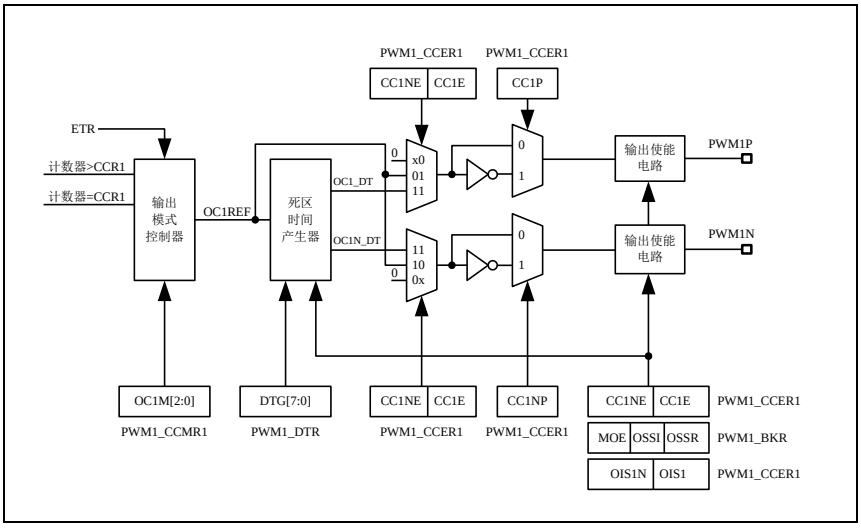
25.7.4　输出模块

输出模块会产生一个用来做参考的中间波形，称为 OCiREF（高有效）。刹车功能和极性的处理都在模块的最后处理。

输出模块框图



通道 1 详细的带互补输出的输出模块框图（其他通道类似）



25.7.5 强制输出模式

在输出模式下，输出比较信号能够直接由软件强制为高或低状态，而不依赖于输出比较寄存器和计数器间的比较结果。

置 PWMA_CCMRi 寄存器的 OCiM=101，可强制 OCiREF 信号为高。

置 PWMA_CCMRi 寄存器的 OCiM=100，可强制 OCiREF 信号为低。

OCi/OCiN 的输出是高还是低则取决于 CCiP/CCiNP 极性标志位。

该模式下，在 PWMA_CCRi 影子寄存器和计数器之间的比较仍然在进行，相应的标志也会被修改，也仍然会产生相应的中断。

25.7.6 输出比较模式

此模式用来控制一个输出波形或者指示一段给定的时间已经达到。

当计数器与捕获/比较寄存器的内容相匹配时，有如下操作：

- 根据不同的输出比较模式，相应的 OCi 输出信号：
  - 保持不变（OCiM=000）
  - 设置为有效电平（OCiM=001）
  - 设置为无效电平（OCiM=010）

— 翻转（OCiM=011）

- 设置中断状态寄存器中的标志位（PWMA_SR1 寄存器中的 CCiIF 位）。
- 若设置了相应的中断使能位（PWMA_IER 寄存器中的 CCiIE 位），则产生一个中断。

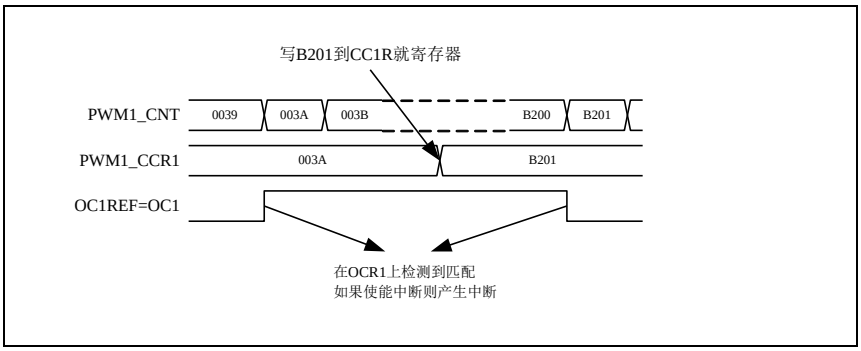
PWMA_CCMRi 寄存器的 OCiM 位用于选择输出比较模式，而 PWMA_CCMRi 寄存器的 CCiP 位用于选择有效和无效的电平极性。PWMA_CCMRi 寄存器的 OCiPE 位用于选择 PWMA_CCRi 寄存器是否需要使用预装载寄存器。在输出比较模式下，更新事件 UEV 对 OCiREF 和 OCi 输出没有影响。时间精度为计数器的一个计数周期。输出比较模式也能用来输出一个单脉冲。

输出比较模式的配置步骤：

1. 选择计数器时钟（内部、外部或者预分频器）。
2. 将相应的数据写入 PWMA_ARR 和 PWMA_CCRi 寄存器中。
3. 如果要产生一个中断请求，设置 CCiIE 位。
4. 选择输出模式步骤：
   1. 设置 OCiM=011，在计数器与 CCRi 匹配时翻转 OCiM 管脚的输出
   2. 设置 OCiPE = 0，禁用预装载寄存器
   3. 设置 CCiP = 0，选择高电平为有效电平
   4. 设置 CCiE = 1，使能输出
   5. 设置 PWMA_CR1 寄存器的 CEN 位来启动计数器

PWMA_CCRi 寄存器能够在任何时候通过软件进行更新以控制输出波形，条件是未使用预装载寄存器（OCiPE=0），否则 PWMA_CCRi 的影子寄存器只能在发生下一次更新事件时被更新。

输出比较模式，翻转 OC1



25.7.7 PWM 模式

脉冲宽度调制（PWM）模式可以产生一个由 PWMA_ARR 寄存器确定频率，由 PWMA_CCRi 寄存器确定占空比的信号。

在 PWMA_CCMRi 寄存器中的 OCiM 位写入 110（PWM 模式 1）或 111（PWM 模式 2），能够独立地设置每个 OCi 输出通道产生一路 PWM。必须设置 PWMA_CCMRi 寄存器的 OCiPE 位使能相应的预装载寄存器，也可以设置 PWMA_CR1 寄存器的 ARPE 位使能自动重装载的预装载寄存器（在向上计数模式或中央对称模式中）。

由于仅当发生一个更新事件的时候，预装载寄存器才能被传送到影子寄存器，因此在计数器开始计数之前，必须通过设置 PWMA_EGR 寄存器的 UG 位来初始化所有的寄存器。

OCi 的极性可以通过软件在 PWMA_CCERi 寄存器中的 CCiP 位设置，它可以设置为高电平有效或低电平有效。OCi 的输出使能通过 PWMA_CCERi 和 PWMA_BKR 寄存器中的 CCiE、MOE、OISi、OSSR 和 OSSI 位的组合来控制。

在 PWM 模式（模式 1 或模式 2）下，PWMA_CNT 和 PWMA_CCRi 始终在进行比较，（依据计数器的计数方向）以确定是否符合 PWMA_CCRi≤PWMA_CNT 或者 PWMA_CNT≤PWMA_CCRi。

根据 PWMA_CR1 寄存器中 CMS 位域的状态，定时器能够产生边沿对齐的 PWM 信号或中央对齐的

PWM 信号。

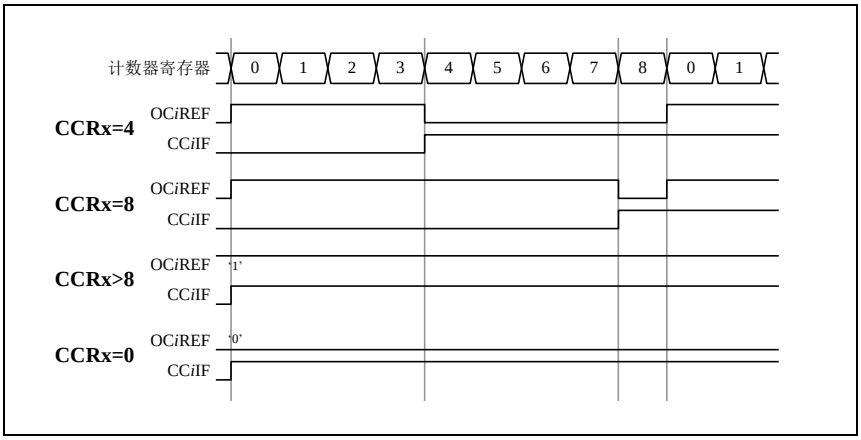
PWM 边沿对齐模式

向上计数配置

当 PWMA_CR1 寄存器中的 DIR 位为 0 时，执行向上计数。

下面是一个 PWM 模式 1 的例子。当 PWMA_CNT<PWMA_CCRi 时，PWM 参考信号 OCiREF 为高，否则为低。如果 PWMA_CCRi 中的比较值大于自动重装载值（PWMA_ARR），则 OCiREF 保持为高。如果比较值为 0，则 OCiREF 保持为低。

边沿对齐，PWM 模式 1 的波形（PWMA_ARR=8）



向下计数的配置

当 PWMA_CR1 寄存器的 DIR 位为 1 时，执行向下计数。

在 PWM 模式 1 时，当 PWMA_CNT>PWMA_CCRi 时参考信号 OCiREF 为低，否则为高。如果 PWMA_CCRi 中的比较值大于 PWMA_ARR 中的自动重装载值，则 OCiREF 保持为高。该模式下不能产生占空比为 0%的 PWM 波形。

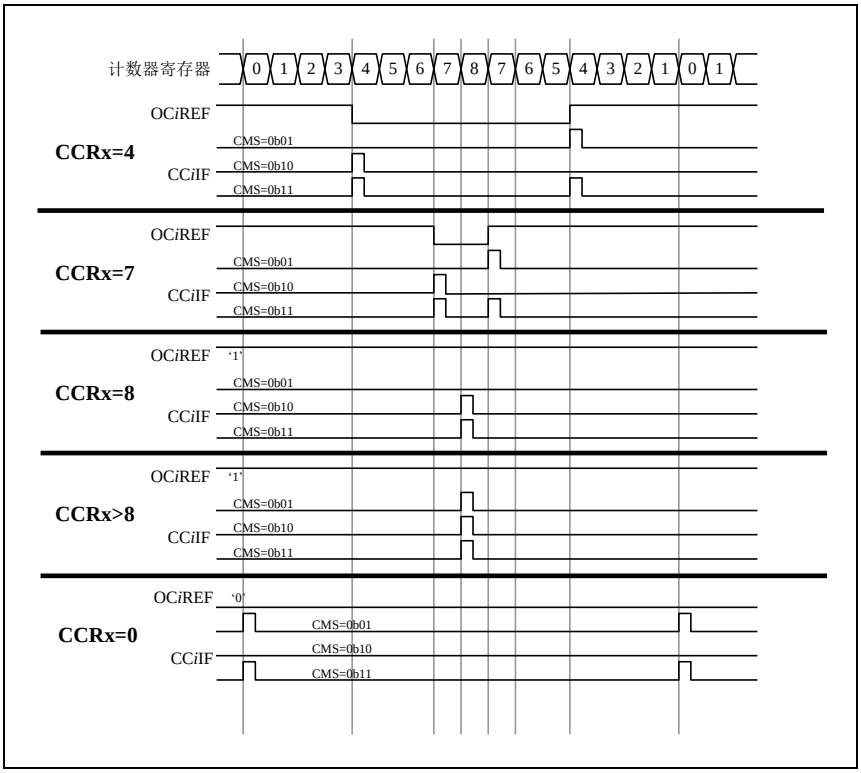
PWM 中央对齐模式

当 PWMA_CR1 寄存器中的 CMS 位不为’00’时为中央对齐模式（所有其他的配置对 OCiREF/OCi 信号都有相同的作用）。

根据不同的 CMS 位的设置，比较标志可以在计数器向上计数，向下计数，或向上和向下计数时被置 1。PWMA_CR1 寄存器中的计数方向位（DIR）由硬件更新，不要用软件修改它。

下面给出了一些中央对齐的 PWM 波形的例子：

- PWMA_ARR=8
- PWM 模式 1
- 标志位在以下三种情况下被置位：
  - 只有在计数器向下计数时（CMS=01）
  - 只有在计数器向上计数时（CMS=10）
  - 在计数器向上和向下计数时（CMS=11）
- 中央对齐的 PWM 波形（PWMA_ARR=8）



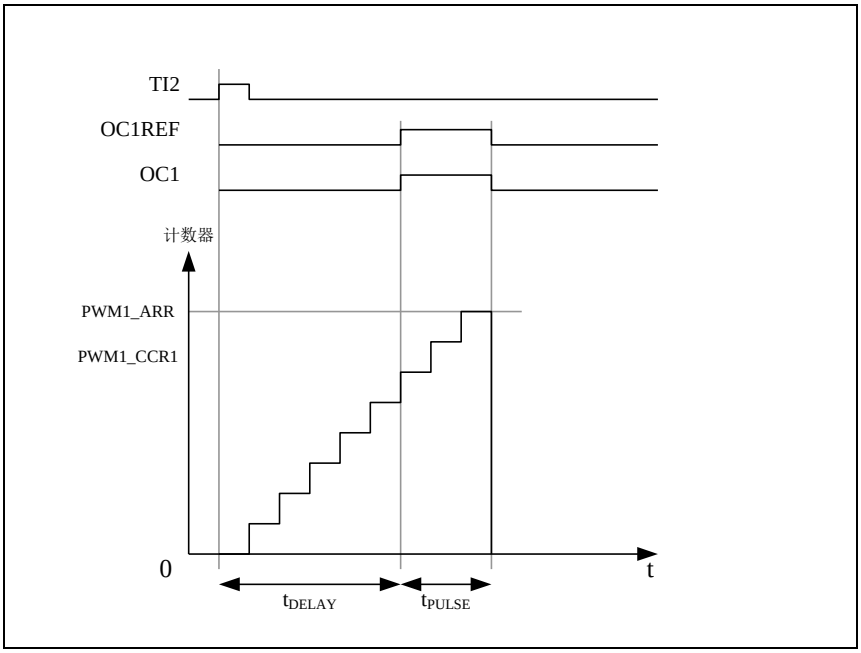
单脉冲模式

单脉冲模式（OPM）是前述众多模式的一个特例。这种模式允许计数器响应一个激励，并在一个程序可控的延时之后产生一个脉宽可控的脉冲。

可以通过时钟/触发控制器启动计数器，在输出比较模式或者 PWM 模式下产生波形。设置 PWMA_CR1 寄存器的 OPM 位将选择单脉冲模式，此时计数器自动地在下一个更新事件 UEV 时停止。仅当比较值与计数器的初始值不同时，才能产生一个脉冲。启动之前（当定时器正在等待触发），必须如下配置：

- 向上计数方式：计数器 CNT < CCRi ≤ ARR，
- 向下计数方式：计数器 CNT > CCRi。

单脉冲模式图例



例如，在从 TI2 输入脚上检测到一个上升沿之后延迟 t_{DELAY}，在 OC1 上产生一个 t_{PULSE} 宽度的正脉冲：（假定 IC2 作为触发 1 通道的触发源）

- 置 PWMA_CCMR2 寄存器的 CC2S=01，把 IC2 映射到 TI2。
- 置 PWMA_CCER1 寄存器的 CC2P=0，使 IC2 能够检测上升沿。
- 置 PWMA_SMCR 寄存器的 TS=110，使 IC2 作为时钟/触发控制器的触发源（TRGI）。
- 置 PWMA_SMCR 寄存器的 SMS=110（触发模式），IC2 被用来启动计数器。OPM 的波形由写入比较寄存器的数值决定（要考虑时钟频率和计数器预分频器）。
- t_{DELAY} 由 PWMA_CCR1 寄存器中的值定义。
- t_{PULSE} 由自动装载值和比较值之间的差值定义（PWMA_ARR – PWMA_CCR1）。
- 假定当发生比较匹配时要产生从 0 到 1 的波形，当计数器达到预装载值时要产生一个从 1 到 0 的波形，首先要置 PWMA_CCMR1 寄存器的 OCiM=111，进入 PWM 模式 2，根据需要有选择的设置 PWMA_CCMR1 寄存器的 OC1PE=1，置位 PWMA_CR1 寄存器中的 ARPE，使能预装载寄存器，然后在 PWMA_CCR1 寄存器中填写比较值，在 PWMA_ARR 寄存器中填写自动装载值，设置 UG 位来产生一个更新事件，然后等待在 TI2 上的一个外部触发事件。

在这个例子中，PWMA_CR1 寄存器中的 DIR 和 CMS 位应该置低。

因为只需要一个脉冲，所以设置 PWMA_CR1 寄存器中的 OPM=1，在下一个更新事件（当计数器从自动装载值翻转到 0）时停止计数。

OCx 快速使能（特殊情况）

在单脉冲模式下，对 TIi 输入脚的边沿检测会设置 CEN 位以启动计数器，然后计数器和比较值间的比较操作产生了单脉冲的输出。但是这些操作需要一定的时钟周期，因此它限制了可得到的最小延时 t_{DELAY}。

如果要以最小延时输出波形，可以设置 PWMA_CCMRi 寄存器中的 OCiFE 位，此时强制 OCiREF（和 OCx）直接响应激励而不再依赖比较的结果，输出的波形与比较匹配时的波形一样。OCiFE 只在通道配置为 PWMA 和 PWMB 模式时起作用。

互补输出和死区插入

PWMA 能够输出两路互补信号，并且能够管理输出的瞬时关断和接通，这段时间通常被称为死区，用户应该根据连接的输出器件和它们的特性（电平转换的延时、电源开关的延时等）来调整死区时间。

配置 PWMA_CCERi 寄存器中的 CCiP 和 CCiNP 位，可以为每一个输出独立地选择极性（主输出 OCi 或互补输出 OCiN）。互补信号 OCi 和 OCiN 通过下列控制位的组合进行控制：PWMA_CCERi 寄存器的 CCiE 和 CCiNE 位，PWMA_BKR 寄存器中的 MOE、OISi、OISiN、OSSI 和 OSSR 位。特别的是，在转换到 IDLE 状态时（MOE 下降到 0）死区控制被激活。

同时设置 CCiE 和 CCiNE 位将插入死区，如果存在刹车电路，则还要设置 MOE 位。每一个通道都有一个 8 位的死区发生器。

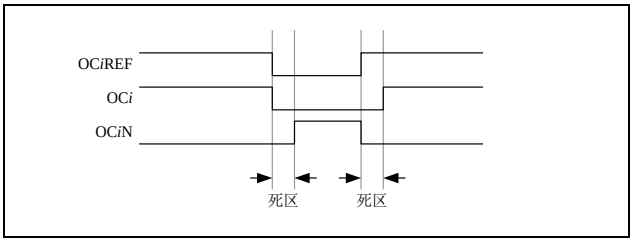
如果 OCi 和 OCiN 为高有效：

- OCi 输出信号与 OCiREF 相同，只是它的上升沿相对于 OCiREF 的上升沿有一个延迟。
- OCiN 输出信号与 OCiREF 相反，只是它的上升沿相对于 OCiREF 的下降沿有一个延迟。

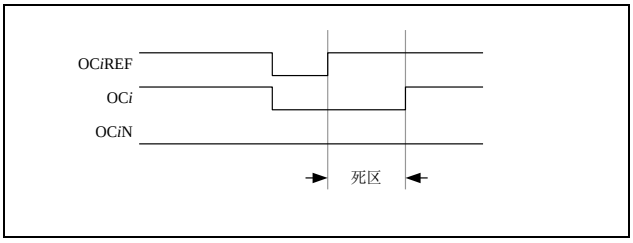
如果延迟大于当前有效的输出宽度（OCi 或者 OCiN），则不会产生相应的脉冲。

下列几张图显示了死区发生器的输出信号和当前参考信号 OCiREF 之间的关系。（假设 CCiP=0、CCiNP=0、MOE=1、CCiE=1 并且 CCiNE=1）

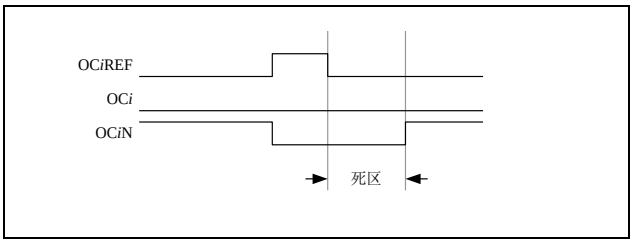
带死区插入的互补输出



死区波形延迟大于负脉冲



死区波形延迟大于正脉冲



每一个通道的死区延时都是相同的，是由 PWMA_DTR 寄存器中的 DTG 位编程配置。

重定向 OCiREF 到 OCi 或 OCiN

在输出模式下（强制输出、输出比较或 PWM 输出），通过配置 PWMA_CCERi 寄存器的 CCiE 和 CCiNE 位，OCiREF 可以被重定向到 OCi 或者 OCiN 的输出。

这个功能可以在互补输出处于无效电平时，在某个输出上送出一个特殊的波形（例如 PWM 或者静态有效电平）。另一个作用是，让两个输出同时处于无效电平，或同时处于有效电平（此时仍然是带死区的互补输出）。

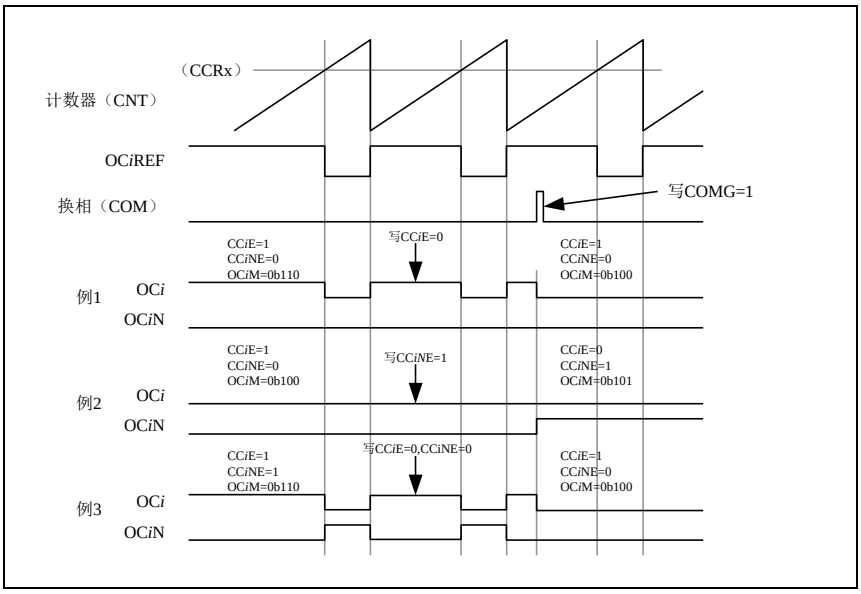
注：当只使能 OCiN（CCiE=0, CCiNE=1）时，它不会反相，而当 OCiREF 变高时立即有效。例如，如果 CCiNP=0，则 OCiN=OCiREF。另一方面，当 OCi 和 OCiN 都被使能时（CCiE=CCiNE=1），当 OCiREF 为高时 OCi 有效；而 OCiN 相反，当 OCiREF 低时 OCiN 变为有效。

针对马达控制的六步 PWM 输出

当在一个通道上需要互补输出时，预装载位有 OCiM、CCiE 和 CCiNE。在发生 COM 换相事件时，这些预装载位被传送到影子寄存器位。这样你就可以预先设置好下一步骤配置，并在同一个时刻同时修更改所有通道的配置。COM 可以通过设置 PWMA_EGR 寄存器的 COMG 位由软件产生，或在 TRGI 上升沿由硬件产生。

下图显示当发生 COM 事件时，三种不同配置下 OCx 和 OCxN 输出。

产生六步 PWM，使用 COM 的例子（OSSR=1）



25.7.8 使用刹车功能（PWMFLT）

刹车功能常用于马达控制中。当使用刹车功能时，依据相应的控制位（PWMA_BKR 寄存器中的 MOE、OSSI 和 OSSR 位），输出使能信号和无效电平都会被修改。

系统复位后，刹车电路被禁止，MOE 位为低。设置 PWMA_BKR 寄存器中的 BKE 位可以使能刹车功能。刹车输入信号的极性可以通过配置同一个寄存器中的 BKP 位选择。BKE 和 BKP 可以被同时修改。

MOE 下降沿相对于时钟模块可以是异步的，因此在实际信号（作用在输出端）和同步控制位（在 PWMA_BKR 寄存器中）之间设置了一个再同步电路。这个再同步电路会在异步信号和同步信号之间产生延迟。特别的，如果当它为低时写 MOE=1，则读出它之前必须先插入一个延时（空指令）才能读到正确的值。这是因为写入的是异步信号而读的是同步信号。

当发生刹车时（在刹车输入端出现选定的电平），有下述动作：

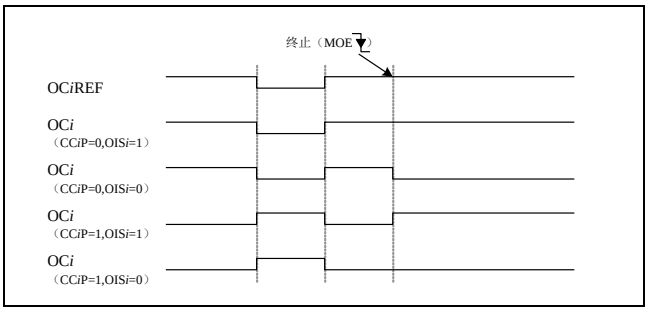
- MOE 位被异步地清除，将输出置于无效状态、空闲状态或者复位状态（由 OSSI 位选择）。这个特性在 MCU 的振荡器关闭时依然有效。
- 一旦 MOE=0，每一个输出通道输出由 PWMA_OISR 寄存器的 OISi 位设定的电平。如果 OSSI=0，则定时器不再控制输出使能信号，否则输出使能信号始终为高。
- 当使用互补输出时：
  - 输出首先被置于复位状态即无效的状态（取决于极性）。这是异步操作，即使定时器没有时钟时，此功能也有效。
  - 如果定时器的时钟依然存在，死区生成器将会重新生效，在死区之后根据 OISi 和 OISiN 位指示的电平驱动输出端口。即使在这种情况下，OCi 和 OCiN 也不能被同时驱动到有效的电平。注：因为重新同步 MOE，死区时间比通常情况下长一些（大约 2 个时钟周期）。
- 如果设置了 PWMA_IER 寄存器的 BIE 位，当刹车状态标志（PWMA_SR1 寄存器中的 BIF 位）为 1 时，则产生一个中断。
- 如果设置了 PWMA_BKR 寄存器中的 AOE 位，在下一个更新事件 UEV 时 MOE 位被自动置位。例如这可以用来进行波形控制，否则，MOE 始终保持低直到被再次置 1。这个特性可以被用在安全方面，你可以把刹车输入连到电源驱动的报警输出、热敏传感器或者其他安全器件上。

注：刹车输入为电平有效。所以，当刹车输入有效时，不能同时（自动地或者通过软件）设置 MOE。同时，状态标志 BIF 不能被清除。

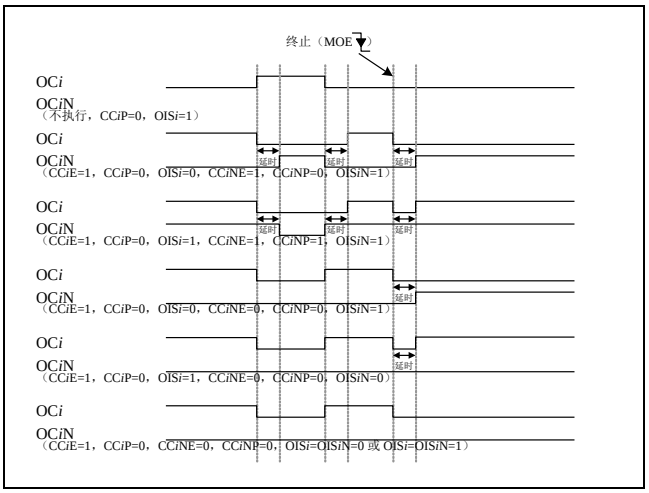
刹车由 BRK 输入产生，它的有效极性是可编程的，且由 PWMA_BKR 寄存器的 BKE 位开启或禁止。除了刹车输入和输出管理，刹车电路中还实现了写保护以保证应用程序的安全。它允许用户冻结几个配

置参数（OCi 极性和被禁止时的状态，OCiM 配置，刹车使能和极性）。用户可以通过 PWMA_BKR 寄存器的 LOCK 位，从三种级别的保护中选择一种。在 MCU 复位后 LOCK 位域只能被修改一次。

刹车响应的输出（不带互补输出的通道）



带互补输出的刹车响应的输出（PWMA 互补输出）



25.7.9 在外部事件发生时清除 OCiREF 信号

对于一个给定的通道，在 ETRF 输入端（设置 PWMA_CCMRi 寄存器中对应的 OCiCE 位为'1'）的高电平能够把 OCiREF 信号拉低，OCiREF 信号将保持为低直到发生下一次的更新事件 UEV。该功能只能用于输出比较模式和 PWM 模式，而不能用于强制模式。

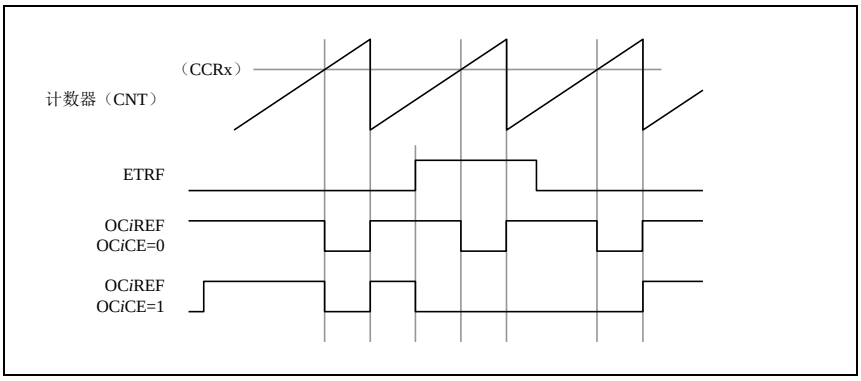
例如，OCiREF 信号可以联到一个比较器的输出，用于控制电流。这时，ETR 必须配置如下：

1. 外部触发预分频器必须处于关闭：PWMA_ETR 寄存器中的 ETPS[1:0]=00。
2. 必须禁止外部时钟模式 2：PWMA_ETR 寄存器中的 ECE=0。
3. 外部触发极性（ETP）和外部触发滤波器（ETF）可以根据需要配置。

下图显示了当 ETRF 输入变为高时，对应不同 OCiCE 的值，OCiREF 信号的动作。

在这个例子中，定时器 PWMA 被置于 PWM 模式。

ETR 清除 PWMA 的 OCiREF



25.7.10 编码器接口模式

编码器接口模式一般用于马达控制。

选择编码器接口模式的方法是：

- 如果计数器只在 TI2 的边沿计数，则置 PWMA_SMCR 寄存器中的 SMS=001；
- 如果只在 TI1 边沿计数，则置 SMS=010；
- 如果计数器同时在 TI1 和 TI2 边沿计数，则置 SMS=011。

通过设置 PWMA_CCER1 寄存器中的 CC1P 和 CC2P 位，可以选择 TI1 和 TI2 极性；如果需要，还可以对输入滤波器编程。

两个输入 TI1 和 TI2 被用来作为增量编码器的接口。假定计数器已经启动（PWMA_CR1 寄存器中的 CEN=1），则计数器在每次 TI1FP1 或 TI2FP2 上产生有效跳变时计数。TI1FP1 和 TI2FP2 是 TI1 和 TI2 在通过输入滤波器和极性控制后的信号。如果没有滤波和极性变换，则 TI1FP1=TI1，TI2FP2=TI2。根据两个输入信号的跳变顺序，产生了计数脉冲和方向信号。依据两个输入信号的跳变顺序，计数器向上或向下计数，同时硬件对 PWMA_CR1 寄存器的 DIR 位进行相应的设置。不管计数器是依靠 TI1 计数、依靠 TI2 计数或者同时依靠 TI1 和 TI2 计数，在任一输入端（TI1 或者 TI2）的跳变都会重新计算 DIR 位。

编码器接口模式基本上相当于使用了一个带有方向选择的外部时钟。这意味着计数器只在 0 到 PWMA_ARR 寄存器的自动装载值之间连续计数（根据方向，或是 0 到 ARR 计数，或是 ARR 到 0 计数）。所以在开始计数之前必须配置 PWMA_ARR。在这种模式下捕获器、比较器、预分频器、重复计数器、触发输出特性等仍工作如常。编码器模式和外部时钟模式 2 不兼容，因此不能同时操作。

编码器接口模式下，计数器依照增量编码器的速度和方向被自动的修改，因此计数器的内容始终指示着编码器的位置，计数方向与相连的传感器旋转的方向对应。

下表列出了所有可能的组合（假设 TI1 和 TI2 不同时变换）。

计数方向与编码器信号的关系

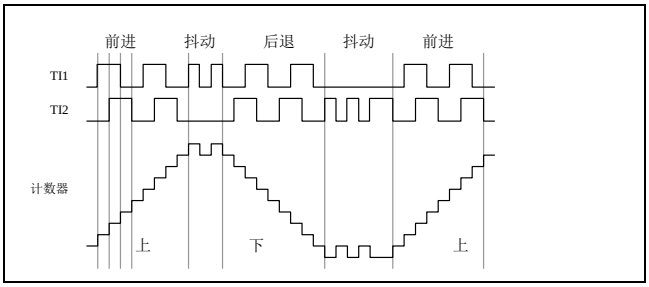
有效边沿	相对信号的电平 （TI1FP1 对应 TI2, TI2FP2 对应 TI1）	TI1FP1 信号		TI2FP2 信号	
		上升	下降	上升	下降
仅在 TI1 计数	高	向下计数	向上计数	不计数	不计数
	低	向上计数	向下计数	不计数	不计数
仅在 TI2 计数	高	不计数	不计数	向上计数	向下计数
	低	不计数	不计数	向下计数	向上计数
在 TI1 和 TI2 上计数	高	向下计数	向上计数	向上计数	向下计数
	低	向上计数	向下计数	向下计数	向上计数

一个外部的增量编码器可以直接与 MCU 连接而不需要外部接口逻辑。但是，一般使用比较器将编码器的差分输出转换成数字信号，这大大增加了抗噪声干扰能力。编码器输出的第三个信号表示机械零点，可以把它连接到一个外部中断输入并触发一个计数器复位。

下面是一个计数器操作的实例，显示了计数信号的产生和方向控制。它还显示了当选择了双边沿时，输入抖动是如何被抑制的；抖动可能会在传感器的位置靠近一个转换点时产生。在这个例子中，我们假定配置如下：

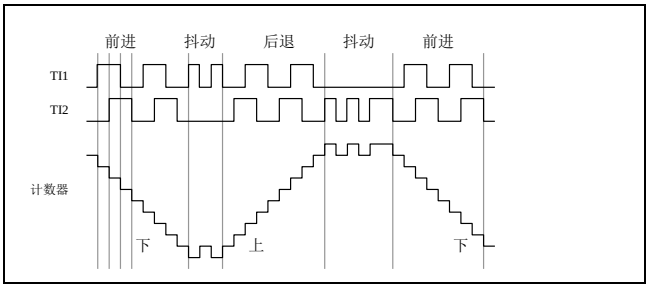
- CC1S=01（PWMA_CCMR1 寄存器，IC1FP1 映射到 TI1）
- CC2S=01（PWMA_CCMR2 寄存器，IC2FP2 映射到 TI2）
- CC1P=0（PWMA_CCER1 寄存器，IC1 不反相，IC1=TI1）
- CC2P=0（PWMA_CCER1 寄存器，IC2 不反相，IC2=TI2）
- SMS=011（PWMA_SMCR 寄存器，所有的输入均在上升沿和下降沿有效）.
- CEN=1（PWMA_CR1 寄存器，计数器使能）

编码器模式下的计数器操作实例



下图为当 IC1 极性反相时计数器的操作实例（CC1P=1，其他配置与上例相同）

IC1 反相的编码器接口模式实例



当定时器配置成编码器接口模式时，提供传感器当前位置的信息。使用另外一个配置在捕获模式下的定时器测量两个编码器事件的间隔，可以获得动态的信息（速度、加速度、减速度）。指示机械零点的编码器输出可被用做此目的。根据两个事件间的间隔，可以按照一定的时间间隔读出计数器。如果可能的话，你可以把计数器的值锁存到第三个输入捕获寄存器（捕获信号必须是周期的并且可以由另一个定时器产生）。

25.8 中断

PWMA/PWMB 各有 8 个中断请求源：

- 刹车中断
- 触发中断
- COM 事件中断
- 输入捕捉/输出比较 4 中断
- 输入捕捉/输出比较 3 中断
- 输入捕捉/输出比较 2 中断
- 输入捕捉/输出比较 1 中断
- 更新事件中断（如：计数器上溢，下溢及初始化）

为了使用中断特性，对每个被使用的中断通道，设置 PWM_IER/PWMB_IER 寄存器中相应的中断使能位：即 BIE，TIE，COMIE，CCiIE，UIE 位。通过设置 PWMA_EGR/PWMB_EGR 寄存器中的相应位，也可以用软件产生上述各个中断源。

25.9 PWMA/PWMB 寄存器描述

25.9.1 高级 PWM 功能脚切换

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_PS	FEB2H	C4PS[1:0]		C3PS[1:0]		C2PS[1:0]		C1PS[1:0]	
PWMB_PS	FEB6H	C8PS[1:0]		C7PS[1:0]		C6PS[1:0]		C5PS[1:0]	
PWMA_ETRPS	FEB0H						BRKAPS	ETRAPS[1:0]	
PWMB_ETRPS	FEB4H						BRKBPS	ETRBPS[1:0]	

C1PS[1:0]：高级 PWM 通道 1 输出脚选择位

C1PS[1:0]	PWM1P	PWM1N
00	P1.0	P1.1
01	P2.0	P2.1
10	P6.0	P6.1
11	-	-

C2PS[1:0]：高级 PWM 通道 2 输出脚选择位

C2PS[1:0]	PWM2P	PWM2N
00	P1.2/P5.4[1]	P1.3
01	P2.2	P2.3
10	P6.2	P6.3
11	-	-

注[1] ：对于部分没有 P1.2 口的单片机型号，此功能在 P5.4 口上

C3PS[1:0]：高级 PWM 通道 3 输出脚选择位

C3PS[1:0]	PWM3P	PWM3N
00	P1.4	P1.5
01	P2.4	P2.5
10	P6.4	P6.5
11	-	-

C4PS[1:0]：高级 PWM 通道 4 输出脚选择位

C4PS[1:0]	PWM4P	PWM4N
00	P1.6	P1.7
01	P2.6	P2.7
10	P6.6	P6.7
11	P3.4	P3.3

C5PS[1:0]：高级 PWM 通道 5 输出脚选择位

C5PS[1:0]	PWM5
00	P2.0
01	P1.7
10	P0.0
11	P7.4

C6PS[1:0]：高级 PWM 通道 6 输出脚选择位

C6PS[1:0]	PWM6
00	P2.1
01	P5.4
10	P0.1
11	P7.5

C7PS[1:0]：高级 PWM 通道 7 输出脚选择位

C7PS[1:0]	PWM7
00	P2.2
01	P3.3
10	P0.2
11	P7.6

C8PS[1:0]：高级 PWM 通道 8 输出脚选择位

C8PS[1:0]	PWM8
00	P2.3
01	P3.4
10	P0.3
11	P7.7

ETRAPS[1:0]：高级 PWMA 的外部触发脚 ERI 选择位

ETRAPS [1:0]	PWMETI
00	P3.2
01	P4.1
10	P7.3
11	-

ETRBPS[1:0]：高级 PWMB 的外部触发脚 ERIB 选择位

ETRBPS [1:0]	PWMETI2
00	P3.2
01	P0.6
10	-
11	-

BRKAPS：高级 PWMA 的刹车脚 PWMFLT 选择位

BRKAPS	PWMFLT
0	P3.5
1	比较器的输出

BRKBPS：高级 PWMB 的刹车脚 PWMFLT2 选择位

BRKBPS	PWMFLT2
0	P3.5
1	比较器的输出

25.9.2 输出使能寄存器（PWMx_ENO）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_ENO	FEB1H	ENO4N	ENO4P	ENO3N	ENO3P	ENO2N	ENO2P	ENO1N	ENO1P
PWMB_ENO	FEB5H	-	ENO8P	-	ENO7P	-	ENO6P	-	ENO5P

ENO8P：PWM8 输出控制位

0：禁止 PWM8 输出

1：使能 PWM8 输出

ENO7P：PWM7 输出控制位

0：禁止 PWM7 输出

1：使能 PWM7 输出

ENO6P：PWM6 输出控制位

0：禁止 PWM6 输出

1：使能 PWM6 输出

ENO5P：PWM5 输出控制位

0：禁止 PWM5 输出

1：使能 PWM5 输出

ENO4N：PWM4N 输出控制位

0：禁止 PWM4N 输出

1：使能 PWM4N 输出

ENO4P：PWM4P 输出控制位

0：禁止 PWM4P 输出

1：使能 PWM4P 输出

ENO3N：PWM3N 输出控制位

0：禁止 PWM3N 输出

1：使能 PWM3N 输出

ENO3P：PWM3P 输出控制位

0：禁止 PWM3P 输出

1：使能 PWM3P 输出

ENO2N：PWM2N 输出控制位

0：禁止 PWM2N 输出

1：使能 PWM2N 输出

ENO2P：PWM2P 输出控制位

0：禁止 PWM2P 输出

1：使能 PWM2P 输出

ENO1N：PWM1N 输出控制位

0：禁止 PWM1N 输出

1：使能 PWM1N 输出

ENO1P：PWM1P 输出控制位

0：禁止 PWM1P 输出

1：使能 PWM1P 输出

25.9.3 输出附加使能寄存器（PWMx_IOAUX）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_IOAUX	FEB3H	AUX4N	AUX4P	AUX3N	AUX3P	AUX2N	AUX2P	AUX1N	AUX1P
PWMB_IOAUX	FEB7H	-	AUX8P	-	AUX7P	-	AUX6P	-	AUX5P

AUX8P：PWM8 输出附加控制位

0：PWM8 的输出直接由 ENO8P 控制

1：PWM8 的输出由 ENO8P 和 PWMB_BKR 共同控制

AUX7P：PWM7 输出附加控制位

0：PWM7 的输出直接由 ENO7P 控制

1：PWM7 的输出由 ENO7P 和 PWMB_BKR 共同控制

AUX6P：PWM6 输出附加控制位

0：PWM6 的输出直接由 ENO6P 控制

1：PWM6 的输出由 ENO6P 和 PWMB_BKR 共同控制

AUX5P：PWM5 输出附加控制位

0：PWM5 的输出直接由 ENO5P 控制

1：PWM5 的输出由 ENO5P 和 PWMB_BKR 共同控制

AUX4N：PWM4N 输出附加控制位

0：PWM4N 的输出直接由 ENO4N 控制

1：PWM4N 的输出由 ENO4N 和 PWMA_BKR 共同控制

AUX4P：PWM4P 输出附加控制位

0：PWM4P 的输出直接由 ENO4P 控制

1：PWM4P 的输出由 ENO4P 和 PWMA_BKR 共同控制

AUX3N：PWM3N 输出附加控制位

0：PWM3N 的输出直接由 ENO3N 控制

1：PWM3N 的输出由 ENO3N 和 PWMA_BKR 共同控制

AUX3P：PWM3P 输出附加控制位

0：PWM3P 的输出直接由 ENO3P 控制

1：PWM3P 的输出由 ENO3P 和 PWMA_BKR 共同控制

AUX2N：PWM2N 输出附加控制位

0：PWM2N 的输出直接由 ENO2N 控制

1：PWM2N 的输出由 ENO2N 和 PWMA_BKR 共同控制

AUX2P：PWM2P 输出附加控制位

0：PWM2P 的输出直接由 ENO2P 控制

1：PWM2P 的输出由 ENO2P 和 PWMA_BKR 共同控制

AUX1N：PWM1N 输出附加控制位

0：PWM1N 的输出直接由 ENO1N 控制

1：PWM1N 的输出由 ENO1N 和 PWMA_BKR 共同控制

AUX1P：PWM1P 输出附加控制位

0：PWM1P 的输出直接由 ENO1P 控制

1：PWM1P 的输出由 ENO1P 和 PWMA_BKR 共同控制

25.9.4 控制寄存器 1（PWMx_CR1）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CR1	FEC0H	ARPEA	CMSA[1:0]		DIRA	OPMA	URSA	UDISA	CENA
PWMB_CR1	FEE0H	ARPEB	CMSB[1:0]		DIRB	OPMB	URSB	UDISB	CENB

ARPEn：自动预装载允许位（n=A,B）

0：PWMn_ARR 寄存器没有缓冲，它可以被直接写入

1：PWMn_ARR 寄存器由预装载缓冲器缓冲

CMSn[1:0]：选择对齐模式（n= A,B）

CMSn[1:0]	对齐模式	说明
00	边沿对齐模式	计数器依据方向位（DIR）向上或向下计数
01	中央对齐模式1	计数器交替地向上和向下计数。 配置为输出的通道的输出比较中断标志位(CCnIF)只在计数器向下计数时被置1。
10	中央对齐模式2	计数器交替地向上和向下计数。 配置为输出的通道的输出比较中断标志位(CCnIF)只在计数器向上计数时被置1。
11	中央对齐模式3	计数器交替地向上和向下计数。 配置为输出的通道的输出比较中断标志位(CCnIF)在计数器向上和向下计数时均被置1。

注 1：在计数器开启时（CEN=1），不允许从边沿对齐模式转换到中央对齐模式。

注 2：在中央对齐模式下，编码器模式（SMS=001，010，011）必须被禁止。

DIRn：计数器的计数方向（n= A,B）

0：计数器向上计数；

1：计数器向下计数。

注：当计数器配置为中央对齐模式或编码器模式时，该位为只读。

OPMn：单脉冲模式（n= A,B）

0：在发生更新事件时，计数器不停止；

1：在发生下一次更新事件时，清除 CEN 位，计数器停止。

URSn：更新请求源（n= A,B）

0：如果 UDIS 允许产生更新事件，则下述任一事件产生一个更新中断：

- 寄存器被更新（计数器上溢/下溢）
- 软件设置 UG 位
- 时钟/触发控制器产生的更新

1：如果 UDIS 允许产生更新事件，则只有当下列事件发生时才产生更新中断，并 UIF 置 1：

- 寄存器被更新（计数器上溢/下溢）

UDISn：禁止更新（n= A,B）

0：一旦下列事件发生，产生更新（UEV）事件：

- 计数器溢出/下溢
- 产生软件更新事件
- 时钟/触发模式控制器产生的硬件复位 被缓存的寄存器被装入它们的预装载值。

1：不产生更新事件，影子寄存器（ARR、PSC、CCRx）保持它们的值。如果设置了 UG 位或时钟/触发控制器发出了一个硬件复位，则计数器和预分频器被重新初始化。

CENn：允许计数器（n= A,B）

0：禁止计数器；

1：使能计数器。

注：在软件设置了 CEN 位后，外部时钟、门控模式和编码器模式才能工作。然而触发模式可以自动地通过硬件设置 CEN 位。

25.9.5 控制寄存器 2（PWMx_CR2），及实时触发 ADC

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CR2	FEC1H	TI1S	MMSA[2:0]			-	COMSA	-	CCPCA
PWMB_CR2	FEE1H	TI5S	MMSB[2:0]			-	COMSB	-	CCPCB

TI1S：第一组 PWM/PWMA 的 TI1 选择

0：PWM1P 输入管脚连到 TI1（数字滤波器的输入）；

1：PWM1P、PWM2P 和 PWM3P 管脚经异或后连到第一组 PWM 的 TI1。

TI5S：第二组 PWM/PWMB 的 TI5 选择

0：PWM5 输入管脚连到 TI5（数字滤波器的输入）；

1：PWM5、PWM6 和 PWM7 管脚经异或后连到第二组 PWM/PWMB 的 TI5。

MMSA[2:0]：主模式选择

MMSA[2:0]	主模式	说明
000	复位	PWMA_EGR寄存器的UG位被用于作为触发输出（TRGO）。如果触发输入（时钟/触发控制器配置为复位模式）产生复位，则TRGO上的信号相对实际的复位会有一个延迟
001	使能	计数器使能信号被用于作为触发输出（TRGO）。其用于启动ADC，以便控制在一段时间内使能ADC。计数器使能信号是通过CEN控制位和门控模式下的触发输入信号的逻辑或产生。除非选择了主/从模式，当计数器使能信号受控于触发输入时，TRGO上会有一个延迟。 注：当需要使用PWM触发ADC转换时，需要先设置ADC_CONTR寄存器中的ADC_POWER、ADC_CHS以及ADC_EPWMT，当PWM产生TRGO内部信号时，系统会自动设置ADC_START来启动AD转换。详细使用请参考范例程序“使用PWM的CEN启动PWMA定时器，实时触发ADC”
010	更新	更新事件被选为触发输出（TRGO）
011	比较脉冲	一旦发生一次捕获或一次比较成功，当CC1IF标志被置1时，触发输出送出一个正脉冲（TRGO）
100	比较	OC1REF信号被用于作为触发输出（TRGO）
101	比较	OC2REF信号被用于作为触发输出（TRGO）
110	比较	OC3REF信号被用于作为触发输出（TRGO）
111	比较	OC4REF信号被用于作为触发输出（TRGO）

MMSB[2:0]：主模式选择

MMSB[2:0]	主模式	说明
000	复位	PWMB_EGR寄存器的UG位被用于作为触发输出（TRGO）。如果触发输入（时钟/触发控制器配置为复位模式）产生复位，则TRGO上的信号相对实际的复位会有一个延迟
001	使能	计数器使能信号被用于作为触发输出（TRGO）。其用于启动多个PWM，以便控制在一段时间内使能从PWM。计数器使能信号是通过CEN控制位和门控模式下的触发输入信号的逻辑或产生。除非选择了主/从模式，当计数器使能信号受控于触发输入时，TRGO上会有一个延迟。
010	更新	更新事件被选为触发输出（TRGO）
011	比较脉冲	一旦发生一次捕获或一次比较成功，当CC5IF标志被置1时，触发输出送出一个正脉冲（TRGO）
100	比较	OC5REF信号被用于作为触发输出（TRGO）
101	比较	OC6REF信号被用于作为触发输出（TRGO）
110	比较	OC7REF信号被用于作为触发输出（TRGO）
111	比较	OC8REF信号被用于作为触发输出（TRGO）

注：只有第一组 PWM（PWMA）的 TRGO 可用于触发启动 ADC

注：只有第二组 PWM（PWMB）的 TRGO 可用于第一组 PWM（PWMA）的 ITR2

COMSn：捕获/比较控制位的更新控制选择（n=A,B）

0：当 CCPCn=1 时，只有在 COMG 位置 1 的时候这些控制位才被更新

1：当 CCPCn=1 时，只有在 COMG 位置 1 或 TRGI 发生上升沿的时候这些控制位才被更新

CCPCn：捕获/比较预装载控制位（n= A,B）

0：CCiE，CCiNE，CCiP，CCiNP 和 OCiM 位不是预装载的

1：CCiE，CCiNE，CCiP，CCiNP 和 OCiM 位是预装载的；设置该位后，它们只在设置了 COMG 位后被更新。

注：该位只对具有互补输出的通道起作用。

25.9.6 从模式控制寄存器(PWMx_SMCR)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_SMCR	FEC2H	MSMA	TSA[2:0]			-	SMSA[2:0]		
PWMB_SMCR	FEE2H	MSMB	TSB[2:0]			-	SMSB[2:0]		

MSMn：主/从模式（n= A,B）

0：无作用

1：触发输入（TRGI）上的事件被延迟了，以允许 PWMn 与它的从 PWM 间的完美同步（通过 TRGO）

TSA[2:0]：触发源选择

TSA[2:0]	触发源
000	-
001	-
010	内部触发ITR2
011	-
100	TI1的边沿检测器（TI1F_ED）
101	滤波后的定时器输入1（TI1FP1）
110	滤波后的定时器输入2（TI2FP2）
111	外部触发输入（ETRF）

TSB[2:0]：触发源选择

TSB[2:0]	触发源
000	-
001	-
010	-
011	-
100	TI5的边沿检测器（TI5F_ED）
101	滤波后的定时器输入1（TI5FP5）
110	滤波后的定时器输入2（TI6FP6）
111	外部触发输入（ETRF）

注：这些位只能在 SMS=000 时被改变，以避免在改变时产生错误的边沿检测。

SMSA[2:0]：时钟/触发/从模式选择

SMSA[2:0]	功能	说明
000	内部时钟模式	如果CEN=1，则预分频器直接由内部时钟驱动
001	编码器模式1	根据TI1FP1的电平，计数器在TI2FP2的边沿向上/下计数
010	编码器模式2	根据TI2FP2的电平，计数器在TI1FP1的边沿向上/下计数
011	编码器模式3	根据另一个输入的电平，计数器在TI1FP1和TI2FP2的边沿向上/下计数
100	复位模式	在选中的触发输入（TRGI）的上升沿时重新初始化计数器，并且产生一个更新 寄存器的信号
101	门控模式	当触发输入（TRGI）为高时，计数器的时钟开启。一旦触发输入变为低，则计 数器停止（但不复位）。计数器的启动和停止都是受控的
110	触发模式	计数器在触发输入TRGI的上升沿启动（但不复位），只有计数器的启动是受控 的
111	外部时钟模式1	选中的触发输入（TRGI）的上升沿驱动计数器。 注：如果TI1F_ED被选为触发输入（TS=100）时，不要使用门控模式。这是因为TI1F_ED在每次TI1F变化时只是输出一个脉冲，然而门控模式是要检查触发输入的电平

SMSB[2:0]：时钟/触发/从模式选择

SMSB[2:0]	功能	说明
000	内部时钟模式	如果CEN=1，则预分频器直接由内部时钟驱动
001	编码器模式1	根据TI5FP5的电平，计数器在TI6FP6的边沿向上/下计数
010	编码器模式2	根据TI6FP6的电平，计数器在TI5FP5的边沿向上/下计数
011	编码器模式3	根据另一个输入的电平，计数器在TI5FP5和TI6FP6的边沿向上/下计数
100	复位模式	在选中的触发输入（TRGI）的上升沿时重新初始化计数器，并且产生一个更新 寄存器的信号
101	门控模式	当触发输入（TRGI）为高时，计数器的时钟开启。一旦触发输入变为低，则计 数器停止（但不复位）。计数器的启动和停止都是受控的
110	触发模式	计数器在触发输入TRGI的上升沿启动（但不复位），只有计数器的启动是受控 的
111	外部时钟模式1	选中的触发输入（TRGI）的上升沿驱动计数器。 注：如果TI5F_ED被选为触发输入（TS=100）时，不要使用门控模式。这是因为TI5F_ED在每次TI5F变化时只是输出一个脉冲，然而门控模式是要检查触发输入的电平

25.9.7 外部触发寄存器(PWMx_ETR)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_ETR	FEC3H	ETP1	ECEA	ETPSA[1:0]		ETFA[3:0]			
PWMB_ETR	FEE3H	ETP2	ECEB	ETPSB[1:0]		ETFB[3:0]			

ETPn：外部触发 ETR 的极性（n= A,B）

0：高电平或上升沿有效

1：低电平或下降沿有效

ECEn：外部时钟使能（n= A,B）

0：禁止外部时钟模式 2

1：使能外部时钟模式 2，计数器的时钟为 ETRF 的有效沿。

注 1：ECE 置 1 的效果与选择把 TRGI 连接到 ETRF 的外部时钟模式 1 相同（PWMn_SMCR 寄存器中，SMS=111，TS=111）。

注 2：外部时钟模式 2 可与下列模式同时使用：触发标准模式；触发复位模式；触发门控模式。但是，此时 TRGI 决不能与 ETRF 相连（PWMn_SMCR 寄存器中，TS 不能为 111）。

注 3：外部时钟模式 1 与外部时钟模式 2 同时使能，外部时钟输入为 ETRF。

ETPSn：外部触发预分频器外部触发信号 EPRP 的频率最大不能超过 fMASTER/4。可用预分频器来降低 ETRP 的频率，当 EPRP 的频率很高时，它非常有用：（n= A,B）

00：预分频器关闭

01：EPRP 的频率/2

02：EPRP 的频率/4

03：EPRP 的频率/8

ETFn[3:0]：外部触发滤波器选择，该位域定义了 ETRP 的采样频率及数字滤波器长度。（n= A,B）

ETFn[3:0]	时钟数	ETF[3:0]	时钟数
0000	1	1000	48
0001	2	1001	64
0010	4	1010	80
0011	8	1011	96
0100	12	1100	128
0101	16	1101	160
0110	24	1110	192
0111	32	1111	256

25.9.8 中断使能寄存器(PWMx_IER)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_IER	FEC4H	BIEA	TIEA	COMIEA	CC4IE	CC3IE	CC2IE	CC1IE	UIEA
PWMB_IER	FEE4H	BIEB	TIEB	COMIEB	CC8IE	CC7IE	CC6IE	CC5IE	UIEB

BIEn：允许刹车中断（n= A,B）

0：禁止刹车中断；

1：允许刹车中断。

TIE：触发中断使能（n= A,B）

0：禁止触发中断；

1：使能触发中断。

COMIE：允许 COM 中断（n= A,B）

0：禁止 COM 中断；

1：允许 COM 中断。

CCnIE：允许捕获/比较 n 中断（n=1,2,3,4,5,6,7,8）

0：禁止捕获/比较 n 中断；

1：允许捕获/比较 n 中断。

UIEn：允许更新中断（n= A,B）

0：禁止更新中断；

1：允许更新中断。

25.9.9 状态寄存器 1(PWMx_SR1)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_SR1	FEC5H	BIFA	TIFA	COMIFA	CC4IF	CC3IF	CC2IF	CC1IF	UIFA
PWMB_SR1	FEE5H	BIFB	TIFB	COMIFB	CC8IF	CC7IF	CC6IF	CC5IF	UIFB

BIFn：刹车中断标记。一旦刹车输入有效，由硬件对该位置 1。如果刹车输入无效，则该位可由软件清 0。（n= A,B）

0：无刹车事件产生

1：刹车输入上检测到有效电平

TIFn：触发器中断标记。当发生触发事件时由硬件对该位置 1。由软件清 0。（n= A,B）

0：无触发器事件产生

1：触发中断等待响应

COMIFn：COM 中断标记。一旦产生 COM 事件该位由硬件置 1。由软件清 0。（n= A,B）

0：无 COM 事件产生

1：COM 中断等待响应

CC8IF：捕获/比较8中断标记，参考CC1IF描述

CC7IF：捕获/比较7中断标记，参考CC1IF描述

CC6IF：捕获/比较6中断标记，参考CC1IF描述

CC5IF：捕获/比较5中断标记，参考CC1IF描述

CC4IF：捕获/比较4中断标记，参考CC1IF描述

CC3IF：捕获/比较3中断标记，参考CC1IF描述

CC2IF：捕获/比较2中断标记，参考CC1IF描述

CC1IF：捕获/比较1中断标记。

如果通道CC1配置为输出模式：

当计数器值与比较值匹配时该位由硬件置1，但在中心对称模式下除外。它由软件清0。

0：无匹配发生；

1：PWMA_CNT 的值与 PWMA_CCR1 的值匹配。

注：在中心对称模式下，当计数器值为 0 时，向上计数，当计数器值为 ARR 时，向下计数（它从 0 向上计数到 ARR-1，再由 ARR 向下计数到 1）。因此，对所有的 SMS 位值，这两个值都不置标记。但是，如果 CCR1>ARR，则当 CNT 达到 ARR 值时，CC1IF 置 1。

如果通道CC1配置为输入模式：

当捕获事件发生时该位由硬件置1，它由软件清0或通过读PWMA_CCR1L清0。

0：无输入捕获产生

1：计数器值已被捕获至 PWMA_CCR1

UIFn：更新中断标记 当产生更新事件时该位由硬件置 1。它由软件清 0。（n= A,B）

0：无更新事件产生

1：更新事件等待响应。当寄存器被更新时该位由硬件置 1

- 若 PWMn_CR1 寄存器的 UDIS=0，当计数器上溢或下溢时
- 若 PWMn_CR1 寄存器的 UDIS=0、URS=0，当设置 PWMn_EGR 寄存器的 UG 位软件对计数器 CNT 重新初始化时
- 若 PWMn_CR1 寄存器的 UDIS=0、URS=0，当计数器 CNT 被触发事件重新初始化时

25.9.10 状态寄存器 2(PWMx_SR2)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_SR2	FEC6H	-	-	-	CC4OF	CC3OF	CC2OF	CC1OF	-
PWMB_SR2	FEE6H	-	-	-	CC8OF	CC7OF	CC6OF	CC5OF	-

CC8OF：捕获/比较8重复捕获标记。参见CC1OF描述。

CC7OF：捕获/比较7重复捕获标记。参见CC1OF描述。

CC6OF：捕获/比较6重复捕获标记。参见CC1OF描述。

CC5OF：捕获/比较5重复捕获标记。参见CC1OF描述。

CC4OF：捕获/比较4重复捕获标记。参见CC1OF描述。

CC3OF：捕获/比较3重复捕获标记。参见CC1OF描述。

CC2OF：捕获/比较2重复捕获标记。参见CC1OF描述。

CC1OF：捕获/比较1重复捕获标记。仅当相应的通道被配置为输入捕获时，该标记可由硬件置1。写0可清除该位。

0：无重复捕获产生；

1：计数器的值被捕获到 PWMA_CCR1 寄存器时，CC1IF 的状态已经为 1。

25.9.11 事件产生寄存器（PWMx_EGR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_EGR	FEC7H	BGA	TGA	COMGA	CC4G	CC3G	CC2G	CC1G	UGA
PWMB_EGR	FEE7H	BGB	TGB	COMGB	CC8G	CC7G	CC6G	CC5G	UGB

BGn：产生刹车事件。该位由软件置 1，用于产生一个刹车事件，由硬件自动清 0（n= A,B）

0：无动作

1：产生一个刹车事件。此时 MOE=0、BIF=1，若开启对应的中断（BIE=1），则产生相应的中断

TGn：产生触发事件。该位由软件置 1，用于产生一个触发事件，由硬件自动清 0（n= A,B）

0：无动作

1：TIF=1，若开启对应的中断（TIE=1），则产生相应的中断

COMGn：捕获/比较事件，产生控制更新。该位由软件置 1，由硬件自动清 0（n= A,B）

0：无动作

1：CCPC=1，允许更新 CCiE、CCiNE、CCiP，CCiNP，OCiM 位。

注：该位只对拥有互补输出的通道有效

CC8G：产生捕获/比较 8 事件。参考 CC1G 描述

CC7G：产生捕获/比较 7 事件。参考 CC1G 描述

CC6G：产生捕获/比较 6 事件。参考 CC1G 描述

CC5G：产生捕获/比较 5 事件。参考 CC1G 描述

CC4G：产生捕获/比较 4 事件。参考 CC1G 描述

CC3G：产生捕获/比较 3 事件。参考 CC1G 描述

CC2G：产生捕获/比较 2 事件。参考 CC1G 描述

CC1G：产生捕获/比较 1 事件。产生捕获/比较 1 事件。该位由软件置 1，用于产生一个捕获/比较事件，由硬件自动清 0。

0：无动作；

1：在通道 CC1 上产生一个捕获/比较事件。

若通道 CC1 配置为输出：设置 CC1IF=1，若开启对应的中断，则产生相应的中断。

若通道 CC1 配置为输入：当前的计数器值被捕获至 PWMA_CCR1 寄存器，设置 CC1IF=1，若开启对应的中断，则产生相应的中断。若 CC1IF 已经为 1，则设置 CC1OF=1。

UGn：产生更新事件 该位由软件置 1，由硬件自动清 0。（n= A,B）

0：无动作；

1：重新初始化计数器，并产生一个更新事件。

注意预分频器的计数器也被清 0（但是预分频系数不变）。若在中心对称模式下或 DIR=0（向上计数）则计数器被清 0；若 DIR=1（向下计数）则计数器取 PWMn_ARR 的值。

25.9.12 捕获/比较模式寄存器 1（PWMx_CCMR1）

通道可用于捕获输入模式或比较输出模式，通道的方向由相应的 CCnS 位定义。该寄存器其它位的作用在输入和输出模式下不同。OCxx 描述了通道在输出模式下的功能，ICxx 描述了通道在输入模式下的功能。因此必须注意，同一个位在输出模式和输入模式下的功能是不同的。

通道配置为比较输出模式

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCMR1	FEC8H	OC1CE	OC1M[2:0]			OC1PE	OC1FE	CC1S[1:0]	
PWMB_CCMR1	FEE8H	OC5CE	OC5M[2:0]			OC5PE	OC5FE	CC5S[1:0]	

OCnCE：输出比较 n 清零使能。该位用于使能使用 PWMETI 引脚上的外部事件来清通道 n 的输出信号（OCnREF）（n=1,5）

0：OCnREF 不受 ETRF 输入的影响；

1：一旦检测到 ETRF 输入高电平，OCnREF=0。

OCnM[2:0]：输出比较 n 模式。该 3 位定义了输出参考信号 OCnREF 的动作，而 OCnREF 决定了 OCn 的值。OCnREF 是高电平有效，而 OCn 的有效电平取决于 CCnP 位。（n=1,5）

OCnM[2:0]	模式	说明
000	冻结	PWMn_CCR1与PWMn_CNT间的比较对OCnREF不起作用
001	匹配时设置通道n的输出为有效电平	当PWMn_CCR1=PWMn_CNT时，OCnREF输出高
010	匹配时设置通道n的输出为无效电平	当PWMn_CCR1=PWMn_CNT时，OCnREF输出低
011	翻转	当PWMn_CCR1=PWMn_CNT时，翻转OCnREF
100	强制为无效电平	强制OCnREF为低
101	强制为有效电平	强制OCnREF为高
110	PWM模式1	在向上计数时，当PWMn_CNT<PWMn_CCR1时 OCnREF输出高，否则OCnREF输出低 在向下计数时，当PWMn_CNT>PWMn_CCR1时 OCnREF输出低，否则OCnREF输出高
111	PWM模式2	在向上计数时，当PWMn_CNT<PWMn_CCR1时 OCnREF输出低，否则OCnREF输出高 在向下计数时，当PWMn_CNT>PWMn_CCR1时 OCnREF输出高，否则OCnREF输出低

注 1：一旦 LOCK 级别设为 3（PWMn_BKR 寄存器中的 LOCK 位）并且 CCnS=00（该通道配置成输出）则该位不能被修改。

注 2：在 PWM 模式 1 或 PWM 模式 2 中，只有当比较结果改变了或在输出比较模式中从冻结模式切换到 PWM 模式时，OCnREF 电平才改变。

注 3：在有互补输出的通道上，这些位是预装载的。如果 PWMn_CR2 寄存器的 CCPC=1，OCM 位只有在 COM 事件发生时，才从预装载位取新值。

OCnPE：输出比较 n 预装载使能（n=1,5）

0：禁止 PWMn_CCR1 寄存器的预装载功能，可随时写入 PWMn_CCR1 寄存器，并且新写入的数值立即起作用。

1：开启 PWMn_CCR1 寄存器的预装载功能，读写操作仅对预装载寄存器操作，PWMn_CCR1 的预装载值在更新事件到来时被加载至当前寄存器中。

注 1：一旦 LOCK 级别设为 3（PWMn_BKR 寄存器中的 LOCK 位）并且 CCnS=00（该通道配置成输出）则该位不能被修改。

注 2：为了操作正确，在 PWM 模式下必须使能预装载功能。但在单脉冲模式下（PWMn_CR1 寄存器的 OPM=1），它不是必须的。

注：OC1PE 必须在通道打开时（PWMA_CCER1 寄存器的 CC1E=1）才是可写的。

注：OC5PE 必须在通道打开时（PWMB_CCER1 寄存器的 CC5E=1）才是可写的。

OCnFE：输出比较 n 快速使能。该位用于加快 CC 输出对触发输入事件的响应。（n=1,5）

0：根据计数器与 CCRn 的值，CCn 正常操作，即使触发器是打开的。当触发器的输入有一个有效沿时，激活 CCn 输出的最小延时为 5 个时钟周期。

1：输入到触发器的有效沿的作用就象发生了一次比较匹配。因此，OC 被设置为比较电平而与比较结果无关。采样触发器的有效沿和 CC1 输出间的延时被缩短为 3 个时钟周期。OCFE 只在通道被配置成 PWMA 或 PWMB 模式时起作用。

注：OC1FE 必须在通道打开时（PWMA_CCER1 寄存器的 CC1E=1）才是可写的。

注：OC5FE 必须在通道打开时（PWMB_CCER1 寄存器的 CC5E=1）才是可写的。

CC1S[1:0]：捕获/比较 1 选择。这两位定义通道的方向（输入/输出），及输入脚的选择

CC1S[1:0]	方向	输入脚
00	输出	
01	输入	IC1映射在TI1FP1上
10	输入	IC1映射在TI2FP1上
11	输入	IC1映射在TRC上。此模式仅工作在内部触发器输入被选中时（由PWMA_SMCR寄存器的TS位选择）

CC5S[1:0]：捕获/比较 5 选择。这两位定义通道的方向（输入/输出），及输入脚的选择

CC5S[1:0]	方向	输入脚
00	输出	
01	输入	IC5映射在TI5FP5上
10	输入	IC5映射在TI6FP5上
11	输入	IC5映射在TRC上。此模式仅工作在内部触发器输入被选中时（由PWM5_SMCR寄存器的TS位选择）

注：CC1S 仅在通道关闭时（PWMA_CCER1 寄存器的 CC1E=0）才是可写的。

注：CC5S 仅在通道关闭时（PWMB_CCER1 寄存器的 CC5E=0）才是可写的。

通道配置为捕获输入模式

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCMR1	FEC8H	IC1F[3:0]				IC1PSC[1:0]		CC1S[1:0]	
PWMB_CCMR1	FEE8H	IC5F[3:0]				IC5PSC[1:0]		CC5S[1:0]	

ICnF[3:0]：输入捕获 n 滤波器选择，该位域定义了 TIn 的采样频率及数字滤波器长度。（n=1,5）

ICnF[3:0]	时钟数	ICnF[3:0]	时钟数
0000	1	1000	48
0001	2	1001	64
0010	4	1010	80
0011	8	1011	96
0100	12	1100	128
0101	16	1101	160
0110	24	1110	192
0111	32	1111	256

注：即使对于带互补输出的通道，该位域也是非预装载的，并且不会考虑 CCPC（PWMn_CR2 寄存器）的值

ICnPSC[1:0]：输入/捕获 n 预分频器。这两位定义了 CCn 输入（IC1）的预分频系数。（n=1,5）

00：无预分频器，捕获输入口上检测到的每一个边沿都触发一次捕获

01：每 2 个事件触发一次捕获

10：每 4 个事件触发一次捕获

11：每 8 个事件触发一次捕获

注：IC1PSC 必须在通道打开时（PWMA_CCER1 寄存器的 CC1E=1）才是可写的。

注：IC5PSC 必须在通道打开时（PWMB_CCER1 寄存器的 CC5E=1）才是可写的。

CC1S[1:0]：捕获/比较 1 选择。这两位定义通道的方向（输入/输出），及输入脚的选择

CC1S[1:0]	方向	输入脚
00	输出	
01	输入	IC1映射在TI1FP1上
10	输入	IC1映射在TI2FP1上
11	输入	IC1映射在TRC上。此模式仅工作在内部触发器输入被选中时（由PWMA_SMCR寄存器的TS位选择）

CC5S[1:0]：捕获/比较 5 选择。这两位定义通道的方向（输入/输出），及输入脚的选择

CC5S[1:0]	方向	输入脚
00	输出	
01	输入	IC5映射在TI5FP5上
10	输入	IC5映射在TI6FP5上
11	输入	IC5映射在TRC上。此模式仅工作在内部触发器输入被选中时（由PWM5_SMCR寄存器的TS位选择）

注：CC1S 仅在通道关闭时（PWMA_CCER1 寄存器的 CC1E=0）才是可写的。

注：CC5S 仅在通道关闭时（PWMB_CCER1 寄存器的 CC5E=0）才是可写的。

PWMA_CCMR1 寄存器设置示例代码：

```
PWMA_CCER1 = 0x00;                  //必须关闭 CC1E 才能设置 CC1S
PWMA_CCMR1 = 0x01;                  //配置 CC1 为输入模式
PWMA_CCER1 |= 0x01;                 //设置 CC1E 为 1,使能 CC1 通道
PWMA_CCMR1 |= 0x04;                 //设置 IC1PSC, IC1PSC 必须在 CC1E 为 1 时才可写入
```

PWMA_CCMR2/PWMA_CCMR3/PWMA_CCMR4 以及 PWMB 组的
PWMB_CCMR1/PWMB_CCMR2/PWMB_CCMR3/PWMB_CCMR4 的设置方法与上面示例代码类似

25.9.13 捕获/比较模式寄存器 2（PWMx_CCMR2）

通道配置为比较输出模式

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCMR2	FEC9H	OC2CE	OC2M[2:0]			OC2PE	OC2FE	CC2S[1:0]	
PWMB_CCMR2	FEE9H	OC6CE	OC6M[2:0]			OC6PE	OC6FE	CC6S[1:0]	

OCnCE：输出比较 n 清零使能。该位用于使能使用 PWMETI 引脚上的外部事件来清通道 n 的输出信号（OCnREF）（n=2,6）

0：OCnREF 不受 ETRF 输入的影响；

1：一旦检测到 ETRF 输入高电平，OCnREF=0。

OCnM[2:0]：输出比较 2 模式，参考 OC1M。（n=2,6）

OCnPE：输出比较 2 预装载使能，参考 OC1PE。（n=2,6）

OCnFE：输出比较 n 快速使能，参考 OC1FE。（n=2,6）

CC2S[1:0]：捕获/比较 2 选择。这两位定义通道的方向（输入/输出），及输入脚的选择

CC2S[1:0]	方向	输入脚
00	输出	
01	输入	IC2映射在TI2FP2上
10	输入	IC2映射在TI1FP2上
11	输入	IC2映射在TRC上。

CC6S[1:0]：捕获/比较 6 选择。这两位定义通道的方向（输入/输出），及输入脚的选择

CC6S[1:0]	方向	输入脚
00	输出	
01	输入	IC6映射在TI6FP6上
10	输入	IC6映射在TI5FP6上
11	输入	IC6映射在TRC上。

通道配置为捕获输入模式

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCMR2	FEC9H	IC2F[3:0]				IC2PSC[1:0]		CC2S[1:0]	
PWMB_CCMR2	FEE9H	IC6F[3:0]				IC6PSC[1:0]		CC6S[1:0]	

ICnF[3:0]：输入捕获 n 滤波器选择，参考 IC1F。（n=2,6）

ICnPSC[1:0]：输入/捕获 n 预分频器，参考 IC1PSC。（n=2,6）

CC2S[1:0]：捕获/比较 2 选择。这两位定义通道的方向（输入/输出），及输入脚的选择

CC2S[1:0]	方向	输入脚
00	输出	
01	输入	IC2映射在TI2FP2上
10	输入	IC2映射在TI1FP2上
11	输入	IC2映射在TRC上。

CC6S[1:0]：捕获/比较 6 选择。这两位定义通道的方向（输入/输出），及输入脚的选择

CC6S[1:0]	方向	输入脚
00	输出	
01	输入	IC6映射在TI6FP6上
10	输入	IC6映射在TI5FP6上
11	输入	IC6映射在TRC上。

25.9.14 捕获/比较模式寄存器 3（PWMx_CCMR3）

通道配置为比较输出模式

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCMR3	FECAH	OC3CE	OC3M[2:0]			OC3PE	OC3FE	CC3S[1:0]	
PWMB_CCMR3	FEEAH	OC7CE	OC7M[2:0]			OC7PE	OC7FE	CC7S[1:0]	

OCnCE：输出比较 n 清零使能。该位用于使能使用 PWMETI 引脚上的外部事件来清通道 n 的输出信号（OCnREF）（n=3,7）

0：OCnREF 不受 ETRF 输入的影响；

1：一旦检测到 ETRF 输入高电平，OCnREF=0。

OCnM[2:0]：输出比较 3 模式，参考 OC1M。（n=3,7）

OCnPE：输出比较 3 预装载使能，参考 OP1PE。（n=3,7）

OCnFE：输出比较 n 快速使能，参考 OC1FE。（n=3,7）

CC3S[1:0]：捕获/比较 3 选择。这两位定义通道的方向（输入/输出），及输入脚的选择

CC3S[1:0]	方向	输入脚
00	输出	
01	输入	IC3映射在TI3FP3上
10	输入	IC3映射在TI4FP3上
11	输入	IC3映射在TRC上。

CC7S[1:0]：捕获/比较 7 选择。这两位定义通道的方向（输入/输出），及输入脚的选择

CC7S[1:0]	方向	输入脚
00	输出	
01	输入	IC7映射在TI7FP7上
10	输入	IC7映射在TI8FP7上
11	输入	IC7映射在TRC上。

通道配置为捕获输入模式

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCMR3	FECAH	IC3F[3:0]				IC3PSC[1:0]		CC3S[1:0]	
PWMB_CCMR3	FEEAH	IC7F[3:0]				IC7PSC[1:0]		CC7S[1:0]	

ICnF[3:0]：输入捕获 n 滤波器选择，参考 IC1F。（n=3,7）

ICnPSC[1:0]：输入/捕获 n 预分频器，参考 IC1PSC。（n=3,7）

CC3S[1:0]：捕获/比较 3 选择。这两位定义通道的方向（输入/输出），及输入脚的选择

CC3S[1:0]	方向	输入脚
00	输出	
01	输入	IC3映射在TI3FP3上
10	输入	IC3映射在TI4FP3上
11	输入	IC3映射在TRC上。

CC7S[1:0]：捕获/比较 7 选择。这两位定义通道的方向（输入/输出），及输入脚的选择

CC7S[1:0]	方向	输入脚
00	输出	
01	输入	IC7映射在TI7FP7上
10	输入	IC7映射在TI8FP7上
11	输入	IC7映射在TRC上。

25.9.15 捕获/比较模式寄存器 4（PWMx_CCMR4）

通道配置为比较输出模式

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCMR4	FECBH	OC4CE	OC4M[2:0]			OC4PE	OC4FE	CC4S[1:0]	
PWMB_CCMR4	FEEBH	OC8CE	OC8M[2:0]			OC8PE	OC8FE	CC8S[1:0]	

OCnCE：输出比较 n 清零使能。该位用于使能使用 PWMETI 引脚上的外部事件来清通道 n 的输出信号（OCnREF）（n=4,8）

0：OCnREF 不受 ETRF 输入的影响；

1：一旦检测到 ETRF 输入高电平，OCnREF=0。

OCnM[2:0]：输出比较 n 模式，参考 OC1M。（n=4,8）

OCnPE：输出比较 n 预装载使能，参考 OP1PE。（n=4,8）

OCnFE：输出比较 n 快速使能，参考 OC1FE。（n=4,8）

CC4S[1:0]：捕获/比较 4 选择。这两位定义通道的方向（输入/输出），及输入脚的选择

CC4S[1:0]	方向	输入脚
00	输出	
01	输入	IC4映射在TI4FP4上
10	输入	IC4映射在TI3FP4上
11	输入	IC4映射在TRC上。

CC8S[1:0]：捕获/比较 8 选择。这两位定义通道的方向（输入/输出），及输入脚的选择

CC8S[1:0]	方向	输入脚
00	输出	
01	输入	IC8映射在TI8FP8上
10	输入	IC8映射在TI7FP8上
11	输入	IC8映射在TRC上。

通道配置为捕获输入模式

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCMR4	FECBH	IC4F[3:0]				IC4PSC[1:0]		CC4S[1:0]	
PWMB_CCMR4	FEEBH	IC8F[3:0]				IC8PSC[1:0]		CC8S[1:0]	

ICnF[3:0]：输入捕获 n 滤波器选择，参考 IC1F。（n=4,8）

ICnPSC[1:0]：输入/捕获 n 预分频器，参考 IC1PSC。（n=4,8）

CC4S[1:0]：捕获/比较 4 选择。这两位定义通道的方向（输入/输出），及输入脚的选择

CC4S[1:0]	方向	输入脚
00	输出	
01	输入	IC4映射在TI4FP4上
10	输入	IC4映射在TI3FP4上
11	输入	IC4映射在TRC上。

CC8S[1:0]：捕获/比较 8 选择。这两位定义通道的方向（输入/输出），及输入脚的选择

CC8S[1:0]	方向	输入脚
00	输出	
01	输入	IC8映射在TI8FP8上
10	输入	IC8映射在TI7FP8上
11	输入	IC8映射在TRC上。

25.9.16 捕获/比较使能寄存器 1（PWMx_CCER1）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCER1	FECCH	CC2NP	CC2NE	CC2P	CC2E	CC1NP	CC1NE	CC1P	CC1E
PWMB_CCER1	FEECH	-	-	CC6P	CC6E	-	-	CC5P	CC5E

CC6P：OC6 输入捕获/比较输出极性。参考 CC1P

CC6E：OC6 输入捕获/比较输出使能。参考 CC1E

CC5P：OC5 输入捕获/比较输出极性。参考 CC1P

CC5E：OC5 输入捕获/比较输出使能。参考 CC1E

CC2NP：OC2N（OC2 通道负极）比较输出极性。参考 CC1NP

CC2NE：OC2N（OC2 通道负极）比较输出使能。参考 CC1NE

CC2P：OC2（OC2 通道正极）输入捕获/比较输出极性。参考 CC1P

CC2E：OC2（OC2 通道正极）输入捕获/比较输出使能。参考 CC1E

CC1NP：OC1N（OC1 通道负极）比较输出极性

0：高电平有效；

1：低电平有效。

注 1：一旦 LOCK 级别（PWMA_BKR 寄存器中的 LOCK 位）设为 3 或 2 且 CC1S=00（通道配置为输出），则该位不能被修改。

注 2：对于有互补输出的通道，该位是预装载的。如果 CCPC=1（PWMA_CR2 寄存器），只有在 COM 事件发生时，CC1NP 位才从预装载位中取新值。

CC1NE：OC1N（OC1 通道负极）比较输出使能

0：关闭比较输出。

1：开启比较输出，其输出电平依赖于 MOE、OSSI、OSSR、OIS1、OIS1N 和 CC1E 位的值。

注：对于有互补输出的通道，该位是预装载的。如果 CCPC=1（PWMA_CR2 寄存器），只有在 COM 事件发生时，CC1NE 位才从预装载位中取新值。

CC1P：OC1（OC1 通道正极）输入捕获/比较输出极性

CC1 通道配置为输出：

0：高电平有效

1：低电平有效

CC1 通道配置为输入或者捕获：

0：捕获发生在 TI1F 或 TI2F 的上升沿；

1：捕获发生在 TI1F 或 TI2F 的下降沿.

注 1：一旦 LOCK 级别（PWMA_BKR 寄存器中的 LOCK 位）设为 3 或 2，则该位不能被修改。

注 2：对于有互补输出的通道，该位是预装载的。如果 CCPC=1（PWMA_CR2 寄存器），只有在 COM 事件发生时，CC1P 位才从预装载位中取新值。

CC1E：OC1（OC1 通道正极）输入捕获/比较输出使能

0：关闭输入捕获/比较输出；

1：开启输入捕获/比较输出。

注：对于有互补输出的通道，该位是预装载的。如果 CCPC=1（PWMA_CR2 寄存器），只有在 COM 事件发生时，CC1E 位才从预装载位中取新值。

带刹车功能的互补输出通道 OCi 和 OCiN 的控制位

控制位					输出状态	
MOE	OSSI	OSSR	CCiE	CCiNE	OCi 输出状态	OCiN 输出状态
1	X	0	0	0	输出禁止	输出禁止
		0	0	1	输出禁止	带极性的 OCiREF
		0	1	0	带极性的 OCiREF	输出禁止
		0	1	1	带极性和死区的 OCiREF	带极性和死区的反向 OCiREF
		1	0	0	输出禁止	输出禁止
		1	0	1	关闭状态 （输出使能且为无效电平） OCi=CCiP	带极性的 OCiREF
		1	1	0	带极性的 OCiREF	关闭状态 （输出使能且为无效电平） OCiN=CCiNP
		1	1	1	带极性和死区的 OCiREF	带极性和死区的反向 OCiREF
0	0	X	X	X	输出禁止	
	1				关闭状态（输出使能且为无效电平）异步地：OCi=CCiP，OCiN=CCiNP；然后，若时钟存在：经过一个死区时间后 OCi=OISi，OCiN=OISiN，假设 OISi 与 OISiN 并不都对应 OCi 和 OCiN 的有效电平。	

注：管脚连接到互补的 OCi 和 OCiN 通道的外部 I/O 管脚的状态，取决于 OCi 和 OCiN 通道状态和 GPIO 寄存器。

25.9.17 捕获/比较使能寄存器 2（PWMx_CCER2）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCER2	FECDH	CC4NP	CC4NE	CC4P	CC4E	CC3NP	CC3NE	CC3P	CC3E
PWMB_CCER2	FEEDH	-	-	CC8P	CC8E	-	-	CC7P	CC7E

CC8P：OC8 输入捕获/比较输出极性。参考 CC1P
CC8E：OC8 输入捕获/比较输出使能。参考 CC1E
CC7P：OC7 输入捕获/比较输出极性。参考 CC1P
CC7E：OC7 输入捕获/比较输出使能。参考 CC1E
CC4NP：OC4N（OC4 通道负极）比较输出极性。参考 CC1NP
CC4NE：OC4N（OC4 通道负极）比较输出使能。参考 CC1NE
CC4P：OC4（OC4 通道正极）输入捕获/比较输出极性。参考 CC1P
CC4E：OC4（OC4 通道正极）输入捕获/比较输出使能。参考 CC1E
CC3NP：OC3N（OC3 通道负极）比较输出极性。参考 CC1NP
CC3NE：OC3N（OC3 通道负极）比较输出使能。参考 CC1NE
CC3P：OC3（OC3 通道正极）输入捕获/比较输出极性。参考 CC1P
CC3E：OC3（OC3 通道正极）输入捕获/比较输出使能。参考 CC1E

25.9.18 计数器高 8 位（PWMx_CNTRH）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CNTRH	FECEH	CNT1[15:8]							
PWMB_CNTRH	FEEEH	CNT2[15:8]							

CNTn[15:8]：计数器的高 8 位值（n= A,B）

25.9.19 计数器低 8 位（PWMx_CNTRL）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CNTRL	FECFH	CNT1[7:0]							
PWMB_CNTRL	FEEFH	CNT2[7:0]							

CNTn[7:0]：计数器的低 8 位值（n= A,B）

25.9.20 预分频器高 8 位（PWMx_PSCRH），输出频率计算公式

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_PSCRH	FED0H	PSC1[15:8]							
PWMB_PSCRH	FEF0H	PSC2[15:8]							

PSCn[15:8]：预分频器的高 8 位值。（n= A,B）

预分频器用于对 CK_PSC 进行分频。计数器的时钟频率（fCK_CNT）等于 fCK_PSC/（PSCR[15:0]+1）。

PSCR 包含了当更新事件产生时装入当前预分频器寄存器的值（更新事件包括计数器被 TIM_EGR 的 UG 位清 0 或被工作在复位模式的从控制器清 0）。这意味着为了使新的值起作用，必须产生一个更新事件。

PWM 输出频率计算公式

PWMA 和 PWMB 两组 PWM 的输出频率计算公式相同，且每组可设置不同的频率。

对齐模式	PWM输出频率计算公式
边沿对齐	$\text{PWM输出频率} = \dfrac{\text{系统工作频率SYSclk}}{(\text{PWMx\_PSCR} + 1) \times (\text{PWMx\_ARR} + 1)}$
中间对齐	$\text{PWM输出频率} = \dfrac{\text{系统工作频率SYSclk}}{(\text{PWMx\_PSCR} + 1) \times \text{PWMx\_ARR} \times 2}$

25.9.21 预分频器低 8 位（PWMx_PSCRL）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_PSCRL	FED1H	PSC1[7:0]							
PWMB_PSCRL	FEF1H	PSC2[7:0]							

PSCn[7:0]：预分频器的低 8 位值。（n= A,B）

25.9.22 自动重装载寄存器高 8 位（PWMx_ARRH）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_ARRH	FED2H	ARR1[15:8]							
PWMB_ARRH	FEF2H	ARR2[15:8]							

ARRn[15:8]：自动重装载高 8 位值（n= A,B）

ARR 包含了将要装载入实际的自动重装载寄存器的值。当自动重装载的值为 0 时，计数器不工作。

25.9.23 自动重装载寄存器低 8 位（PWMx_ARRL）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_ARRL	FED3H	ARR1[7:0]							
PWMB_ARRL	FEF3H	ARR2[7:0]							

ARRn[7:0]：自动重装载低 8 位值（n= A,B）

25.9.24 重复计数器寄存器（PWMx_RCR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_RCR	FED4H	REP1[7:0]							
PWMB_RCR	FEF4H	REP2[7:0]							

REPn[7:0]：重复计数器值（n= A,B）

开启了预装载功能后，这些位允许用户设置比较寄存器的更新速率（即周期性地从预装载寄存器 传输到当前寄存器）；如果允许产生更新中断，则会同时影响产生更新中断的速率。每次向下计数器 REP_CNT 达到 0，会产生一个更新事件并且计数器 REP_CNT 重新从 REP 值开 始计数。由于 REP_CNT 只有在周期更新事件 U_RC 发生时才重载 REP 值，因此对 PWMn_RCR 寄存器写入的新值只在下次周期更新事件发生时才起作用。这意味着在 PWM 模式中，（REP+1）对应着：

- 在边沿对齐模式下，PWM 周期的数目；
- 在中心对称模式下，PWM 半周期的数目；

25.9.25 捕获/比较寄存器 1/5 高 8 位（PWMx_CCR1H）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCR1H	FED5H	CCR1[15:8]							
PWMB_CCR5H	FEF5H	CCR5[15:8]							

CCRn[15:8]：捕获/比较 n 的高 8 位值（n=1,5）

若 CCn 通道配置为输出：CCRn 包含了装入当前比较值（预装载值）。如果在 PWMn_CCMR1 寄存器（OCnPE 位）中未选择预装载功能，写入的数值会立即传输至当前寄存器中。否则只有当更新事件发生时，此预装载值才传输至当前捕获/比较 n 寄存器中。当前比较值同计数器 PWMn_CNT 的值相比较，并在 OCn 端口上产生输出信号。

若 CCn 通道配置为输入：CCRn 包含了上一次输入捕获事件发生时的计数器值（此时该寄存器为只读）。

25.9.26 捕获/比较寄存器 1/5 低 8 位（PWMx_CCR1L）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCR1L	FED6H	CCR1[7:0]							
PWMB_CCR5L	FEF6H	CCR5[7:0]							

CCRn[7:0]：捕获/比较 n 的低 8 位值（n=1,5）

25.9.27 捕获/比较寄存器 2/6 高 8 位（PWMx_CCR2H）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCR2H	FED7H	CCR2[15:8]							
PWMB_CCR6H	FEF7H	CCR6[15:8]							

CCRn[15:8]：捕获/比较 n 的高 8 位值（n=2,6）

25.9.28 捕获/比较寄存器 2/6 低 8 位（PWMx_CCR2L）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCR2L	FED8H	CCR2[7:0]							
PWMB_CCR6L	FEF8H	CCR6[7:0]							

CCRn[7:0]：捕获/比较 n 的低 8 位值（n=2,6）

25.9.29 捕获/比较寄存器 3/7 高 8 位（PWMx_CCR3H）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCR3H	FED9H	CCR3[15:8]							
PWMB_CCR7H	FEF9H	CCR7[15:8]							

CCRn[15:8]：捕获/比较 n 的高 8 位值（n=3,7）

25.9.30 捕获/比较寄存器 3/7 低 8 位（PWMx_CCR3L）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCR3L	FEDAH	CCR3[7:0]							
PWMB_CCR7L	FEFAH	CCR7[7:0]							

CCRn[7:0]：捕获/比较 n 的低 8 位值（n=3,7）

25.9.31 捕获/比较寄存器 4/8 高 8 位（PWMx_CCR4H）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCR4H	FEDBH	CCR4[15:8]							
PWMB_CCR8H	FEFBH	CCR8[15:8]							

CCRn[15:8]：捕获/比较 n 的高 8 位值（n=4,8）

25.9.32 捕获/比较寄存器 4/8 低 8 位（PWMx_CCR4L）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCR4L	FEDCH	CCR4[7:0]							
PWMB_CCR8L	FEFCH	CCR8[7:0]							

CCRn[7:0]：捕获/比较 n 的低 8 位值（n=4,8）

25.9.33 刹车寄存器（PWMx_BKR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_BKR	FEDDH	MOEA	AOEA	BKPA	BKEA	OSSRA	OSSIA	LOCKA[1:0]	
PWMB_BKR	FEFDH	MOEB	AOEB	BKPB	BKEB	OSSRB	OSSIB	LOCKB[1:0]	

MOEn：主输出使能。一旦刹车输入有效，该位被硬件异步清 0。根据 AOE 位的设置值，该位可以由软件置 1 或被自动置 1。它仅对配置为输出的通道有效。（n= A,B）

0：禁止 OC 和 OCN 输出或强制为空闲状态

1：如果设置了相应的使能位（PWMn_CCERX 寄存器的 CCiE 位），则使能 OC 和 OCN 输出。

注：一旦 LOCK 级别（PWMn_BKR 寄存器中的 LOCK 位）设为 1，则该位不能被修改

AOEn：自动输出使能（n= A,B）

0：MOE 只能被软件置 1；

1：MOE 能被软件置 1 或在下一个更新事件被自动置 1（如果刹车输入无效）。

注：一旦 LOCK 级别（PWMn_BKR 寄存器中的 LOCK 位）设为 1，则该位不能被修改

BKPn：刹车输入极性（n= A,B）

0：刹车输入低电平有效

1：刹车输入高电平有效

注：一旦 LOCK 级别（PWMn_BKR 寄存器中的 LOCK 位）设为 1，则该位不能被修改

BKEn：刹车功能使能（n= A,B）

0：禁止刹车输入（BRK）

1：开启刹车输入（BRK）

注：一旦 LOCK 级别（PWMn_BKR 寄存器中的 LOCK 位）设为 1，则该位不能被修改。

OSSRn：运行模式下“关闭状态”选择。该位在 MOE=1 且通道设为输出时有效（n= A,B）

0：当 PWM 不工作时，禁止 OC/OCN 输出（OC/OCN 使能输出信号=0）；

1：当 PWM 不工作时，一旦 CCiE=1 或 CCiNE=1，首先开启 OC/OCN 并输出无效电平，然后置 OC/OCN 使能输出信号=1。

注：一旦 LOCK 级别（PWMn_BKR 寄存器中的 LOCK 位）设为 2，则该位不能被修改。

OSSIn：空闲模式下“关闭状态”选择。该位在 MOE=0 且通道设为输出时有效。（n= A,B）

0：当 PWM 不工作时，禁止 OC/OCN 输出（OC/OCN 使能输出信号=0）；

1：当 PWM 不工作时，一旦 CCiE=1 或 CCiNE=1，OC/OCN 首先输出其空闲电平，然后 OC/OCN 使能输出信号=1。

注：一旦 LOCK 级别（PWMn_BKR 寄存器中的 LOCK 位）设为 2，则该位不能被修改。

LOCKn[1:0]：锁定设置。该位为防止软件错误而提供的写保护措施（n= A,B）

LOCKn[1:0]	保护级别	保护内容
00	无保护	寄存器无写保护
01	锁定级别1	不能写入PWMn_BKR寄存器的MOE、BKE、BKP、AOE位和PWMn_OISR寄存器的OISI位
10	锁定级别2	不能写入锁定级别1中的各位， 也不能写入CC极性位以及OSSR/OSSI位
11	锁定级别3	不能写入锁定级别2中的各位， 也不能写入CC控制位

注：由于 MOE、BKE、BKP、AOE、OSSR、OSSI 位可被锁定（依赖于 LOCK 位），因此在第一次写 PWMn_BKR 寄存器时必须对它们进行设置。

25.9.34 死区寄存器（PWMx_DTR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_DTR	FEDEH	DTGA[7:0]							
PWMB_DTR	FEFEH	DTGB[7:0]							

DTGn[7:0]：死区发生器设置。（n= A,B）

这些位定义了插入互补输出之间的死区持续时间。（t_{CK_PSC} 为 PWMn 的时钟脉冲）

DTGn[7:5]	死区时间
000	DTGn[7:0] * t_{CK_PSC}
001	
010	
011	
100	(64 + DTGn[6:0]) * 2 * t_{CK_PSC}
101	
110	(32 + DTGn[5:0]) * 8 * t_{CK_PSC}
111	(32 + DTGn[4:0]) * 16 * t_{CK_PSC}

注：一旦 LOCK 级别（PWMx_BKR 寄存器中的 LOCK 位）设为 1、2 或 3 时，则该位不能被修改。

25.9.35 输出空闲状态寄存器（PWMx_OISR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_OISR	FEDFH	OIS4N	OIS4	OIS3N	OIS3	OIS2N	OIS2	OIS1N	OIS1
PWMB_OISR	FEFFH	-	OIS8	-	OIS7	-	OIS6	-	OIS5

OIS8：空闲状态时 OC8 输出电平

OIS7：空闲状态时 OC7 输出电平

OIS6：空闲状态时 OC6 输出电平

OIS5：空闲状态时 OC5 输出电平

OIS4N：空闲状态时 OC4N 输出电平

OIS4：空闲状态时 OC4 输出电平

OIS3N：空闲状态时 OC3N 输出电平

OIS3：空闲状态时 OC3 输出电平

OIS2N：空闲状态时 OC2N 输出电平

OIS2：空闲状态时 OC2 输出电平

OIS1N：空闲状态时 OC1N 输出电平

0：当 MOE=0 时，则在一个死区时间后，OC1N=0；

1：当 MOE=0 时，则在一个死区时间后，OC1N=1。

注：一旦 LOCK 级别（PWMx_BKR 寄存器中的 LOCK 位）设为 1、2 或 3 时，则该位不能被修改。

OIS1：空闲状态时 OC1 输出电平

0：当 MOE=0 时，如果 OC1N 使能，则在一个死区后，OC1=0；

1：当 MOE=0 时，如果 OC1N 使能，则在一个死区后，OC1=1。

注：一旦 LOCK 级别（PWMx_BKR 寄存器中的 LOCK 位）设为 1、2 或 3 时，则该位不能被修改。

25.10 范例程序

25.10.1 PWMA+PWMB 实现 8 组定时器

```
//测试工作频率为 12MHz

/************ 功能说明 **************
本例程基于 STC8H8K64U 为主控芯片的实验箱 9 进行编写测试，STC8H 系列芯片可通用参考.
利用高级 PWMA+PWMB 中断实现 8 组定时器功能.

PWMA 的时钟频率为 SYSclk/2
PWMA 通道 1：定时周期 1ms
PWMA 通道 2：定时周期 2ms
PWMA 通道 3：定时周期 4ms
PWMA 通道 4：定时周期 5ms

PWMB 的时钟频率为 SYSclk/10000
PWMB 通道 1：定时周期 1s
PWMB 通道 2：定时周期 2s
PWMB 通道 3：定时周期 3s
PWMB 通道 4：定时周期 4s

下载时，选择时钟 24MHZ (用户可自行修改频率).
*****************************************/

#include   "stc8h.h"
#include   "intrins.h"

#define    MAIN_Fosc     24000000L                  //定义主时钟

#define    PSCRA         2
#define    PWMA_T1       MAIN_Fosc/PSCRA/1000       //定时周期 1ms(1KHz)
#define    PWMA_T2       MAIN_Fosc/PSCRA/500        //定时周期 2ms(500Hz)
#define    PWMA_T3       MAIN_Fosc/PSCRA/250        //定时周期 4ms(250Hz)
#define    PWMA_T4       MAIN_Fosc/PSCRA/200        //定时周期 5ms(200Hz)

#define    PSCRB         10000
#define    PWMB_T5       MAIN_Fosc/PSCRB*1          //定时周期 1s
#define    PWMB_T6       MAIN_Fosc/PSCRB*2          //定时周期 2s
#define    PWMB_T7       MAIN_Fosc/PSCRB*3          //定时周期 3s
#define    PWMB_T8       MAIN_Fosc/PSCRB*4          //定时周期 4s

typedef    unsigned char    u8;
typedef    unsigned int     u16;
typedef    unsigned long    u32;

/*****************************************************************************/

void PWMA_Timer();
void PWMB_Timer();

u16 cnt1, cnt2, cnt3, cnt4, cnt5, cnt6, cnt7, cnt8;

/******************** 主函数 **************************/
void main(void)
{
```

```
    P_SW2 |= 0x80;                                  //扩展寄存器(XFR)访问使能

    P0M1 = 0x00;    P0M0 = 0x00;                    //设置为准双向口
    P1M1 = 0x00;    P1M0= 0x00;                     //设置为准双向口
    P2M1 = 0x00;    P2M0= 0x00;                     //设置为准双向口
    P3M1 = 0x00;    P3M0= 0x00;                     //设置为准双向口
    P4M1 = 0x00;    P4M0= 0x00;                     //设置为准双向口
    P5M1 = 0x00;    P5M0= 0x00;                     //设置为准双向口
    P6M1 = 0x00;    P6M0= 0x00;                     //设置为准双向口
    P7M1 = 0x00;    P7M0= 0x00;                     //设置为准双向口

    PWMA_Timer();
    PWMB_Timer();

    P40  = 0;                                       //给 LED 供电
    EA = 1;                                         //打开总中断

    while (1);
}

//========================================================================
// 函数: void PWMA_Timer()
// 描述: PWMA 配置函数.
//========================================================================
void PWMA_Timer()
{
    PWMA_PSCR =PSCRA-1;                             //设置预分频器
    PWMA_ARR = 0xffff;
    PWMA_CCER1 = 0x00;
    PWMA_CCMR1 = 0x00;
    PWMA_CCMR2 = 0x00;
    PWMA_CCMR3 = 0x00;
    PWMA_CCMR4 = 0x00;
    cnt1 = PWMA_CCR1 = PWMA_T1;
    cnt2 = PWMA_CCR2 = PWMA_T2;
    cnt3 = PWMA_CCR3 = PWMA_T3;
    cnt4 = PWMA_CCR4 = PWMA_T4;
    PWMA_IER = 0x1e;                                // 使能中断
    PWMA_CR1 |= 0x81;                               //使能 ARR 预装载，开始计时

    cnt1 += PWMA_T1;
    cnt2 += PWMA_T2;
    cnt3 += PWMA_T3;
    cnt4 += PWMA_T4;
}

//========================================================================
// 函数: void PWMB_Timer()
// 描述: PWMB 配置函数.
//========================================================================
void PWMB_Timer()
{
    PWMB_PSCR =PSCRB-1;                             //设置预分频器
    PWMB_ARR = 0xffff;
    PWMB_CCER1 = 0x00;
    PWMB_CCMR1 = 0x00;
    PWMB_CCMR2 = 0x00;
    PWMB_CCMR3 = 0x00;
    PWMB_CCMR4 = 0x00;
```

```
    cnt5 = PWMB_CCR5 = PWMB_T5;
    cnt6 = PWMB_CCR6 = PWMB_T6;
    cnt7 = PWMB_CCR7 = PWMB_T7;
    cnt8 = PWMB_CCR8 = PWMB_T8;
    PWMB_IER = 0x1e;                                //  使能中断
    PWMB_CR1 |= 0x81;                               //使能 ARR 预装载，开始计时

    cnt5 += PWMB_T5;
    cnt6 += PWMB_T6;
    cnt7 += PWMB_T7;
    cnt8 += PWMB_T8;
}

//========================================================================
//  函数: PWMA_ISR()      interrupt 26
//  描述: PWMA 中断函数.
//========================================================================
void PWMA_ISR()     interrupt 26
{
    u8 sr;

    sr = PWMA_SR1;
    PWMA_SR1 = 0;

    if (sr & 0x02)
    {
        PWMA_CCR1 =cnt1;                            //更新比较值
        cnt1 += PWMA_T1;                            //计算下一个比较值
        P60 = ~P60;
    }
    if (sr & 0x04)
    {
        PWMA_CCR2 =cnt2;                            //更新比较值
        cnt2 += PWMA_T2;                            //计算下一个比较值
        P61 = ~P61;
    }
    if (sr & 0x08)
    {
        PWMA_CCR3 =cnt3;                            //更新比较值
        cnt3 += PWMA_T3;                            //计算下一个比较值
        P62 = ~P62;
    }
    if (sr & 0x10)
    {
        PWMA_CCR4 =cnt4;                            //更新比较值
        cnt4 += PWMA_T4;                            //计算下一个比较值
        P63 = ~P63;
    }
}

//========================================================================
//  函数: PWMB_ISR()      interrupt 27
//  描述: PWMB 中断函数.
//========================================================================
void PWMB_ISR()     interrupt 27
{
    u8 sr;

    sr = PWMB_SR1;
```

```
    PWMB_SR1 = 0;

    if (sr & 0x02)
    {
        PWMB_CCR5 = cnt5;                   //更新比较值
        cnt5 +=   PWMB_T5;                  //计算下一个比较值
        P24  = ~P24;
        P64  = ~P64;
    }
    if (sr & 0x04)
    {
        PWMB_CCR6 = cnt6;                   //更新比较值
        cnt6 +=   PWMB_T6;                  //计算下一个比较值
        P25  = ~P25;
        P65  = ~P65;
    }
    if (sr & 0x08)
    {
        PWMB_CCR7 = cnt7;                   //更新比较值
        cnt7 +=   PWMB_T7;                  //计算下一个比较值
        P26  = ~P26;
        P66  = ~P66;
    }
    if (sr & 0x10)
    {
        PWMB_CCR8 = cnt8;                   //更新比较值
        cnt8 +=   PWMB_T8;                  //计算下一个比较值
        P27  = ~P27;
        P67  = ~P67;
    }
}
```

25.10.2 高级 PWM 时钟输出应用（系统时钟 2 分频输出）

```
//测试工作频率为 24MHz

#include "stc8h.h"

#define   FOSC              24000000UL
#define   PWM_PERIOD        (2-1)           //定义 PWM 周期值
                                            //(频率=FOSC/(PWM_PERIOD+1)=12MHz)
#define   PWM_DUTY          (1)             //定义 PWM 的占空比值
                                            //(占空比=PWM_DUTY/PWM_PERIOD*100%=50%)

void SYS_Init();
void PWM_Init();

void main()
{
    SYS_Init();
    PWM_Init();

    while (1);
}

void SYS_Init()
```

```
{
    P_SW2 |= 0x80;                          //扩展寄存器(XFR)访问使能

    P0M1 = 0x00;    P0M0 = 0x00;
    P1M1 = 0x00;    P1M0 = 0x00;
    P2M1 = 0x00;    P2M0 = 0x00;
    P3M1 = 0x00;    P3M0 = 0x00;
    P4M1 = 0x00;    P4M0 = 0x00;
    P5M1 = 0x00;    P5M0 = 0x00;
    P6M1 = 0x00;    P6M0 = 0x00;
    P7M1 = 0x00;    P7M0 = 0x00;
}

void PWM_Init()
{
    PWMA_CCER1 = 0x00;                      //写 CCMRx 前必须先清零 CCxE 关闭通道
    PWMA_CCMR1 = 0x60;                      //通道模式配置 PWM 模式 1
    PWMA_CCER1 = 0x01;                      //配置通道输出使能和极性
    PWMA_ARR = PWM_PERIOD;                  //设置周期时间
    PWMA_CCR1 = PWM_DUTY;                   //设置占空比时间
    PWMA_ENO = 0x01;                        //使能 PWM 输出
    PWMA_BKR = 0x80;                        //使能主输出
    PWMA_CR1 = 0x01;                        //开始计时
}
```

25.10.3 输出任意周期和任意占空比的波形

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "stc8h.h"
#include "intrins.h"

void main()
{
    P_SW2 |= 0x80;                          //使能访问 XFR，没有冲突不用关闭

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    PWMA_CCER1 = 0x00;                      //写 CCMRx 前必须先清零 CCERx 关闭通道
    PWMA_CCMR1 = 0x60;                      //设置 CC1 为 PWMA 输出模式
    PWMA_CCER1 = 0x01;                      //使能 CC1 通道
    PWMA_CCR1 = 100;                        //设置占空比时间
    PWMA_ARR = 500;                         //设置周期时间
    PWMA_ENO = 0x01;                        //使能 PWM1P 端口输出
    PWMA_BKR = 0x80;                        //使能主输出
```

```
    PWMA_CR1 = 0x01;                                //开始计时

    while (1);
}
```

25.10.4 输出占空比为 100%和 0%的 PWM 波形的方法（以 PWM1P 为例）

25.10.4.1 方法 1：设置 PWMx_ENO 禁止输出 PWM

C 语言代码

```
//测试工作频率为 11.0592MHz
#include "stc8h.h"
#include "intrins.h"

void main()
{
    P_SW2 |= 0x80;                                  //使能访问 XFR，没有冲突不用关闭

    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    PWMA_ENO &= ~0x01;                              //禁止 PWM1P 端口输出
                                                    //此时 PWM1P 端口为 GPIO，可通过
                                                    //直接操作 I/O 口寄存器强制输出高电平或者低电平

    while (1);
}
```

25.10.4.2 方法 2：设置 PWMx_CCMRn 寄存器强制输出有效/无效电平

C 语言代码

```
//测试工作频率为 11.0592MHz
#include "stc8h.h"
#include "intrins.h"

void main()
{
    P_SW2 |= 0x80;                                  //使能访问 XFR，没有冲突不用关闭

    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;
```

```
    PWMA_CCER1 = 0x00;
    PWMA_CCMR1 &= ~0x03;                    //CC1 为输出模式
    PWMA_CCMR1 |= 0x40;                     //CC1 强制输出无效电平（占空比 0%）
//  PWMA_CCMR1 |= 0x50;                     //CC1 强制输出有效电平（占空比 100%）
    PWMA_CCER1 |= 0x01;                     //使能 CC1 输出，且设置高电平为有效电平
    PWMA_ENO = 0x01;                        //使能 PWM1P 端口输出
    PWMA_BKR = 0x80;                        //使能主输出
    PWMA_CR1 = 0x01;                        //开始计时

    while (1);
}
```

25.10.5 高级 PWM 输出-频率可调-脉冲计数（软件方式）

C 语言代码

```
//测试工作频率为 24MHz

/*************  功能说明  **************
本例程基于 STC8H8K64U 为主控芯片的实验箱 8 进行编写测试，STC8H 系列芯片可通用参考.
高级 PWM 定时器实现高速 PWM 脉冲输出.
周期/占空比可调, 通过比较/捕获中断进行脉冲个数计数.
通过 P6 口演示输出,每隔 10ms 输出一次 PWM,计数 10 个脉冲后停止输出.
定时器每 1ms 调整 PWM 周期.
下载时,选择时钟 24MHZ (用户可自行修改频率).
*****************************************/

#include   "stc8h.h"
#include   "intrins.h"

#define    MAIN_Fosc        24000000L

typedef    unsigned char    u8;
typedef    unsigned int     u16;
typedef    unsigned long    u32;

/*****************************用户定义宏 ***********************************/

#define    Timer0_Reload    (65536UL -(MAIN_Fosc / 1000))        //Timer0 中断频率,1000 次/秒

#define    PWM1_1           0x00                 //P:P1.0   N:P1.1
#define    PWM1_2           0x01                 //P:P2.0   N:P2.1
#define    PWM1_3           0x02                 //P:P6.0   N:P6.1

#define    PWM2_1           0x00                 //P:P1.2   N:P1.3
#define    PWM2_2           0x04                 //P:P2.2   N:P2.3
#define    PWM2_3           0x08                 //P:P6.2   N:P6.3

#define    PWM3_1           0x00                 //P:P1.4   N:P1.5
#define    PWM3_2           0x10                 //P:P2.4   N:P2.5
#define    PWM3_3           0x20                 //P:P6.4   N:P6.5

#define    PWM4_1           0x00                 //P:P1.6   N:P1.7
#define    PWM4_2           0x40                 //P:P2.6   N:P2.7
#define    PWM4_3           0x80                 //P:P6.6   N:P6.7
```

```
#define  PWM4_4          0xC0                    //P:P3.4   N:P3.3

#define  ENO1P           0x01
#define  ENO1N           0x02
#define  ENO2P           0x04
#define  ENO2N           0x08
#define  ENO3P           0x10
#define  ENO3N           0x20
#define  ENO4P           0x40
#define  ENO4N           0x80

/*************本地变量声明**************/
bit    B_1ms;                                    //1ms 标志
bit    PWM1_Flag;

u16   Period;
u8 Counter;
u8 msSecond;

void UpdatePwm(void);
void TxPulse(void);

/********************* 主函数 *************************/
void main(void)
{
    P_SW2 |= 0x80;                               //使能 XFR 访问

    P0M1 = 0x00;    P0M0 = 0x00;                 //设置为准双向口
    P1M1 = 0x00;    P1M0 = 0x00;                 //设置为准双向口
    P2M1 = 0x00;    P2M0 = 0x00;                 //设置为准双向口
    P3M1 = 0x00;    P3M0 = 0x00;                 //设置为准双向口
    P4M1 = 0x00;    P4M0 = 0x00;                 //设置为准双向口
    P5M1 = 0x00;    P5M0 = 0x00;                 //设置为准双向口
    P6M1 = 0x00;    P6M0 = 0x00;                 //设置为准双向口
    P7M1 = 0x00;    P7M0 = 0x00;                 //设置为准双向口

    PWM1_Flag = 0;
    Counter = 0;
    Period = 0x1000;

    //Timer0 初始化
    AUXR = 0x80;                                 //Timer0 set as 1T,16 bits timer auto-reload,
    TH0 = (u8)(Timer0_Reload / 256);
    TL0 = (u8)(Timer0_Reload % 256);
    ET0 = 1;                                     //Timer0 interrupt enable
    TR0 = 1;                                     //Tiner0 run

    PWMA_ENO = 0x00;
    PWMA_ENO |= ENO1P;                           //使能输出

    PWMA_PS = 0x00;                              //高级 PWM 通道输出脚选择位
    PWMA_PS |= PWM1_3;                           //选择 PWM1_3 通道

    UpdatePwm();
    PWMA_BKR = 0x80;                             //使能主输出
    PWMA_CR1 |= 0x01;                            //开始计时

    P40 = 0;                                     //给 LED 供电
    EA = 1;                                      //打开总中断
```

```
    while (1)
    {
        if(B_1ms)
        {
            B_1ms = 0;
            msSecond++;
            if(msSecond >= 10)
            {
                msSecond = 0;
                TxPulse();                          //10ms 启动一次 PWM 输出
            }
        }
    }
}

/************* 发送脉冲函数 **************/
void TxPulse(void)
{
    PWMA_CCER1 = 0x00;                              //写 CCMRx 前必须先清零 CCxE  关闭通道
    PWMA_CCMR1 = 0x60;                              //设置 PWM1 模式 1 输出
    PWMA_CCER1 = 0x01;                              //使能 CC1E 通道, 高电平有效
    PWMA_SR1 = 0;                                   //清标志位
    PWMA_CNTR = 0;                                  //清计数器
    PWMA_IER = 0x02;                                //使能捕获/比较 1 中断
}

/*********************       Timer0 1ms 中断函数 **********************/
void timer0(void) interrupt 1
{
    B_1ms = 1;
    if(PWM1_Flag)
    {
        Period++;                                   //周期递增
        if(Period >= 0x1000) PWM1_Flag = 0;
    }
    else
    {
        Period--;                                   //周期递减
        if(Period <= 0x0100) PWM1_Flag = 1;
    }
    UpdatePwm();                                    //设置周期、占空比
}

/******************** PWM 中断函数 ********************/
void PWMA_ISR() interrupt 26
{
    if(PWMA_SR1 & 0X02)
    {
        PWMA_SR1 &=~0X02;                           //清标志位

        Counter++;
        if(Counter >= 10)                           //计数 10 个脉冲后关闭 PWM 计数器
        {
            Counter = 0;
            PWMA_CCER1 = 0x00;                      //写 CCMRx 前必须先清零 CCxE  关闭通道
            PWMA_CCMR1 = 0x40;                      //设置 PWM1 强制为无效电平
            PWMA_CCER1 = 0x01;                      //使能 CC1E 通道, 高电平有效
            PWMA_IER = 0x00;                        //关闭中断
```

```
            }
        }
}

//========================================================================
// 函数: UpdatePwm(void)
// 描述: 更新 PWM 周期占空比.
// 参数: none.
// 返回: none.
// 版本: V1.0
//========================================================================
void UpdatePwm(void)
{
    PWMA_ARR = Period;
    PWMA_CCR1 = (Period >> 1);                  //设置占空比时间: Period/2
}
```

25.10.6 高级 PWM 输出-频率可调-脉冲计数（硬件方式）

C 语言代码

```
//测试工作频率为 24MHz

/************ 功能说明 **************
本例程基于 STC8H8K64U 为主控芯片的实验箱 9 进行编写测试，STC8H 系列芯片可通用参考.
高级 PWM 定时器实现高速 PWM 脉冲输出.
周期/占空比可调, 通过比较/捕获中断进行脉冲个数计数.
通过 P6 口演示输出,每隔 10ms 输出一次 PWM,计数 10 个脉冲后停止输出.
使用单脉冲模式配合重复计数寄存器，纯硬件控制脉冲个数.
定时器每 1ms 调整 PWM 周期.
下载时,  选择时钟 24MHZ (用户可自行修改频率).
*****************************************/

#include   "stc8h.h"
#include   "intrins.h"

#define    MAIN_Fosc    24000000L              //定义主时钟

typedef    unsigned char   u8;
typedef    unsigned int    u16;
typedef    unsigned long   u32;

/*****************************  用户定义宏 ***********************************/

#define    Timer0_Reload  (65536UL -(MAIN_Fosc / 1000))    //Timer 0 中断频率, 1000 次/秒

/****************************************************************************/

#define    PWM1_1       0x00                   //P:P1.0   N:P1.1
#define    PWM1_2       0x01                   //P:P2.0   N:P2.1
#define    PWM1_3       0x02                   //P:P6.0   N:P6.1

#define    PWM2_1       0x00                   //P:P1.2/P5.4   N:P1.3
#define    PWM2_2       0x04                   //P:P2.2   N:P2.3
#define    PWM2_3       0x08                   //P:P6.2   N:P6.3

#define    PWM3_1       0x00                   //P:P1.4   N:P1.5
#define    PWM3_2       0x10                   //P:P2.4   N:P2.5
```

```
#define   PWM3_3    0x20                    //P:P6.4   N:P6.5

#define   PWM4_1    0x00                    //P:P1.6   N:P1.7
#define   PWM4_2    0x40                    //P:P2.6   N:P2.7
#define   PWM4_3    0x80                    //P:P6.6   N:P6.7
#define   PWM4_4    0xC0                    //P:P3.4   N:P3.3

#define   ENO1P     0x01
#define   ENO1N     0x02
#define   ENO2P     0x04
#define   ENO2N     0x08
#define   ENO3P     0x10
#define   ENO3N     0x20
#define   ENO4P     0x40
#define   ENO4N     0x80

/************* 本地变量声明   **************/
bit B_1ms;                                //1ms 标志
bit PWM1_Flag;

u16 Period;
u8 Counter;
u8 msSecond;

void UpdatePwm(void);
void TxPulse(u8 rep);

/********************* 主函数 **************************/
void main(void)
{
    P_SW2 |= 0x80;                        //扩展寄存器(XFR)访问使能

    P0M1 = 0x00;   P0M0 = 0x00;           //设置为准双向口
    P1M1 = 0x00;   P1M0 = 0x00;           //设置为准双向口
    P2M1 = 0x00;   P2M0 = 0x00;           //设置为准双向口
    P3M1 = 0x00;   P3M0 = 0x00;           //设置为准双向口
    P4M1 = 0x00;   P4M0 = 0x00;           //设置为准双向口
    P5M1 = 0x00;   P5M0 = 0x00;           //设置为准双向口
    P6M1 = 0x00;   P6M0 = 0x00;           //设置为准双向口
    P7M1 = 0x00;   P7M0 = 0x00;           //设置为准双向口

    PWM1_Flag = 0;
    Counter = 0;
    Period = 0x1000;

                                          // Timer0 初始化
    AUXR = 0x80;                          //Timer0 set as 1T, 16 bits timer auto-reload,
    TH0 = (u8)(Timer0_Reload / 256);
    TL0 = (u8)(Timer0_Reload % 256);
    ET0 = 1;                              //Timer0 interrupt enable
    TR0 = 1;                              //Tiner0 run

    PWMA_ENO = 0x00;
    PWMA_ENO |= ENO1P;                    //使能输出

    PWMA_CCER1 = 0x00;                    //写 CCMRx 前必须先清零 CCxE   关闭通道
    PWMA_CCMR1 = 0x68;                    //设置 PWM1 模式 1 输出
    //PWMA_CCER1 = 0x01;                  //使能     CC1E 通道, 高电平有效
    PWMA_CCER1 = 0x03;                    //使能 CC1E 通道, 低电平有效
```

```
    PWMA_PS = 0x00;                                 //高级 PWM 通道输出脚选择位
    PWMA_PS |= PWM1_3;                              //选择 PWM1_3 通道

    UpdatePwm();
    PWMA_BKR = 0x80;                                //使能主输出
    //PWMA_CR1 |= 0x89;                             //使能 ARR 预装载，单脉冲模式，开始计时

    P40 = 0;                                        //给 LED 供电
    EA = 1;                                         //打开总中断

    while (1)
    {
        if(B_1ms)
        {
            B_1ms = 0;
            msSecond++;
            if(msSecond >= 10)                      //10ms 启动一次 PWM 输出
            {
                msSecond = 0;
                TxPulse(10);                        //输出 10 个脉冲
            }
        }
    }
}

/************* 发送脉冲函数 **************/
void TxPulse(u8 rep)
{
    if(rep == 0) return;
    rep  -= 1;

    PWMA_RCR = rep;                                 //重复计数寄存器，计数 rep 个脉冲后产生更新事件
    PWMA_CR1 |= 0x89;                               //使能 ARR 预装载，单脉冲模式，开始计时
}

/********************** Timer0 1ms 中断函数 ************************/
void timer0(void) interrupt 1
{
    B_1ms = 1;
    if(PWM1_Flag)
    {
        Period++;                                   //周期递增
        if(Period >= 0x1000) PWM1_Flag = 0;
    }
    else
    {
        Period--;                                   //周期递减
        if(Period <= 0x0100) PWM1_Flag = 1;
    }
    UpdatePwm();                                    //设置周期、占空比
}

//========================================================================
// 函数: UpdatePwm(void)
// 描述: 更新 PWM 周期占空比.
// 参数: none.
// 返回: none.
// 版本: V1.0
```

```
//========================================================================
void UpdatePwm(void)
{
    PWMA_ARR = Period;
    PWMA_CCR1 = (Period >> 1);                  //设置占空比时间: Period/2
}
```

25.10.7　高级 PWM 端口做外部中断（仅下降沿中断或者仅上升沿中断）

特别注意：单个 PWM 口不能实现同时检测上升沿和下降沿信号，如果需要同时检测上升沿和下降沿，可将外部信号同时连接到两路 PWM，使用其中一路检测信号的上升沿，另外一路 PWM 检测信号的下降沿。

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "stc8h.h"
#include "intrins.h"

void main(void)
{
    P_SW2 |= 0x80;                              //使能 XFR 访问

    P1M1 = 0x00;
    P1M0 = 0x00;
    P3M1 = 0x00;
    P3M0 = 0x00;
                                                //(捕获 PWM1P 上升沿/下降沿)
    PWMA_CCER1 = 0x00;
    PWMA_CCMR1 = 0x01;                          //CC1 为输入模式,且映射到 TI1FP1 上
    PWMA_CCER1 = 0x01;                          //使能 CC1 上的捕获功能
    PWMA_CCER1 |= 0x00;                         //设置捕获极性为 CC1 的上升沿
//  PWMA_CCER1 |= 0x02;                         //设置捕获极性为 CC1 的下降沿
    PWMA_CR1 = 0x01;
    PWMA_IER = 0x02;
    EA = 1;

    while (1);
}

void PWMA_ISR() interrupt 26
{
    if(PWMA_SR1 & 0X02)
    {
        P37 = ~P37;
        PWMA_SR1 &=~0X02;
    }
}
```

25.10.8 高级 PWM 一个端口做外部中断（同时捕获下降沿和上升沿）

原理：使用高级 PWM 内部的两通道的捕获模块 CCx 和 CCx+1 同时捕获外部的同一个管脚，CCx 捕获此管脚的上升沿，CCx+1 捕获此管脚的下降沿，即可同时捕获此管脚的上升沿中断和下降沿中断

范例：使用 PWMA 的第一组捕获模块 CC1 和第二组捕获模块 CC2，同时捕获 PWM1P 管脚（P1.0），其中 CC1 捕获 PWM1P 的上升沿，CC2 捕获 PWM1P 的下降沿。

注意：1、使用的是芯片内部的两路捕获模块同时捕获外部的同一个管脚，所以不需要将外部信号连接到多个 PWM 捕获管脚。

2、只有 CC1+CC2、CC3+CC4、CC5+CC6、CC7+CC8 这 4 种组合才能完成上面的功能。CC1+CC2 组合可以同时仅捕获 PWM1P 管脚，也可以同时仅捕获 PWM2P 管脚；CC3+CC4 组合可以同时仅捕获 PWM3P 管脚，也可以同时仅捕获 PWM4P 管脚；CC5+CC6 组合可以同时仅捕获 PWM5 管脚，也可以同时仅捕获 PWM6 管脚；CC7+CC8 组合可以同时仅捕获 PWM7 管脚，也可以同时仅捕获 PWM8 管脚

C 语言代码

```
//测试工作频率为 11.0592MHz
#include "stc8h.h"
#include "intrins.h"
void main()
{
    P_SW2 |= 0x80;                          //使能访问 XFR，没有冲突不用关闭

    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;
                                            //(CC1 捕获 TI1 上升沿,CC2 捕获 TI1 下降沿)
    PWMA_CCER1 = 0x00;
    PWMA_CCMR1 = 0x01;                      //CC1 为輸入模式,且映射到 TI1FP1(PWM1P)上
    PWMA_CCMR2 = 0x02;                      //CC2 为輸入模式,且映射到 TI1FP2(PWM1P)上
    PWMA_CCER1 = 0x11;                      //使能 CC1/CC2 上的捕获功能
    PWMA_CCER1 |= 0x00;                     //设置捕获极性为 CC1 的上升沿
    PWMA_CCER1 |= 0x20;                     //设置捕获极性为 CC2 的下降沿
    PWMA_CR1 = 0x01;

    PWMA_IER = 0x06;                        //使能 CC1 和 CC2 捕获中断
    EA = 1;

    while (1);
}

void PWMA_ISR() interrupt 26
{
    unsigned int cnt;
    if (PWMA_SR1 & 0x02)
    {
        PWMA_SR1 &= ~0x02;                  //此时为 PWM1P（P1.0）口的上升沿中断
    }
    if (PWMA_SR1 & 0x04)
    {
        PWMA_SR1 &= ~0x04;                  //此时为 PWM1P（P1.0）口的下降沿中断
    }
```

```
}
```

25.10.9　输入捕获模式测量脉冲周期（捕获上升沿到上升沿或者下降沿到下降沿）

原理：使用高级 PWM 内部的某一通道的捕获模块 CCx，捕获外部的端口的上升沿或者下降沿，两个上升沿之间或者两个下降沿之间的时间即为脉冲的周期，也就是说，两次捕获计数值的差值即为周期值。

范例：使用 PWMA 的第一组捕获模块 CC1 捕获功能，捕获 PWM1P（P1.0）管脚上的上升沿，在中断中对前后两次的捕获值相减得到周期

注意：只有 PWM1P、PWM2P、PWM3P、PWM4P、PWM5、PWM6、PWM7、PWM8 这些管脚以及相应切换管脚才有捕获功能

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "stc8h.h"
#include "intrins.h"

int         cap;
int         cap_new;
int         cap_old;

void main(void)
{
    P_SW2 |= 0x80;                          //使能 XFR 访问

    P0M1 = 0x00;
    P0M0 = 0x00;
    P1M1 = 0x00;
    P1M0 = 0x00;

    PWMA_PS = 0x00;                         //00:PWM at P1

    PWMA_PSCRH = 0x00;                      //预分频寄存器
    PWMA_PSCRL = 0x00;

    PWMA_CCMR1 = 0x01;                      //通道模式配置
    PWMA_CCER1 = 0x01;                      //使能通道并配置通道极性
    PWMA_IER = 0x02;                        //使能中断

    PWMA_CR1 |= 0x01;                       //使能计数器

    EA = 1;
    while (1);
}

void PWMA_ISR() interrupt 26
{
    if(PWMA_SR1 & 0X02)
    {
        cap_old = cap_new;
        cap_new = PWMA_CCR1;                //读取 CCR1
        cap = cap_new - cap_old;
        PWMA_SR1 &= ~0x02;
    }
}
```

25.10.10 输入捕获模式测量脉冲高电平宽度(捕获上升沿到下降沿)

原理：使用高级 PWM 内部的两通道的捕获模块 CCx 和 CCx+1 同时捕获外部的同一个管脚，CCx 捕获此管脚的上升沿，CCx+1 捕获此管脚的下降沿，然利用 CCx+1 的捕获值减去 CCx 的捕获值，其差值即为脉冲高电平的宽度。

范例：使用 PWMA 的第一组捕获模块 CC1 和第二组捕获模块 CC2，同时捕获 PWM1P 管脚（P1.0），其中 CC1 捕获 PWM1P 的上升沿，CC2 捕获 PWM1P 的下降沿，在中断中使用 CC2 的捕获值减去 CC1 的捕获值，其差值即为脉冲高电平的宽度。

注意：1、使用的是芯片内部的两路捕获模块同时捕获外部的同一个管脚，所以不需要将外部的多个管脚相连接。

2、只有 CC1+CC2、CC3+CC4、CC5+CC6、CC7+CC8 这 4 种组合才能完成上面的功能。CC1+CC2 组合可以同时捕获 PWM1P 管脚，也可以同时捕获 PWM2P 管脚；CC3+CC4 组合可以同时捕获 PWM3P 管脚，也可以同时捕获 PWM4P 管脚；CC5+CC6 组合可以同时捕获 PWM5 管脚，也可以同时捕获 PWM6 管脚；CC7+CC8 组合可以同时捕获 PWM7 管脚，也可以同时捕获 PWM8 管脚

C 语言代码

```
//测试工作频率为 11.0592MHz
#include "stc8h.h"
#include "intrins.h"

void main()
{
    P_SW2 |= 0x80;                          //使能访问 XFR，没有冲突不用关闭

    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

                                            //(CC1 捕获 TI1 上升沿,CC2 捕获 TI1 下降沿)
    PWMA_CCER1 = 0x00;
    PWMA_CCMR1 = 0x01;                      //CC1 为输入模式,且映射到 TI1FP1 上
    PWMA_CCMR2 = 0x02;                      //CC2 为输入模式,且映射到 TI1FP2 上
    PWMA_CCER1 = 0x11;                      //使能 CC1/CC2 上的捕获功能
    PWMA_CCER1 |= 0x00;                     //设置捕获极性为 CC1 的上升沿
    PWMA_CCER1 |= 0x20;                     //设置捕获极性为 CC2 的下降沿
    PWMA_CR1 = 0x01;

    PWMA_IER = 0x04;                        //使能 CC2 捕获中断
    EA = 1;

    while (1);
}

void PWMA_ISR() interrupt 26
{
    unsigned int cnt;

    if (PWMA_SR1 & 0x04)
    {
        PWMA_SR1 &= ~0x04;
```

```
        cnt = PWMA_CCR2 - PWMA_CCR1;                //差值即为高电平宽度
    }
}
```

25.10.11 输入捕获模式测量脉冲低电平宽度(捕获下降沿到上升沿)

原理：使用高级 PWM 内部的两通道的捕获模块 CCx 和 CCx+1 同时捕获外部的同一个管脚，CCx 捕获此管脚的下降沿，CCx+1 捕获此管脚的上升沿，然利用 CCx+1 的捕获值减去 CCx 的捕获值，其差值即为脉冲低电平的宽度。

范例：使用 PWMA 的第一组捕获模块 CC1 和第二组捕获模块 CC2，同时捕获 PWM1P 管脚（P1.0），其中 CC1 捕获 PWM1P 的下降沿，CC2 捕获 PWM1P 的上升沿，在中断中使用 CC2 的捕获值减去 CC1 的捕获值，其差值即为脉冲低电平的宽度。

注意：1、使用的是芯片内部的两路捕获模块同时捕获外部的同一个管脚，所以不需要将外部的多个管脚相连接。

2、只有 CC1+CC2、CC3+CC4、CC5+CC6、CC7+CC8 这 4 种组合才能完成上面的功能。CC1+CC2 组合可以同时捕获 PWM1P 管脚，也可以同时捕获 PWM2P 管脚；CC3+CC4 组合可以同时捕获 PWM3P 管脚，也可以同时捕获 PWM4P 管脚；CC5+CC6 组合可以同时捕获 PWM5 管脚，也可以同时捕获 PWM6 管脚；CC7+CC8 组合可以同时捕获 PWM7 管脚，也可以同时捕获 PWM8 管脚

C 语言代码

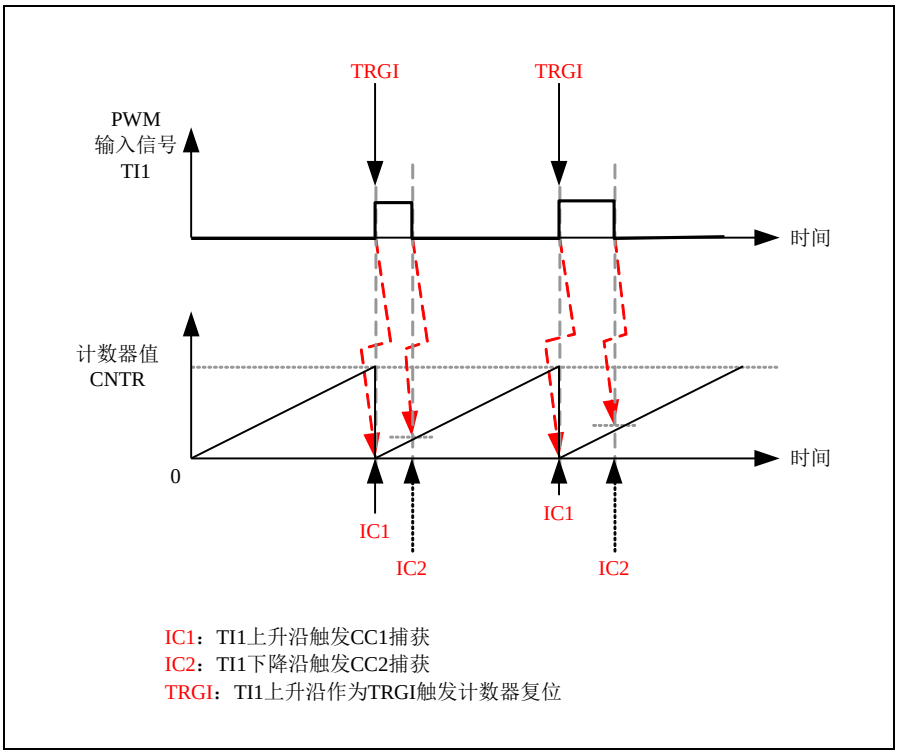
```
//测试工作频率为 11.0592MHz
#include "stc8h.h"
#include "intrins.h"

void main()
{
    P_SW2 |= 0x80;                                  //使能访问 XFR，没有冲突不用关闭

    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

                                                    //(CC1 捕获 TI1 上升沿,CC2 捕获 TI1 下降沿)
    PWMA_CCER1 = 0x00;
    PWMA_CCMR1 = 0x01;                              //CC1 为输入模式,且映射到 TI1FP1 上
    PWMA_CCMR2 = 0x02;                              //CC2 为输入模式,且映射到 TI1FP2 上
    PWMA_CCER1 = 0x11;                              //使能 CC1/CC2 上的捕获功能
    PWMA_CCER1 |= 0x00;                             //设置捕获极性为 CC1 的上升沿
    PWMA_CCER1 |= 0x20;                             //设置捕获极性为 CC2 的下降沿
    PWMA_CR1 = 0x01;

    PWMA_IER = 0x02;                                //使能 CC1 捕获中断
    EA = 1;

    while (1);
}

void PWMA_ISR() interrupt 26
{
```

```
    unsigned int cnt;

    if (PWMA_SR1 & 0x02)
    {
        PWMA_SR1 &= ~0x02;

        cnt = PWMA_CCR1 - PWMA_CCR2;              //差值即为低电平宽度
    }
}
```

25.10.12　输入捕获模式同时测量脉冲周期和高电平宽度（占空比）

原理：使用高级 PWM 内部的两通道的捕获模块 CCx 和 CCx+1 同时捕获外部的同一个管脚，CCx 捕获此管脚的上升沿，CCx+1 捕获此管脚的下降沿，同时使能此管脚的上升沿信号为复位触发信号，CCx 的捕获值即为周期，CCx+1 的捕获值即为占空比。

范例：使用 PWMA 的第一组捕获模块 CC1 和第二组捕获模块 CC2，同时捕获 PWM1P 管脚（P1.0），其中 CC1 捕获 PWM1P 的上升沿，CC2 捕获 PWM1P 的下降沿，并设置 PWM1P 的上升沿信号为复位触发信号，CC1 的捕获值即为周期，CC2 的捕获值即为占空比。



注意：1、使用的是芯片内部的两路捕获模块同时捕获外部的同一个管脚，所以不需要将外部的多个管脚相连接。

2、只有 CC1+CC2、CC5+CC6 这两种组合才能完成上面的功能。CC1+CC2 组合可以同时捕获 PWM1P 管脚，也可以同时捕获 PWM2P 管脚； CC5+CC6 组合可以同时捕获 PWM5 管脚，也可以同时捕获 PWM6 管脚

3、由于设置了复位触发信号，所以捕获值即为周期值和占空比值，无需再减前一次的捕获值。

C 语言代码

```
//测试工作频率为 11.0592MHz
#include "stc8h.h"
#include "intrins.h"

void main()
```

```
{
    P_SW2 |= 0x80;                          //使能访问 XFR，没有冲突不用关闭

    P1M0 = 0x00;
    P1M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

                                            //(CC1 捕获 TI1 上升沿,CC2 捕获 TI1 下降沿)
                                            //CC1 捕获周期宽度,CC2 捕获高电平宽度
    PWMA_CCER1 = 0x00;
    PWMA_CCMR1 = 0x01;                      //CC1 为输入模式,且映射到 TI1FP1 上
    PWMA_CCMR2 = 0x02;                      //CC2 为输入模式,且映射到 TI1FP2 上
    PWMA_CCER1 = 0x11;                      //使能 CC1/CC2 上的捕获功能
    PWMA_CCER1 |= 0x00;                     //设置捕获极性为 CC1 的上升沿
    PWMA_CCER1 |= 0x20;                     //设置捕获极性为 CC2 的下降沿
    PWMA_SMCR = 0x54;                       //TS=TI1FP1,SMS=TI1 上升沿复位模式
    PWMA_CR1 = 0x01;

    PWMA_IER = 0x06;                        //使能 CC1/CC2 捕获中断
    EA = 1;

    while (1);
}

void PWMA_ISR() interrupt 26
{
    unsigned int cnt;

    if (PWMA_SR1 & 0x02)
    {
        PWMA_SR1 &= ~0x02;

        cnt = PWMA_CCR1;                    //CC1 捕获周期宽度
    }
    if (PWMA_SR1 & 0x04)
    {
        PWMA_SR1 &= ~0x04;

        cnt = PWMA_CCR2;                    //CC2 捕获占空比（高电平宽度）
    }
}
```

25.10.13 同时捕获 4 路输入信号的周期和高电平宽度（占空比）

原理：使用高级 PWM 内部的两通道的捕获模块 CCx 和 CCx+1 同时捕获外部的同一个管脚，CCx 捕获此管脚的上升沿，CCx+1 捕获此管脚的下降沿，CCx 的两次捕获值的差值即为周期，CCx+1 的捕获值与 CCx 的前一次捕获值的差值即为占空比。

范例：使用 PWMA 的第一组捕获模块 CC1 和第二组捕获模块 CC2，同时捕获 PWM1P 管脚（P1.0），其中 CC1 捕获 PWM1P 的上升沿（CC1/CC2 均既可以捕获 PWM1P 的上升沿，也可捕获 PWM1P 的下降沿，本例仅演示 CC1 捕获 PWM1P 的上升沿，CC2 捕获 PWM1P 的下降沿。CC3/CC4，CC5/CC6，CC7/CC8 同理），CC2 捕获 PWM1P 的下降沿，CC1 的捕获值减去前一次捕获值即为周期，CC2 的捕获值减去 CC1 的前一次捕获值即为占空比。PWMA 的 CC3 和 CC4 同时捕获 PWM3P（P1.4）、

PWMB 的 CC5 和 CC6 同时捕获 PWM5（P2.0）、PWMB 的 CC7 和 CC8 同时捕获 PWM7（P2.2）。另外本例演示使用定时器 0 在 P1.0 上产生波形、定时器 1 在 P1.4 上产生波形、定时器 3 在 P2.0 上产生波形、定时器 4 在 P2.2 上产生波形。捕获值通过串口送到 PC。

注意：1、使用的是芯片内部的两路捕获模块同时捕获外部的同一个管脚，所以不需要将外部的多个管脚相连接。

2、由于没有设置复位触发信号，所以周期值和占空比值均需要作相应的减法运算才能得到。

```
//测试工作频率为 11.0592MHz

#include "stc8h.h"
#include "intrins.h"
#include "stdio.h"

#define   FOSC        12000000UL
#define   BRT         (65536 - FOSC / 115200 / 4)
#define   T10K        (65536 - FOSC / 10000)
#define   T11K        (65536 - FOSC / 11000)
#define   T12K        (65536 - FOSC / 12000)
#define   T13K        (65536 - FOSC / 13000)

unsigned int ccr1;
unsigned int ccr3;
unsigned int ccr5;
unsigned int ccr7;

unsigned int cycle1;
unsigned int duty1;
unsigned int cycle2;
unsigned int duty2;
unsigned int cycle3;
unsigned int duty3;
unsigned int cycle4;
unsigned int duty4;

bit f1, f2, f3, f4;

void main()
{

    P_SW2 |= 0x80;                                  //使能访问 XFR，没有冲突不用关闭

    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    AUXR |= 0x80;                                   //定时器 0 使用 1T 模式
    AUXR |= 0x40;                                   //定时器 1 使用 1T 模式
    TMOD = 0x00;                                    //定时器 0/1 使用 16 位自动重载模式
    TL0 = T10K;                                     //定时器 0 周期 10K
    TH0 = T10K >> 8;
```

```
    TL1 = T11K;                                 //定时器 1 周期 11K
    TH1 = T11K >> 8;
    TR0 = 1;                                    //定时器 0 开始计数
    TR1 = 1;                                    //定时器 1 开始计数
    ET0 = 1;                                    //使能定时器 0 中断
    ET1 = 1;                                    //使能定时器 1 中断

    T3L = T12K;                                 //定时器 3 周期 12K
    T3H = T12K >> 8;
    T4L = T13K;                                 //定时器 4 周期 13K
    T4H = T13K >> 8;
    T4T3M = 0xaa;                               //定时器 3/4 使用 1T 模式
    IE2 |= 0x20;                                //使能定时器 3 中断
    IE2 |= 0x40;                                //使能定时器 4 中断

    SCON = 0x52;
    T2L = BRT;
    T2H = BRT >> 8;
    AUXR |= 0x15;

    printf("PWM Test .\n");

    PWMA_CCER1 = 0x00;
    PWMA_CCMR1 = 0x01;                          //CC1 为输入模式,且映射到 TI1FP1 上
    PWMA_CCMR2 = 0x02;                          //CC2 为输入模式,且映射到 TI1FP2 上
    PWMA_CCER1 = 0x11;                          //使能 CC1 上的捕获功能,使能 CC2 上的捕获功能
    PWMA_CCER1 |= 0x00;                         //设置捕获极性为 CC1 的上升沿
    PWMA_CCER1 |= 0x20;                         //设置捕获极性为 CC2 的下降沿

    PWMA_CCER2 = 0x00;
    PWMA_CCMR3 = 0x01;                          //CC3 为输入模式,且映射到 TI3FP3 上
    PWMA_CCMR4 = 0x02;                          //CC4 为输入模式,且映射到 TI3FP4 上
    PWMA_CCER2 = 0x11;                          //使能 CC3 上的捕获功能,使能 CC4 上的捕获功能
    PWMA_CCER2 |= 0x00;                         //设置捕获极性为 CC3 的上升沿
    PWMA_CCER2 |= 0x20;                         //设置捕获极性为 CC4 的下降沿
    PWMA_CR1 = 0x01;

    PWMA_IER = 0x1e;                            //使能 CC1/CC2/CC3/CC4 捕获中断

    PWMB_CCER1 = 0x00;
    PWMB_CCMR1 = 0x01;                          //CC5 为输入模式,且映射到 TI5FP5 上
    PWMB_CCMR2 = 0x02;                          //CC6 为输入模式,且映射到 TI5FP6 上
    PWMB_CCER1 = 0x11;                          //使能 CC5 上的捕获功能,使能 CC6 上的捕获功能
    PWMB_CCER1 |= 0x00;                         //设置捕获极性为 CC5 的上升沿
    PWMB_CCER1 |= 0x20;                         //设置捕获极性为 CC6 的下降沿

    PWMB_CCER2 = 0x00;
    PWMB_CCMR3 = 0x01;                          //CC7 为输入模式,且映射到 TI7FP8 上
    PWMB_CCMR4 = 0x02;                          //CC8 为输入模式,且映射到 TI7FP8 上
    PWMB_CCER2 = 0x11;                          //使能 CC7 上的捕获功能,使能 CC8 上的捕获功能
    PWMB_CCER2 |= 0x00;                         //设置捕获极性为 CC7 的上升沿
    PWMB_CCER2 |= 0x20;                         //设置捕获极性为 CC8 的下降沿
    PWMB_CR1 = 0x01;

    PWMB_IER = 0x1e;                            //使能 CC5/CC6/CC7/CC8 捕获中断

    EA = 1;

    while (1)
```

```
    {
        if (f1)
        {
            f1 = 0;
            printf("cycle1 = %04x duty1 = %04x\n", cycle1, duty1);
        }
        if (f2)
        {
            f2 = 0;
            printf("cycle2 = %04x duty2 = %04x\n", cycle2, duty2);
        }
        if (f3)
        {
            f3 = 0;
            printf("cycle3 = %04x duty3 = %04x\n", cycle3, duty3);
        }
        if (f4)
        {
            f4 = 0;
            printf("cycle4 = %04x duty4 = %04x\n", cycle4, duty4);
        }
    }
}

void TMR0_ISR() interrupt TMR0_VECTOR                   //产生 CC1 波形到 P1.0 口
{
    static unsigned int cnt = 0;

    cnt++;
    if (cnt == 10)
    {
        P10 = 0;
    }
    else if (cnt == 30)
    {
        P10 = 1;
        cnt = 0;
    }
}

void TMR1_ISR() interrupt TMR1_VECTOR                   //产生 CC3 波形到 P1.4 口
{
    static unsigned int cnt = 0;

    cnt++;
    if (cnt == 10)
    {
        P14 = 0;
    }
    else if (cnt == 30)
    {
        P14 = 1;
        cnt = 0;
    }
}

void TMR3_ISR() interrupt TMR3_VECTOR                   //产生 CC5 波形到 P2.0 口
{
    static unsigned int cnt = 0;
```

```
        cnt++;
        if (cnt == 10)
        {
            P20 = 0;
        }
        else if (cnt == 30)
        {
            P20 = 1;
            cnt = 0;
        }
}

void TMR4_ISR() interrupt TMR4_VECTOR                   //产生 CC7 波形到 P2.2 口
{
        static unsigned int cnt = 0;

        cnt++;
        if (cnt == 10)
        {
            P22 = 0;
        }
        else if (cnt == 30)
        {
            P22 = 1;
            cnt = 0;
        }
}

void PWMA_ISR() interrupt PWMA_VECTOR
{
        unsigned int ccr;

        if (PWMA_SR1 & 0x02)                            //CC1 捕获中断
        {
            PWMA_SR1 &= ~0x02;

            ccr = (PWMA_CCR1H << 8) + PWMA_CCR1L;       //读取捕获值
            cycle1 = ccr - ccr1;                        //计算周期
            ccr1 = ccr;                                 //保存当前捕获值
            f1 = 1;                                     //波形 1 的周期和占空比捕获完成，触发串口发送
        }
        if (PWMA_SR1 & 0x04)                            //CC2 捕获中断
        {
            PWMA_SR1 &= ~0x04;

            ccr = (PWMA_CCR2H << 8) + PWMA_CCR2L;       //读取捕获值
            duty1 = ccr - ccr1;                         //计算占空比
        }

        if (PWMA_SR1 & 0x08)                            //CC3 捕获中断
        {
            PWMA_SR1 &= ~0x08;

            ccr = (PWMA_CCR3H << 8) + PWMA_CCR3L;       //读取捕获值
            cycle2 = ccr - ccr3;                        //计算周期
            ccr3 = ccr;                                 //保存当前捕获值
            f2 = 1;                                     //波形 2 的周期和占空比捕获完成，触发串口发送
        }
```

```
    if (PWMA_SR1 & 0x10)                    //CC4 捕获中断
    {
        PWMA_SR1 &= ~0x10;

        ccr = (PWMA_CCR4H << 8) + PWMA_CCR4L;    //读取捕获值
        duty2 = ccr - ccr3;                      //计算占空比
    }
}

void PWMB_ISR() interrupt PWMB_VECTOR
{
    unsigned int ccr;

    if (PWMB_SR1 & 0x02)                    //CC5 捕获中断
    {
        PWMB_SR1 &= ~0x02;

        ccr = (PWMB_CCR5H << 8) + PWMB_CCR5L;    //读取捕获值
        cycle3 = ccr - ccr5;                     //计算周期
        ccr5 = ccr;                              //保存当前捕获值
        f3 = 1;                                  //波形 3 的周期和占空比捕获完成，触发串口发送
    }
    if (PWMB_SR1 & 0x04)                    //CC6 捕获中断
    {
        PWMB_SR1 &= ~0x04;

        ccr = (PWMB_CCR6H << 8) + PWMB_CCR6L;    //读取捕获值
        duty3 = ccr - ccr5;                      //计算占空比
    }

    if (PWMB_SR1 & 0x08)                    //CC7 捕获中断
    {
        PWMB_SR1 &= ~0x08;

        ccr = (PWMB_CCR7H << 8) + PWMB_CCR7L;    //读取捕获值
        cycle4 = ccr - ccr7;                     //计算周期
        ccr7 = ccr;                              //保存当前捕获值
        f4 = 1;                                  //波形 4 的周期和占空比捕获完成，触发串口发送
    }
    if (PWMB_SR1 & 0x10)                    //CC8 捕获中断
    {
        PWMB_SR1 &= ~0x10;

        ccr = (PWMB_CCR8H << 8) + PWMB_CCR8L;    //读取捕获值
        duty4 = ccr - ccr7;                      //计算占空比
    }
}
```

25.10.14 输出两路有相位差的 PWM（PWMA）

```
//测试工作频率为 12MHz

/************ 功能说明 **************
MCU 主频 12MHz, PWM 时钟 12 分频, 时间的分辨率为 1us.
PWM1 通道工作于 PWM 模式 1, PWM2 切换输出模式. 输出 PWM 周期 1ms.
通道 1 输出: P2.0, PWMA1P 输出, 本例子输出 200us 高电平.
```

```
通道 2 输出: P2.2, PWMA2P 输出, PWMA2 相对 PWMA1 滞后一个角度(相位), 本例子要求在 PWM1P 输出下降沿时,PWM2P 输出上升沿, 高电平 200us.
*****************************************/

#include "STC8H.h"

bit B_OutState=0;

unsigned int pwm1H,pwm2H;

void PWMA_config(void);

/******************** 主函数 **************************/
void main(void)
{
    P_SW2 |= 0x80;                                  //扩展寄存器访问使能
    P0M1 = 0x00; P0M0 = 0x00;                       //设置为准双向口
    P1M1 = 0x00; P1M0 = 0x00;                       //设置为准双向口
    P2M1 = 0x10; P2M0 = 0x2c;                       //设置为准双向口
    P3M1 = 0x00; P3M0 = 0x00;                       //设置为准双向口
    P4M1 = 0x00; P4M0 = 0x00;                       //设置为准双向口
    P5M1 = 0x00; P5M0 = 0x00;                       //设置为准双向口
    P6M1 = 0x00; P6M0 = 0x00;                       //设置为准双向口
    P7M1 = 0x00; P7M0 = 0x00;                       //设置为准双向口

    pwm1H = 200;
    pwm2H = 400;

    PWMA_config();

    EA = 1;

    while (1)；
}

//========================================================================
// 函数: void PWMA_config(void)
// 描述: PWM 配置函数。
//========================================================================
void PWMA_config(void)
{
    PWMA_CCER1 = 0;
    PWMA_CCER2 = 0;
    PWMA_ENO = 0;                                   // IO 输出禁止
    PWMA_IER = 0;                                   // 禁止中断
    PWMA_SR1 = 0;                                   // 清除状态
    PWMA_SR2 = 0;                                   // 清除状态
    PWMA_CR1 = 0;                                   // 清除控制寄存器
    PWMA_CR2 = 0;                                   // 清除控制寄存器

    PWMA_PSCRH = 0;                                 // 预分频寄存器, PWM 时钟 = 12MHz/(11+1)=1MHz,
                                                    //分频 Fck_cnt = Fck_psc/(PSCR[15:0}+1),
                                                    //边沿对齐 PWM 频率 = SYSclk/((PSCR+1)*(AAR+1))
                                                    //中央对齐 PWM 频率 = SYSclk/((PSCR+1)*AAR*2).
    PWMA_PSCRL = 11;
    PWMA_ARRH = (1000-1)/256;                       // 自动重装载寄存器, 控制 PWM 周期
    PWMA_ARRL = (1000-1)%256;
    PWMA_IER |= 0x01;                               // 使能中断
```

```
    PWMA_CCMR1 = 0x68;                      // 通道模式配置, PWM 模式 1, 预装载允许
    PWMA_CCR1 = pwm1H;                      // 比较值, 控制占空比(高电平时钟数)
    PWMA_CCER1 |= 0x05;                     // 开启比较输出, 高电平有效
    PWMA_PS |= 1;                           // 0:选择 P1.0 P1.1,
                                            // 1:选择 P2.0 P2.1,
                                            // 2:选择 P6.0 P6.1,
    PWMA_ENO |= 0x01;                       // IO 输出允许, bit7: ENO4N,   bit6: ENO4P,
                                            // bit5: ENO3N,   bit4: ENO3P,   bit3: ENO2N,
                                            // bit2: ENO2P,   bit1: ENO1N,   bit0: ENO1P

    PWMA_CCMR2  = 0x70;                     // 通道模式配置, PWM 模式 2
    PWMA_CCR2 = pwm1H;                      // 匹配值
    PWMA_CCER1 |= 0x50;                     // 开启比较输出, 高电平有效
    PWMA_PS |= (1<<2);                      // 选择 IO, 0:选择 P1.2 P1.3, 1:选择 P2.2 P2.3,
                                            //2:选择 P6.2 P6.3,
    PWMA_ENO    |= 0x04;                    // IO 输出允许, bit7: ENO4N,   bit6: ENO4P,
                                            // bit5: ENO3N,   bit4: ENO3P,   bit3: ENO2N,
                                            // bit2: ENO2P,   bit1: ENO1N,   bit0: ENO1P
    PWMA_IER |= 0x04;                       // 使能通道 2 匹配中断

    PWMA_BKR = 0x80;                        // 主输出使能 相当于总开关
    PWMA_CR1 = 0x81;                        // 使能计数器, 允许自动重装载寄存器缓冲,
                                            // 边沿对齐模式, 向上计数,   bit7=1:写自动重装载
                                            // 寄存器缓冲(本周期不会被打扰), =0:直接写自动
                                            // 重装载寄存器本(周期可能会乱掉)
}

//========================================================================
// 函数: void PWMA_ISR(void) interrupt 26
// 描述: PWMA 中断处理程序.
//========================================================================
void PWMA_ISR(void) interrupt 26
{
    unsigned char   sr1;
    sr1 = PWMA_SR1;                         //为了快速,中断标志用一个局部变量处理
    PWMA_SR1 = 0;                           //清除中断标志
    PWMA_SR2 = 0;                           //清除中断标志

    if(sr1 & 0x01)                          //更新中断,装载 PWM2 匹配输出高电平的时刻
    {
        PWMA_CCR2   = pwm1H;                // 匹配值, PWM2 在 200 时刻输出高电平
        PWMA_CCMR2 = 0x70;                  // 通道模式配置, PWM 模式 2
        B_OutState = 0;
    }
    if(sr1 & 0x04)                          //通道 2 匹配中断,装载 PWM2 匹配输出低电平的时刻
    {
        if(!B_OutState)
        {
            B_OutState = 1;
            PWMA_CCMR2 = 0x50;              // 通道模式配置, 强制为有效电平
            PWMA_CCR2 = pwm2H;              // 匹配值, PWM2 在 400 时刻输出低电平
            PWMA_CCMR2 = 0x60;              // 通道模式配置, PWM 模式 1
        }
        else PWMA_CCMR2 = 0x40;             // 通道模式配置, 强制为无效电平
    }
}
```

25.10.15 输出两路有相位差的 PWM（PWMB）

```
//测试工作频率为 12MHz

/************ 功能说明 **************
MCU 主频 12MHz, PWM 时钟 12 分频, 时间的分辨率为 1us.

PWM5 通道工作于 PWM 模式 1, PWM6 切换输出模式. 输出 PWM 周期 1ms.
通道 1 输出: P2.0, PWM5, 本例子输出 200us 高电平.
通道 2 输出: P2.1, PWM6, PWMB6 相对 PWMB5 滞后一个角度(相位), 本例子要求在 PWM5 输出下降沿时,PWM6 输出上升沿, 高电平 200us.
*****************************************/

#include "STC8H.h"

bit B_OutState=0;

unsigned int pwm1H,pwm2H;

void PWMB_config(void);

/******************** 主函数 **************************/
void main(void)
{
    P_SW2 |= 0x80;                                  //扩展寄存器访问使能
    P0M1 = 0x00; P0M0 = 0x00;                       //设置为准双向口
    P1M1 = 0x00; P1M0 = 0x00;                       //设置为准双向口
    P2M1 = 0x10; P2M0 = 0x2c;                       //设置为准双向口
    P3M1 = 0x00; P3M0 = 0x00;                       //设置为准双向口
    P4M1 = 0x00; P4M0 = 0x00;                       //设置为准双向口
    P5M1 = 0x00; P5M0 = 0x00;                       //设置为准双向口
    P6M1 = 0x00; P6M0 = 0x00;                       //设置为准双向口
    P7M1 = 0x00; P7M0 = 0x00;                       //设置为准双向口

    pwm1H = 200;
    pwm2H = 400;

    PWMB_config();

    EA = 1;

    while (1);
}

//========================================================================
// 函数: void PWMA_config(void)
// 描述: PWM 配置函数。
//========================================================================
void PWMB_config(void)
{
    PWMB_CCER1 = 0;
    PWMB_CCER2 = 0;
    PWMB_ENO = 0;                                   // IO 输出禁止
    PWMB_IER = 0;                                   // 禁止中断
    PWMB_SR1 = 0;                                   // 清除状态
    PWMB_SR2 = 0;                                   // 清除状态
    PWMB_CR1 = 0;                                   // 清除控制寄存器
```

```
    PWMB_CR2 = 0;                                   // 清除控制寄存器

    PWMB_PSCRH = 0;                                 // 预分频寄存器, PWM 时钟 = 12MHz/(11+1)=1MHz,
                                                    //分频 Fck_cnt = Fck_psc/(PSCR[15:0}+1),
                                                    //边沿对齐 PWM 频率 = SYSclk/((PSCR+1)*(AAR+1))
                                                    //中央对齐 PWM 频率 = SYSclk/((PSCR+1)*AAR*2).
    PWMB_PSCRL = 11;
    PWMB_ARRH = (1000-1)/256;                       // 自动重装载寄存器,  控制 PWM 周期
    PWMB_ARRL = (1000-1)%256;
    PWMB_IER |= 0x01;                               // 使能中断

    PWMB_CCMR1 = 0x68;                              // 通道模式配置, PWM 模式 1, 预装载允许
    PWMB_CCR5 = pwm1H;                              // 比较值, 控制占空比(高电平时钟数)
    PWMB_CCER1 |= 0x05;                             // 开启比较输出, 高电平有效
    PWMB_PS |= 1;                                   // 0:选择 P2.0, 1:选择 P1.7, 2:选择 P0.0, 3:选择 P7.4
    PWMB_ENO |= 0x01;                               // IO 输出允许, bit6: ENO8P, bit4: ENO7P,
                                                    // bit2: ENO6P,   bit0: ENO5P
    PWMB_CCMR2  = 0x70;                             // 通道模式配置, PWM 模式 2
    PWMB_CCR6 = pwm1H;                              // 匹配值
    PWMB_CCER1 |= 0x50;                             // 开启比较输出, 高电平有效
    PWMB_PS |= (1<<2);                              // 0:选择 P2.0, 1:选择 P1.7, 2:选择 P0.0, 3:选择 P7.4
    PWMB_ENO |= 0x04;                               // IO 输出允许, bit6: ENO8P, bit4: ENO7P,
                                                    // bit2: ENO6P,   bit0: ENO5P
    PWMB_IER |= 0x04;                               // 使能通道 2 匹配中断

    PWMB_BKR = 0x80;                                // 主输出使能 相当于总开关
    PWMB_CR1 = 0x81;                                // 使能计数器, 允许自动重装载寄存器缓冲,
                                                    // 边沿对齐模式, 向上计数,  bit7=1:写自动重装载
                                                    // 寄存器缓冲(本周期不会被打扰), =0:直接写自动
                                                    // 重装载寄存器本(周期可能会乱掉)
}

//========================================================================
// 函数: void PWMB_ISR(void) interrupt 27
// 描述: PWMB 中断处理程序.
//========================================================================
void PWMB_ISR(void) interrupt 27
{
    unsigned char   sr1;

    sr1 = PWMB_SR1;                                 //为了快速,中断标志用一个局部变量处理
    PWMB_SR1 = 0;                                   //清除中断标志
    PWMB_SR2 = 0;                                   //清除中断标志

    if(sr1 & 0x01)                                  //更新中断,装载 PWM6 匹配输出高电平的时刻
    {
        PWMB_CCR6   = pwm1H;                        // 匹配值, PWM6 在 200 时刻输出高电平
        PWMB_CCMR2 = 0x70;                          // 通道模式配置, PWM 模式 2
        B_OutState = 0;
    }
    if(sr1 & 0x04)                                  //通道 2 匹配中断,装载 PWM6 匹配输出低电平的时刻
    {
        if(!B_OutState)
        {
            B_OutState = 1;
            PWMB_CCMR2 = 0x50;                      // 通道模式配置, 强制为有效电平
            PWMB_CCR6 = pwm2H;                      // 匹配值, PWM6 在 400 时刻输出低电平
            PWMB_CCMR2 = 0x60;                      // 通道模式配置, PWM 模式 1
        }
```

```
        else PWMB_CCMR2 = 0x40;                         // 通道模式配置，强制为无效电平
    }
}
```

25.10.16 带死区控制的 PWM 互补输出

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "stc8h.h"
#include "intrins.h"

void main(void)
{
    P_SW2 |= 0x80;                                      //使能 XFR 访问

    P0M1 = 0x00;
    P0M0 = 0xFF;
    P1M1 = 0x00;
    P1M0 = 0xFF;

    PWMA_ENO = 0xFF;                                    //IO 输出 PWM
    PWMA_PS = 0x00;                                     //00:PWM at P1

/*********************************************************
PWMx_duty = [CCRx/(ARR + 1)]*100
*********************************************************/
    PWMA_PSCRH = 0x00;                                  //预分频寄存器
    PWMA_PSCRL = 0x00;
    PWMA_DTR = 0x00;                                    //死区时间配置

    PWMA_CCMR1 = 0x68;                                  //通道模式配置
    PWMA_CCMR2 = 0x68;
    PWMA_CCMR3 = 0x68;
    PWMA_CCMR4 = 0x68;

    PWMA_ARRH = 0x08;                                   //自动重装载寄存器，计数器 overflow 点
    PWMA_ARRL = 0x00;

    PWMA_CCR1H = 0x04;                                  //计数器比较值
    PWMA_CCR1L = 0x00;
    PWMA_CCR2H = 0x02;
    PWMA_CCR2L = 0x00;
    PWMA_CCR3H = 0x01;
    PWMA_CCR3L = 0x00;
    PWMA_CCR4H = 0x01;
    PWMA_CCR4L = 0x00;

    PWMA_CCER1 = 0x55;                                  //配置通道输出使能和极性
    PWMA_CCER2 = 0x55;                                  //配置通道输出使能和极性

    PWMA_BKR = 0x80;                                    //主输出使能 相当于总开关
    PWMA_IER = 0x02;                                    //使能中断
    PWMA_CR1 = 0x01;                                    //使能计数器

    EA = 1;
    while (1);
```

```
}

void PWMA_ISR() interrupt 26
{
    if(PWMA_SR1 & 0X02)
    {
        P03 = ~P03;
        PWMA_SR1 &=~0X02;
    }
}
```

25.10.17 利用 PWM 实现互补 SPWM

高级 PWM 定时器 PWM1P/PWM1N，PWM2P/PWM2N，PWM3P/PWM3N，PWM4P/PWM4N 每个通道都可独立实现 PWM 输出，或者两两互补对称输出。演示使用 PWM1P，PWM1N 产生互补的 SPWM。主时钟选择 24MHZ，PWM 时钟选择 1T，PWM 周期 2400，死区 12 个时钟(0.5us)，正弦波表用 200 点，输出正弦波频率 = 24000000 / 2400 / 200 = 50 HZ。

本程序仅仅是一个 SPWM 的演示程序，用户可以通过上面的计算方法修改 PWM 周期和正弦波的点数和幅度。本程序输出频率固定，如果需要变频，请用户自己设计变频方案。

C 语言代码

```
//测试工作频率为24MHz

#include "stc8h.h"
#include "intrins.h"

#define    MAIN_Fosc        24000000L              //定义主时钟

typedef    unsigned char    u8;
typedef    unsigned int     u16;
typedef    unsigned long    u32;

/*****************************  用户定义宏  ***********************************/

#define    PWMA_ENO         (*(unsigned char  volatile xdata *)  0xFEB1)
#define    PWMA_PS          (*(unsigned char  volatile xdata *)  0xFEB2)
#define    PWMB_ENO         (*(unsigned char  volatile xdata *)  0xFEB5)
#define    PWMB_PS          (*(unsigned char  volatile xdata *)  0xFEB6)

#define    PWMA_CR1         (*(unsigned char  volatile xdata *)  0xFEC0)
#define    PWMA_CR2         (*(unsigned char  volatile xdata *)  0xFEC1)
#define    PWMA_SMCR        (*(unsigned char  volatile xdata *)  0xFEC2)
#define    PWMA_ETR         (*(unsigned char  volatile xdata *)  0xFEC3)
#define    PWMA_IER         (*(unsigned char  volatile xdata *)  0xFEC4)
#define    PWMA_SR1         (*(unsigned char  volatile xdata *)  0xFEC5)
#define    PWMA_SR2         (*(unsigned char  volatile xdata *)  0xFEC6)
#define    PWMA_EGR         (*(unsigned char  volatile xdata *)  0xFEC7)
#define    PWMA_CCMR1       (*(unsigned char  volatile xdata *)  0xFEC8)
#define    PWMA_CCMR2       (*(unsigned char  volatile xdata *)  0xFEC9)
#define    PWMA_CCMR3       (*(unsigned char  volatile xdata *)  0xFECA)
#define    PWMA_CCMR4       (*(unsigned char  volatile xdata *)  0xFECB)
#define    PWMA_CCER1       (*(unsigned char  volatile xdata *)  0xFECC)
#define    PWMA_CCER2       (*(unsigned char  volatile xdata *)  0xFECD)
#define    PWMA_CNTRH       (*(unsigned char  volatile xdata *)  0xFECE)
#define    PWMA_CNTRL       (*(unsigned char  volatile xdata *)  0xFECF)
```

```
#define   PWMA_PSCRH    (*(unsigned char  volatile xdata *)  0xFED0)
#define   PWMA_PSCRL    (*(unsigned char  volatile xdata *)  0xFED1)
#define   PWMA_ARRH     (*(unsigned char  volatile xdata *)  0xFED2)
#define   PWMA_ARRL     (*(unsigned char  volatile xdata *)  0xFED3)
#define   PWMA_RCR      (*(unsigned char  volatile xdata *)  0xFED4)
#define   PWMA_CCR1H    (*(unsigned char  volatile xdata *)  0xFED5)
#define   PWMA_CCR1L    (*(unsigned char  volatile xdata *)  0xFED6)
#define   PWMA_CCR2H    (*(unsigned char  volatile xdata *)  0xFED7)
#define   PWMA_CCR2L    (*(unsigned char  volatile xdata *)  0xFED8)
#define   PWMA_CCR3H    (*(unsigned char  volatile xdata *)  0xFED9)
#define   PWMA_CCR3L    (*(unsigned char  volatile xdata *)  0xFEDA)
#define   PWMA_CCR4H    (*(unsigned char  volatile xdata *)  0xFEDB)
#define   PWMA_CCR4L    (*(unsigned char  volatile xdata *)  0xFEDC)
#define   PWMA_BKR      (*(unsigned char  volatile xdata *)  0xFEDD)
#define   PWMA_DTR      (*(unsigned char  volatile xdata *)  0xFEDE)
#define   PWMA_OISR     (*(unsigned char  volatile xdata *)  0xFEDF)

/*****************************************************************************/

#define   PWMA_1        0x00                    //P:P1.0   N:P1.1
#define   PWMA_2        0x01                    //P:P2.0   N:P2.1
#define   PWMA_3        0x02                    //P:P6.0   N:P6.1

#define   PWMB_1        0x00                    //P:P1.2   N:P1.3
#define   PWMB_2        0x04                    //P:P2.2   N:P2.3
#define   PWMB_3        0x08                    //P:P6.2   N:P6.3

#define   PWM3_1        0x00                    //P:P1.4   N:P1.5
#define   PWM3_2        0x10                    //P:P2.4   N:P2.5
#define   PWM3_3        0x20                    //P:P6.4   N:P6.5

#define   PWM4_1        0x00                    //P:P1.6   N:P1.7
#define   PWM4_2        0x40                    //P:P2.6   N:P2.7
#define   PWM4_3        0x80                    //P:P6.6   N:P6.7
#define   PWM4_4        0xC0                    //P:P3.4   N:P3.3

#define   ENO1P         0x01
#define   ENO1N         0x02
#define   ENO2P         0x04
#define   ENO2N         0x08
#define   ENO3P         0x10
#define   ENO3N         0x20
#define   ENO4P         0x40
#define   ENO4N         0x80

/*************  本地变量声明  **************/

unsigned int code T_SinTable[]=
{
    1220, 1256, 1292, 1328, 1364, 1400, 1435, 1471,
    1506, 1541, 1575, 1610, 1643, 1677, 1710, 1742,
    1774, 1805, 1836, 1866, 1896, 1925, 1953, 1981,
    2007, 2033, 2058, 2083, 2106, 2129, 2150, 2171,
    2191, 2210, 2228, 2245, 2261, 2275, 2289, 2302,
    2314, 2324, 2334, 2342, 2350, 2356, 2361, 2365,
    2368, 2369, 2370, 2369, 2368, 2365, 2361, 2356,
    2350, 2342, 2334, 2324, 2314, 2302, 2289, 2275,
    2261, 2245, 2228, 2210, 2191, 2171, 2150, 2129,
    2106, 2083, 2058, 2033, 2007, 1981, 1953, 1925,
```

```
    1896, 1866, 1836, 1805, 1774, 1742, 1710, 1677,
    1643, 1610, 1575, 1541, 1506, 1471, 1435, 1400,
    1364, 1328, 1292, 1256, 1220, 1184, 1148, 1112,
    1076, 1040, 1005,  969,  934,  899,  865,  830,
     797,  763,  730,  698,  666,  635,  604,  574,
     544,  515,  487,  459,  433,  407,  382,  357,
     334,  311,  290,  269,  249,  230,  212,  195,
     179,  165,  151,  138,  126,  116,  106,   98,
      90,   84,   79,   75,   72,   71,   70,   71,
      72,   75,   79,   84,   90,   98,  106,  116,
     126,  138,  151,  165,  179,  195,  212,  230,
     249,  269,  290,  311,  334,  357,  382,  407,
     433,  459,  487,  515,  544,  574,  604,  635,
     666,  698,  730,  763,  797,  830,  865,  899,
     934,  969, 1005, 1040, 1076, 1112, 1148, 1184,
};

u16 PWMA_Duty;
u8 PWM_Index;                                   //SPWM 查表索引

/******************** 主函数 **************************/
void main(void)
{
    P_SW2 |= 0x80;

    P0M1 = 0;   P0M0 = 0;   //设置为准双向口
    P1M1 = 0;   P1M0 = 0;   //设置为准双向口
    P2M1 = 0;   P2M0 = 0;   //设置为准双向口
    P3M1 = 0;   P3M0 = 0;   //设置为准双向口
    P4M1 = 0;   P4M0 = 0;   //设置为准双向口
    P5M1 = 0;   P5M0 = 0;   //设置为准双向口
    P6M1 = 0;   P6M0 = 0;   //设置为准双向口
    P7M1 = 0;   P7M0 = 0;   //设置为准双向口

    PWMA_Duty = 1220;

    PWMA_CCER1 = 0x00;                          //写 CCMRx 前必须先清零 CCxE 关闭通道
    PWMA_CCER2 = 0x00;
    PWMA_CCMR1 = 0x60;                          //通道模式配置
//  PWMA_CCMR2 = 0x60;
//  PWMA_CCMR3 = 0x60;
//  PWMA_CCMR4 = 0x60;
    PWMA_CCER1 = 0x05;                          //配置通道输出使能和极性
//  PWMA_CCER2 = 0x55;

    PWMA_ARRH = 0x09;                               //设置周期时间
    PWMA_ARRL = 0x60;

    PWMA_CCR1H = (u8)(PWMA_Duty >> 8);          //设置占空比时间
    PWMA_CCR1L = (u8)(PWMA_Duty);

    PWMA_DTR = 0x0C;                            //设置死区时间

    PWMA_ENO = 0x00;
    PWMA_ENO |= ENO1P;                          //使能输出
    PWMA_ENO |= ENO1N;                          //使能输出
//  PWMA_ENO |= ENO2P;                          //使能输出
//  PWMA_ENO |= ENO2N;                          //使能输出
//  PWMA_ENO |= ENO3P;                          //使能输出
```

```
//      PWMA_ENO |= ENO3N;                          //使能输出
//      PWMA_ENO |= ENO4P;                          //使能输出
//      PWMA_ENO |= ENO4N;                          //使能输出

        PWMA_PS = 0x00;                             //高级 PWM 通道输出脚选择位
        PWMA_PS |= PWMA_3;                          //选择 PWMA_3 通道
//      PWMA_PS |= PWMB_3;                          //选择 PWMB_3 通道
//      PWMA_PS |= PWM3_3;                          //选择 PWM3_3 通道
//      PWMA_PS |= PWM4_3;                          //选择 PWM4_3 通道

        PWMA_BKR = 0x80;                            //使能主输出
        PWMA_IER = 0x01;                            //使能中断
        PWMA_CR1 |= 0x01;                           //开始计时

        EA = 1;                                     //打开总中断

        while (1)
        {
        }
}

/********************  中断函数  ************************/
void PWMA_ISR() interrupt 26
{
        if (PWMA_SR1 & 0x01)
        {
                PWMA_SR1 &=~0x01;
                PWMA_Duty = T_SinTable[PWM_Index];
                if (++PWM_Index >= 200)
                        PWM_Index = 0;

                PWMA_CCR1H = (u8)(PWMA_Duty >> 8);          //设置占空比时间
                PWMA_CCR1L = (u8)(PWMA_Duty);
        }
        PWMA_SR1 = 0;
}
```

25.10.18 产生 3 路相位差 120 度的互补 PWM 波形（网友提供）

```
//测试工作频率为 24MHz

/*---------------------------------------------------
主要功能: P2.0-P2.5 输出互补的三路相位差 120 度的 PWM
第 1 路 P2.0/P2.1 为 PWM 输出模式，第 2 路 P2.2/P2.3 和第 3 路 P2.4/P2.5 为比较输出模式
程序下载进目标芯片,输出 50hz 的 SPWM,占空比 25%
---------------------------------------------------*/

#include "stc8h.h"

#define     FOSC                24000000UL

#define     PWM_PSC             (240-1)             //定义 PWM 时钟预分频系数
#define     PWM_PERIOD          2000                //定义 PWM 周期值
                                                    //(频率=FOSC/(PWM_PSC+1)/PWM_PERIOD=50Hz)
#define     PWM_DUTY            500                 //定义 PWM 的占空比值
```

```
                                                    //(占空比=PWM_DUTY/PWM_PERIOD*100%=25%)

void SYS_Init();
void PWM_Init();

void main()
{
    SYS_Init();
    PWM_Init();

    EA = 1;                                         //打开总中断

    while (1);
}

void SYS_Init()
{
    P_SW2 |= 0x80;                                  //扩展寄存器(XFR)访问使能

    P0M1 = 0x00; P0M0 = 0x00;
    P1M1 = 0x00; P1M0 = 0x00;
    P2M1 = 0x00; P2M0 = 0x00;
    P3M1 = 0x00; P3M0 = 0x00;
    P4M1 = 0x00; P4M0 = 0x00;
    P5M1 = 0x00; P5M0 = 0x00;
    P6M1 = 0x00; P6M0 = 0x00;
    P7M1 = 0x00; P7M0 = 0x00;
}

void PWM_Init()
{
    PWMA_PSCR = PWM_PSC;                            //配置预分频系数

    PWMA_CCER1 = 0x00;                              //写 CCMRx 前必须先清零 CCxE 关闭通道
    PWMA_CCER2 = 0x00;

    PWMA_CCMR1 = 0x60;                              //通道模式配置 PWM 模式 1
    PWMA_CCMR2 = 0x30;                              //通道模式配置输出比较模式
    PWMA_CCMR3 = 0x30;                              //通道模式配置输出比较模式

    PWMA_CCER1 = 0x55;                              //配置通道 1,2,3 输出使能和极性
    PWMA_CCER2 = 0x05;

    PWMA_ARR = PWM_PERIOD;                          //设置周期时间

    PWMA_ENO = 0x3f;                                //使能 PWM 输出
    PWMA_PS = 0x15;                                 //高级 PWM 通道输出脚选择 P2.0-P2.5

    PWMA_CCR1 = PWM_DUTY;                           //设置占空比时间
    PWMA_CCR2 = PWM_PERIOD/3;                       //设置 OC2 起始翻转位
    PWMA_CCR3 = PWM_PERIOD/3*2;                     //设置 OC3 起始翻转位

    PWMA_IER = 0x0d;                                //使能 OC2/OC3 比较中断,更新中断

    PWMA_BKR = 0x80;                                //使能主输出
    PWMA_CR1 |= 0x01;                               //开始计时
}

void PWMA_ISR() interrupt 26
```

```
{
    if (PWMA_SR1 & 0x01)
    {
        PWMA_CCR2 = PWM_PERIOD/3;                   //设置占空比时间
        PWMA_CCR3 = PWM_PERIOD/3*2;                 //设置占空比时间
        PWMA_SR1 &= ~0x01;
    }
    else if (PWMA_SR1 & 0x04)
    {
        PWMA_CCR2 = (PWM_PERIOD/3+PWM_DUTY);    //设置 0C2 结束翻转位
        PWMA_SR1 &= ~0x04;
    }
    else if (PWMA_SR1 & 0x08)
    {
        PWMA_CCR3 = (PWM_PERIOD/3*2+PWM_DUTY); //设置 0C3 结束翻转位
        PWMA_SR1 &= ~0x08;
    }
    else
    {
        PWMA_SR1 = 0;
    }
}
```

25.10.19 使用 PWM 的 CEN 启动 PWMA 定时器，实时触发 ADC

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "stc8h.h"
#include "intrins.h"

void delay()
{
    int i;
    for (i=0; i<100; i++);
}

void main()
{
    P_SW2 |= 0x80;                                  //使能访问 XFR，没有冲突不用关闭

    P1M0 = 0x00;
    P1M1 = 0x01;
    P3M0 = 0x00;
    P3M1 = 0x00;

    ADC_CONTR = ADC_POWER | ADC_EPWMT | 0;          //选择 P1.0 为 ADC 输入通道
    delay();                                        //等待 ADC 电源稳定
    EADC = 1;

    PWMA_CR2 = 0x10;                                //CEN 信号为 TRGO,可用于触发 ADC
    PWMA_ARR = 5000;
    PWMA_IER = 0x01;
    PWMA_CR1 = 0x01;                                //设置 CEN 启动 PWMA 定时器，实时触发 ADC
    EA = 1;

    while (1);
```

```
}

void ADC_ISR() interrupt 5
{
    ADC_CONTR &= ~ADC_FLAG;
}

void PWMA_ISR() interrupt 26
{
    if(PWMA_SR1 & 0x01)
    {
        PWMA_SR1 &=~0x01;
    }
}
```

25.10.20 PWM 周期重复触发 ADC

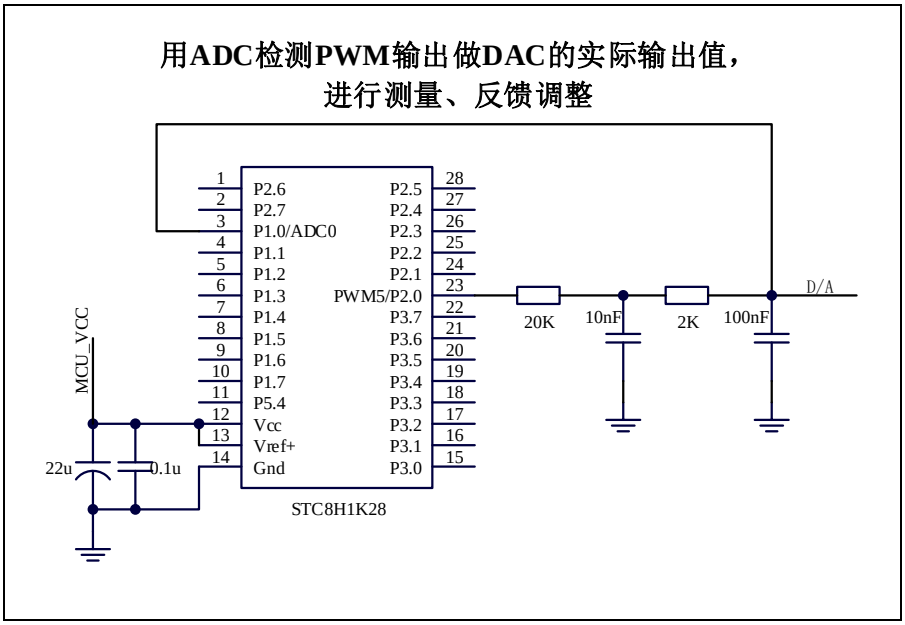
C 语言代码

```
//测试工作频率为 11.0592MHz

#include "stc8h.h"
#include "intrins.h"

void delay()
{
    int i;
    for (i=0; i<100; i++);
}

void main()
{
    P_SW2 |= 0x80;                          //使能访问 XFR，没有冲突不用关闭

    P1M0 = 0x00;
    P1M1 = 0x01;
    P3M0 = 0x00;
    P3M1 = 0x00;

    ADC_CONTR = ADC_POWER | ADC_EPWMT | 0;  //选择 P1.0 为 ADC 输入通道
    delay();                                //等待 ADC 电源稳定
    EADC = 1;

    PWMA_CR2 = 0x20;                        //周期更新事件为 TRGO,用于周期触发 ADC
    PWMA_ARR = 5000;
    PWMA_IER = 0x01;
    PWMA_CR1 = 0x01;                        //设置 CEN 启动 PWMA 定时器
    EA = 1;

    while (1);
}

void ADC_ISR() interrupt 5
{
    ADC_CONTR &= ~ADC_FLAG;
}

void PWMA_ISR() interrupt 26
```

```
{
    if(PWMA_SR1 & 0x01)
    {
        PWMA_SR1 &=~0x01;
    }
}
```

25.10.21 利用 PWM 实现 16 位 DAC 的参考线路图

STC8H 系列单片机的高级 PWM 定时器可输出 16 位的 PWM 波形，再经过两级低通滤波即可产生 16 位的 DAC 信号，通过调节 PWM 波形的高电平占空比即可实现 DAC 信号的改变。应用线路图如下图所示，输出的 DAC 信号可输入到 MCU 的 ADC 进行反馈测量。



25.10.22 正交编码器模式

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "stc8h.h"
#include "intrins.h"

unsigned char cnt_H, cnt_L;

void main(void)
{
    P_SW2 |= 0x80;                          //使能 XFR 访问

    P1M1 = 0x0f;
    P1M0 = 0x00;

    PWMA_ENO = 0x00;                        //配置成 TRGI 的 pin 需关掉 ENO 对应 bit 并配成 input
    PWMA_PS = 0x00;                         //00:PWM at P1

    PWMA_PSCRH = 0x00;                      //预分频寄存器
    PWMA_PSCRL = 0x00;
```

```
    PWMA_CCMR1 = 0x21;                                  //通道模式配置为输入，接编码器,滤波器 4 时钟
    PWMA_CCMR2 = 0x21;                                  //通道模式配置为输入，接编码器,滤波器 4 时钟

    PWMA_SMCR  = 0x03;                                  //编码器模式 3

    PWMA_CCER1 = 0x55;                                  //配置通道使能和极性
    PWMA_CCER2 = 0x55;                                  //配置通道使能和极性

    PWMA_IER = 0x02;                                    //使能中断

    PWMA_CR1 |= 0x01;                                   //使能计数器

    EA = 1;

    while (1);
}

/********************* PWM 中断读编码器计数值*************************/
void PWMA_ISR() interrupt 26
{
    if (PWMA_SR1 & 0X02)
    {
        P03 = ~P03;
        cnt_H = PWMA_CCR1H;
        cnt_L = PWMA_CCR1L;
        PWMA_SR1 & = ~0X02;
    }
}
```

25.10.23 使用高级 PWM 实现编码器演示程序

C 语言代码

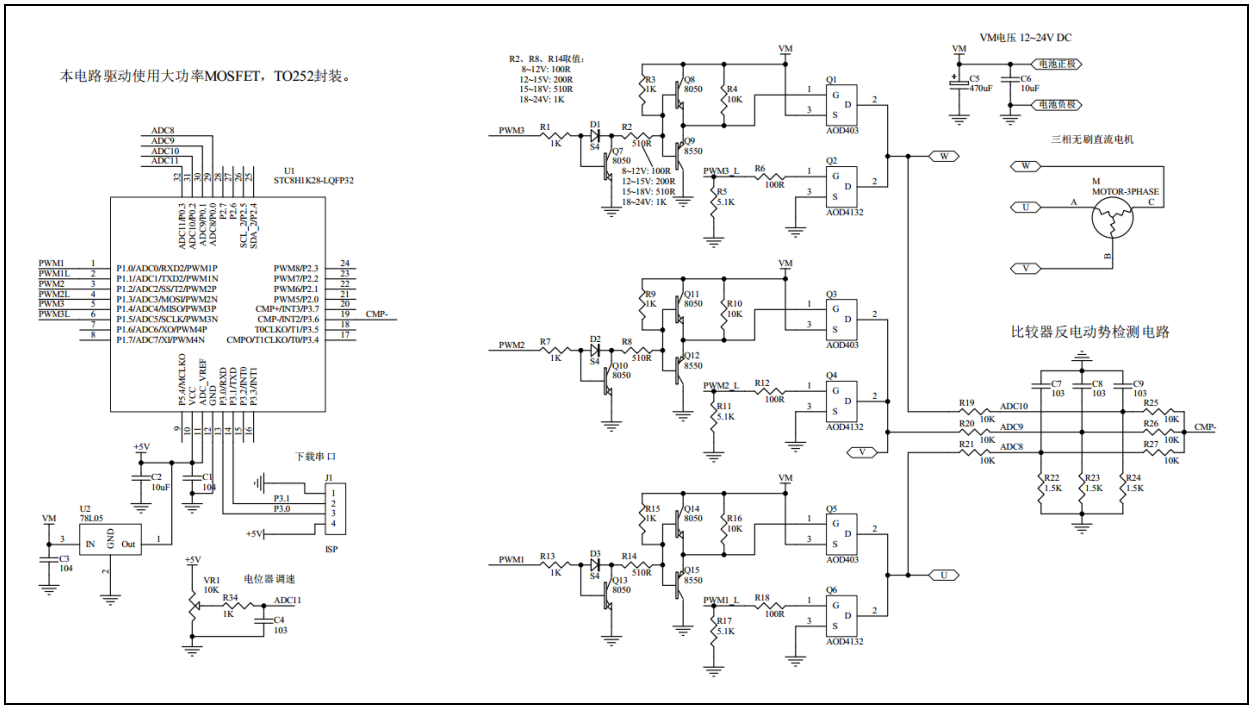
```
//测试工作频率为 11.0592MHz

#define    MAIN_Fosc    11059200L//定义主时钟

#include   <reg51.h>


/*************        功能说明**************

PWMA 模块工作于编码器模式. PWMA 模块只能接一个编码器.
串口 1(RXD-->P3.0   TXD-->P3.1) 返回读数结果, 串口设置 115200,8,n,1;
编码器 A 相输入: PWM1P (P1.0)
编码器 B 相输入: PWM2P (P1.2)

编码器模式,          模式 1: 每个脉冲两个边沿加减 2.
                      模式 2: 每个脉冲两个边沿加减 2.
                      模式 3: 每个脉冲两个边沿加减 4.

*****************************************/

sfr        P1M1   =    0x91;
sfr        P1M0   =    0x92;
sfr        AUXR   =    0x8E;
sfr        P_SW1 =    0xA2;
```

```
sfr         P_SW2 =         0xBA;

#define     PWMA_CNTR       (*(unsigned     int volatile xdata *)0xfece)
#define     PWMA_PSCR       (*(unsigned     int volatile xdata *)0xfed0)
#define     PWMA_ARR        (*(unsigned     int volatile xdata *)0xfed2)
#define     PWMA_CCMR1      (*(unsigned     char volatile xdata *)0xfec8)
#define     PWMA_CCMR2      (*(unsigned     char volatile xdata *)0xfec9)
#define     PWMA_CR1        (*(unsigned     char volatile xdata *)0xfec0)
#define     PWMA_SMCR       (*(unsigned     char volatile xdata *)0xfec2)
#define     PWMA_ENO        (*(unsigned     char volatile xdata *)0xfeb1)
#define     PWMA_PS         (*(unsigned     char volatile xdata *)0xfeb2)
#define     PWMA_CCER1      (*(unsigned     char volatile xdata *)0xfecc)
#define     PWMA_IER        (*(unsigned     char volatile xdata *)0xfec4)
#define     PWMA_SR1        (*(unsigned     char volatile xdata *)0xfec5)


unsigned int    pulse;                                  //编码器脉冲
bit             B_Change;                               //编码器计数改变

bit             B_TX1_Busy;                             // 发送忙标志

void            PWMA_config(void);
void            UART1_config(unsigned long brt);        // brt: 通信波特率
void            UART1_TxByte(unsigned char dat);


void main(void)
{
    unsigned int j;

    P_SW2 |= 0x80;                                      //使能 XFR 访问

    P1M1 = 0x00;
    P1M0 = 0x00;

    UART1_config(115200UL);                             // brt: 通信波特率

    EA = 1;

    PWMA_config();
    pulse = 10;

    while (1)
    {
        if(B_Change)
        {
            B_Change = 0;
            j = pulse;
            UART1_TxByte(j/10000+'0');                  //转成十进制文本并发送
            UART1_TxByte((j%10000)/1000+'0');
            UART1_TxByte((j%1000)/100+'0');
            UART1_TxByte((j%100)/10+'0');
            UART1_TxByte( j%10+'0');
            UART1_TxByte(0x0d);
            UART1_TxByte(0x0a);
        }
    }
}
```

```
//========================================================================
// 函数: void PWMA_config(void)
// 描述: PPWM 配置函数。
// 参数: noe.
// 返回: none.
// 版本: V1.0
// 备注:
//========================================================================
void PWMA_config(void)
{
    PWMA_PSCR = 0;                          //预分频寄存器,分频Fck_cnt = Fck_psc/(PSCR[15:0]+1),
                                            //边沿对齐 PWM 频率 =SYSclk/((PSCR+1)*(ARR+1)),
                                            //中央对齐 PWM 频率 =SYSclk/((PSCR+1)*(ARR+1)*2).
    PWMA_ARR = 0xffff;                      //自动重装载寄存器,控制 PWM 周期
    PWMA_CNTR = 0;                          //清零编码器计数器值
    PWMA_ENO = 0;                           //IO 禁止输出 PWM

    PWMA_CCMR1 = 0x01+(10<<4);              //通道 1 模式配置, 配置成输入通道,
                                            //0~15 对应输入滤波时钟数:
                                            //1 2 4 8 12 16  24 32 48 64 80 96 128 160 192 256
    PWMA_CCMR2 = 0x01+(10<<4);              //通道 2 模式配置, 配置成输入通道,
                                            //0~15 对应输入滤波时钟数:
                                            //1 2 4 8 12 16 24 32 48 64 80 96 128 160 192 256
    PWMA_SMCR = 2;                          //编码器模式,模式 1 或模式 2:每个脉冲两个边沿加减 2.
                                            //模式 3:  每个脉冲四个边沿加减 4.
    PWMA_CCER1 = 0x55;                      //配置通道输入使能和极性, 允许输入,下降沿
    PWMA_PS = 0;                            //IO 选择 P1.0 P1.2
    PWMA_IER = 0x02;                        //使能中断
    PWMA_CR1 = 0x01;                        //使能计数器, 允许自动重装载寄存器缓冲,
                                            //边沿对齐模式,      向上计数,
                                            // bit7=1:写自动重装载寄存器缓冲(本周期不会被打扰),
                                            //    =0:直接写自动重装载寄存器本(周期可能会乱掉)
}

//========================================================================
// 函数: void PWMA_ISR(void) interrupt PWMA_VECTOR
// 描述: PWMA 中断处理程序.
// 参数: None
// 返回: none.
// 版本: V1.0
//========================================================================
void PWMA_ISR(void) interrupt 26
{
    if(PWMA_SR1 & 0x02)                     //编码器中断
    {
        pulse = PWMA_CNTR;                  //读取当前编码器计数值
        B_Change = 1;                       //标志已有捕捉值
    }
    PWMA_SR1 = 0;
}

//========================================================================
// 函数: void        UART1_config(u32 brt)
// 描述: UART1 初始化函数。
// 参数:    brt:          通信波特率.
// 返回: none.
// 版本: VER1.0
// 备注:
```

```
//========================================================================
void UART1_config(unsigned long brt)                    // brt:  通信波特率
{
    Brt = 65536UL - (MAIN_Fosc / 4) / brt;
    AUXR &= ~0x01;                                      //S1 BRT Use Timer1;
    AUXR |= (1<<6);                                     //Timer1 set as 1T mode
    TMOD &= 0x0f;                                       //Timer1 16bits AutoReload;
    TH1 = (unsigned char)(brt >> 8);
    TL1 = (unsigned char)brt;
    TR1 = 1;                                            // 运行 Timer1
    P_SW1 &= ~0xc0;                                     //串口 1 切换到 P3.0 P3.1
    SCON = (SCON & 0x3f) | (1<<6);                      // 8 位数据, 1 位起始位, 1 位停止位, 无校验
    ES = 1;                                             //允许中断
    REN = 1;                                            //允许接收
}

//========================================================================
// 函数: void UART1_TxByte(u8 dat)
// 描述: 串口 1 查询发送一个字节函数.
// 参数: dat: 要发送的字节数据.
// 返回: none.
// 版本: VER1.0
// 备注:
//========================================================================
void UART1_TxByte(unsigned char dat)
{
    B_TX1_Busy = 1;                                     //标志发送忙
    SBUF = dat;                                         //发一个字节
    while(B_TX1_Busy);                                  //等待发送完成
}

//========================================================================
// 函数: void UART1_int    (void) interrupt UART1_VECTOR
// 描述: 串口 1 中断函数
// 参数: none.
// 返回: none.
// 版本: VER1.0
// 备注:
//========================================================================
void UART1_int (void) interrupt   4
{
    if(RI)
        RI = 0;

    if(TI)
    {
        TI = 0;
        B_TX1_Busy = 0;
    }
}
```

25.10.24 无 HALL 三相无刷电机驱动演示程序，电位器调速用



详细代码讲解及视频请上 STC 官网论坛：三相无刷电机驱动-STC8H-无 HALL，有视频讲解

C 语言代码

```
/************ 功能说明 **************

本程序试验使用 STC8H1K28-LQFP32 来驱动无传感器无刷三相直流电机.

P0.3 接的电位器控制转速，逆时针旋转电位器电压降低电机减速，顺时针旋转电位器电压升高电机加速.
关于无感三相无刷直流电机的原理，用户自行学习了解，本例不予说明.


*****************************************/

#define MAIN_Fosc       24000000L                   //定义主时钟

#include   "STC8Hxxx.h"


#define ADC_START  (1<<6)                           /*  自动清 0 */
#define ADC_FLAG   (1<<5)                           /*  软件清 0 */

#define    ADC_SPEED   1                            /* 0~15, ADC 时钟 = SYSclk/2/(n+1) */
#define    RES_FMT     (1<<5)

#define CSSETUP         (0<<7)
#define CSHOLD          (1<<5)
#define SMPDUTY         20

sbit PWM1   = P1^0;
sbit PWM1_L = P1^1;
sbit PWM2   = P1^2;
sbit PWM2_L = P1^3;
sbit PWM3   = P1^4;
sbit PWM3_L = P1^5;
```

```
u8    step;                                     //切换步骤
u8    PWM_Value;                                //  决定 PWM 占空比的值
bit   B_RUN;                                    //运行标志
u8    PWW_Set;                                  //目标 PWM 设置
u16   adc11;
bit   B_4ms;                                    //4ms 定时标志

u8    TimeOut;                                  //堵转超时
bit   B_start;                                  //启动模式
bit   B_Timer3_OverFlow;

u8    TimeIndex;                                //换相时间保存索引
u16   PhaseTimeTmp[8];                          //8 个换相时间, 其 sum/16 就是 30 度电角度
u16   PhaseTime;                                //换相时间计数
u8    XiaoCiCnt;                                //1:需要消磁, 2:正在消磁, 0 已经消磁


/*************************/

void  Delay_n_ms(u8 dly)                        // N ms 延时函数
{
    u16  j;
    do
    {
        j = MAIN_Fosc / 10000;
        while(--j)   ;
    }while(--dly);
}

void delay_us(u8 us)   //N us 延时函数
{
    do
    {
        NOP(20);  //@24MHz
    }
    while(--us);
}

//========================================================================
//  函数: u16    Get_ADC10bitResult(u8 channel))            //channel = 0~15
//========================================================================
u16  Get_ADC10bitResult(u8 channel)                 //channel = 0~15
{
    u8 i;
    ADC_RES = 0;
    ADC_RESL = 0;
    ADC_CONTR = 0x80 | ADC_START | channel;
    NOP(5);
//  while((ADC_CONTR & ADC_FLAG) == 0)              ;//等待 ADC 结束
        i = 255;
        while(i != 0)
        {
            i--;
            if((ADC_CONTR & ADC_FLAG) != 0) break;  //等待 ADC 结束
        }
    ADC_CONTR &= ~ADC_FLAG;
    return      ((u16)ADC_RES * 256 + (u16)ADC_RESL);
}
```

```
void  Delay_500ns(void)
{
    NOP(6);
}

void StepMotor(void)                                    // 换相序列函数
{
    switch(step)
    {
    case 0:   // AB   PWM1, PWM2_L=1
        PWMA_ENO = 0x00; PWM1_L=0; PWM3_L=0;
        Delay_500ns();
        PWMA_ENO = 0x01;                                // 打开 A 相的高端 PWM
        PWM2_L = 1;                                     // 打开 B 相的低端
        ADC_CONTR = 0x80+10;                            // 选择 P0.2 作为 ADC 输入 即 C 相电压
        CMPCR1 = 0x8c + 0x10;                           //比较器下降沿中断
        break;
    case 1:   // AC   PWM1, PWM3_L=1
        PWMA_ENO = 0x01; PWM1_L=0; PWM2_L=0;            // 打开 A 相的高端 PWM
        Delay_500ns();
        PWM3_L = 1;                                     // 打开 C 相的低端
        ADC_CONTR = 0x80+9;                             // 选择 P0.1 作为 ADC 输入 即 B 相电压
        CMPCR1 = 0x8c + 0x20;                           //比较器上升沿中断
        break;
    case 2:   // BC   PWM2, PWM3_L=1
        PWMA_ENO = 0x00; PWM1_L=0; PWM2_L=0;
        Delay_500ns();
        PWMA_ENO = 0x04;                                // 打开 B 相的高端 PWM
        PWM3_L = 1;                                     // 打开 C 相的低端
        ADC_CONTR = 0x80+8;                             // 选择 P0.0 作为 ADC 输入 即 A 相电压
        CMPCR1 = 0x8c + 0x10;                           //比较器下降沿中断
        break;
    case 3:   // BA   PWM2, PWM1_L=1
        PWMA_ENO = 0x04; PWM2_L=0; PWM3_L=0;            // 打开 B 相的高端 PWM
        Delay_500ns();
        PWM1_L = 1;                                     // 打开 C 相的低端
        ADC_CONTR = 0x80+10;                            // 选择 P0.2 作为 ADC 输入 即 C 相电压
        CMPCR1 = 0x8c + 0x20;                           //比较器上升沿中断
        break;
    case 4:   // CA   PWM3, PWM1_L=1
        PWMA_ENO = 0x00; PWM2_L=0; PWM3_L=0;
        Delay_500ns();
        PWMA_ENO = 0x10;                                // 打开 C 相的高端 PWM
        PWM1_L = 1;                                     // 打开 A 相的低端
        adc11 = ((adc11 *7)>>3) + Get_ADC10bitResult(11);
        ADC_CONTR = 0x80+9;                             // 选择 P0.1 作为 ADC 输入 即 B 相电压
        CMPCR1 = 0x8c + 0x10;                           //比较器下降沿中断
        break;
    case 5:   // CB   PWM3, PWM2_L=1
        PWMA_ENO = 0x10; PWM1_L=0; PWM3_L=0;            // 打开 C 相的高端 PWM
        Delay_500ns();
        PWM2_L = 1;                                     // 打开 B 相的低端
        ADC_CONTR = 0x80+8;                             // 选择 P0.0 作为 ADC 输入 即 A 相电压
        CMPCR1 = 0x8c + 0x20;                           //比较器上升沿中断
        break;

    default:
```

```
                break;
        }

        if(B_start)         CMPCR1 = 0x8C;                  // 启动时禁止下降沿和上升沿中断
}



void PWMA_config(void)
{
        P_SW2 |= 0x80;          //SFR enable

        PWM1     = 0;
        PWM1_L = 0;
        PWM2     = 0;
        PWM2_L = 0;
        PWM3     = 0;
        PWM3_L = 0;
        P1n_push_pull(0x3f);

        PWMA_PSCR = 3;                                  //预分频寄存器,分频 Fck_cnt = Fck_psc/(PSCR[15:0}+1),
                                                        //边沿对齐 PWM 频率 = SYSclk/((PSCR+1)*(AAR+1)),
                                                        //中央对齐 PWM 频率 = SYSclk/((PSCR+1)*AAR*2).
        PWMA_DTR    = 24;                               // 死区时间配置, n=0~127: DTR= n T,
                                                        //0x80 ~(0x80+n), n=0~63: DTR=(64+n)*2T,
                                                        //0xc0 ~(0xc0+n), n=0~31: DTR=(32+n)*8T,
                                                        //0xE0 ~(0xE0+n), n=0~31: DTR=(32+n)*16T,
        PWMA_ARR       = 255;                           // 自动重装载寄存器,   控制 PWM 周期
        PWMA_CCER1    = 0;
        PWMA_CCER2    = 0;
        PWMA_SR1       = 0;
        PWMA_SR2       = 0;
        PWMA_ENO       = 0;
        PWMA_PS        = 0;
        PWMA_IER       = 0;
//      PWMA_ISR_En = 0;

        PWMA_CCMR1    = 0x68;                           // 通道模式配置, PWM 模式 1,  预装载允许
        PWMA_CCR1     = 0;                              // 比较值,  控制占空比(高电平时钟数)
        PWMA_CCER1 |= 0x05;                             // 开启比较输出,  高电平有效
        PWMA_PS       |= 0;                             // 选择 IO, 0:P1.0 P1.1, 1:P2.0 P2.1, 2:P6.0 P6.1,
//      PWMA_IER      |= 0x02;                          // 使能中断

        PWMA_CCMR2    = 0x68;                           // 通道模式配置, PWM 模式 1,  预装载允许
        PWMA_CCR2     = 0;                              // 比较值,  控制占空比(高电平时钟数)
        PWMA_CCER1 |= 0x50;                             // 开启比较输出,  高电平有效
        PWMA_PS       |= (0<<2);                        // 选择 IO, 0:P1.2 P1.3, 1:P2.2 P2.3, 2:P6.2 P6.3,
//      PWMA_IER      |= 0x04;                          // 使能中断

        PWMA_CCMR3    = 0x68;                           // 通道模式配置, PWM 模式 1,  预装载允许
        PWMA_CCR3     = 0;                              // 比较值,  控制占空比(高电平时钟数)
        PWMA_CCER2 |= 0x05;                             // 开启比较输出,  高电平有效
        PWMA_PS       |= (0<<4);                        // 选择 IO, 0:P1.4 P1.5, 1:P2.4 P2.5, 2:P6.4 P6.5,
//      PWMA_IER      |= 0x08;                          // 使能中断

        PWMA_BKR       = 0x80;                          // 主输出使能  相当于总开关
        PWMA_CR1       = 0x81;                          // 使能计数器,  允许自动重装载寄存器缓冲,
                                                        //边沿对齐模式,  向上计数,   bit7=1:写自动重装载
                                                        //寄存器缓冲(本周期不会被打扰), =0:直接写自动
```

```
                                        //重装载寄存器本(周期可能会乱掉)
    PWMA_EGR    = 0x01;                 //产生一次更新事件, 清除计数器和与分频计数器,
                                        //装载预分频寄存器的值
//  PWMA_ISR_En = PWMA_IER;             //设置标志允许通道 1~4 中断处理
}

void ADC_config(void)                   //ADC 初始化函数(为了使用 ADC 输入端做比较器信号,
                                        //实际没有启动 ADC 转换)
{
    P1n_pure_input(0xc0);               //设置为高阻输入
    P0n_pure_input(0x0f);               //设置为高阻输入
    ADC_CONTR = 0x80 + 6;               //ADC on + channel
    ADCCFG = RES_FMT + ADC_SPEED;
    P_SW2 |=  0x80;                     //访问 XSFR
    ADCTIM = CSSETUP + CSHOLD + SMPDUTY;
}

void CMP_config(void)                   //比较器初始化程序
{
    CMPCR1 = 0x8C;                      // 1000 1100  打开比较器，P3.6 作为比较器的反相
                                        //输入端，ADC 引脚作为正输入端
    CMPCR2 = 60;                        //60 个时钟滤波    比较结果变化延时周期数, 0~63
    P3n_pure_input(0x40);               //CMP-(P3.6)设置为高阻.

    P_SW2 |= 0x80;                      //SFR enable
//  CMPEXCFG |= (0<<6);                 //bit7 bit6:  比较器迟滞输入选择: 0: 0mV,
                                        //1: 10mV, 2: 20mV, 3: 30mV
//  CMPEXCFG |= (0<<2);                 //bit2:  输入负极性选择, 0:  选择 P3.6 做输入,
                                        //1:  选择内部 BandGap 电压 BGv 做负输入.
//  CMPEXCFG |=  0;                     //bit1 bit0:  输入正极性选择, 0:  选择 P3.7 做输入,
                                        //1:  选择 P5.0 做输入,   2:  选择 P5.1 做输入,
                                        //3:  选择 ADC 输入(由 ADC_CHS[3:0]所选择的
                                        //ADC 输入端做正输入).
//  CMPEXCFG = (0<<6)+(0<<2)+3;
}

void CMP_ISR(void) interrupt 21         //比较器中断函数, 检测到反电动势过 0 事件
{
    u8  i;
    CMPCR1 &= ~0x40;                    //  需软件清除中断标志位

    if(XiaoCiCnt == 0)                  //消磁后才检测过 0 事件,    XiaoCiCnt=1:需要消磁,
                                        // =2:正在消磁, =0 已经消磁
    {
        T4T3M &= ~(1<<3);               // Timer3 停止运行
        if(B_Timer3_OverFlow)           //切换时间间隔(Timer3)有溢出
        {
            B_Timer3_OverFlow = 0;
            PhaseTime = 8000;           //换相时间最大 8ms, 2212 电机 12V 空转最高速 130us
                                        //切换一相(200RPS 12000RPM), 480mA
        }
        else
        {
            PhaseTime = (((u16)T3H << 8) + T3L) >> 1;   //单位为 1us
            if(PhaseTime >= 8000) PhaseTime = 8000;     //换相时间最大 8ms, 2212 电机 12V 空转最高速 130us
                                        //切换一相(200RPS 12000RPM), 480mA
        }
        T3H = 0;   T3L = 0;
        T4T3M |=  (1<<3);               //Timer3 开始运行
```

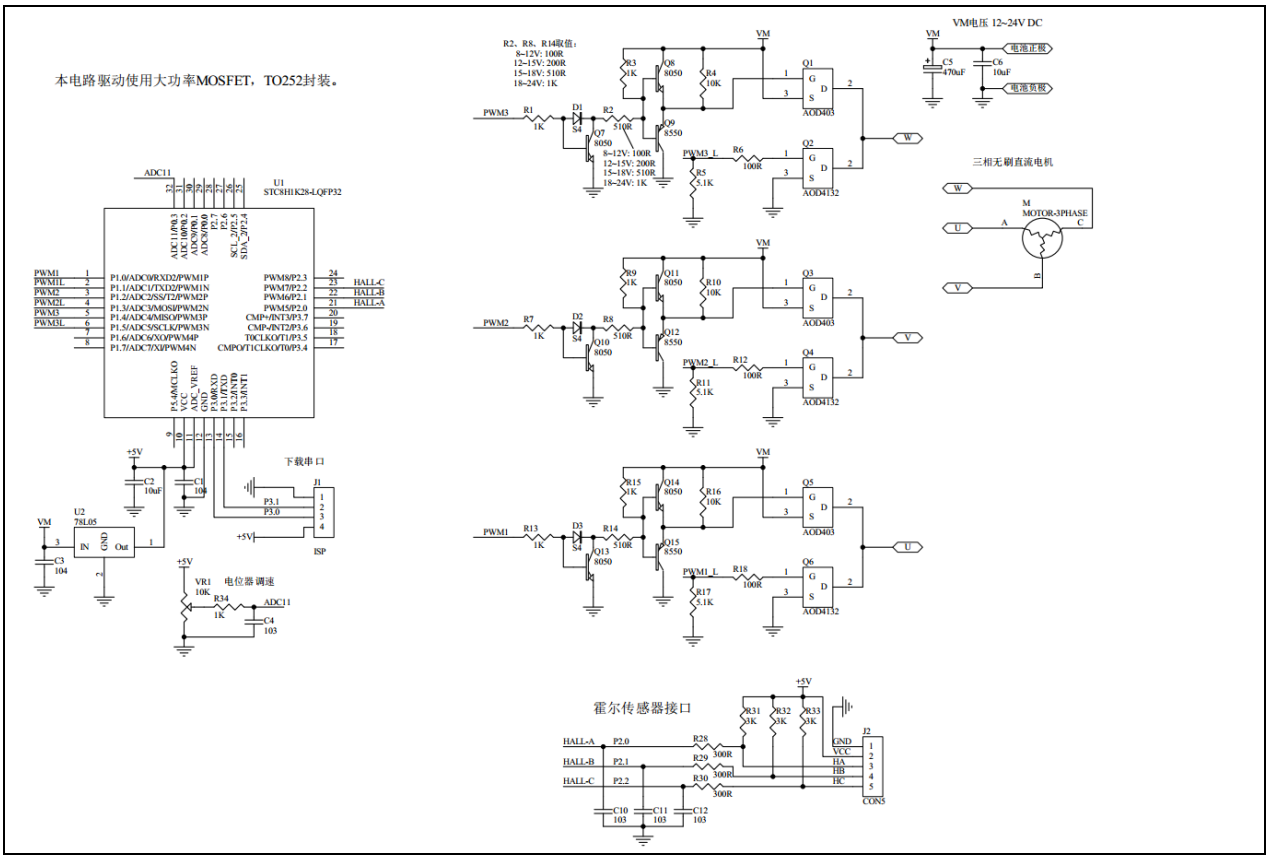
```
            PhaseTimeTmp[TimeIndex] = PhaseTime;                //保存一次换相时间
            if(++TimeIndex >= 8) TimeIndex = 0;                 //累加 8 次
            for(PhaseTime=0, i=0; i<8; i++) PhaseTime += PhaseTimeTmp[i];  //求 8 次换相时间累加和
            PhaseTime = PhaseTime >> 4;                         //求 8 次换相时间的平均值的一半, 即 30 度电角度
            if((PhaseTime >= 40) && (PhaseTime <= 1000)) TimeOut = 125;    //堵转 500ms 超时
            if( PhaseTime >= 60)  PhaseTime -= 40;              //修正由于滤波电容引起的滞后时间
            else   PhaseTime   = 20;

        //  PhaseTime = 20;                                     //只给 20us, 则无滞后修正, 用于检测滤波电容
                                                                //引起的滞后时间
            T4T3M &= ~(1<<7);                                   //Timer4 停止运行
            PhaseTime   = PhaseTime   << 1;                     //2 个计数 1us
            PhaseTime = 0 - PhaseTime;
            T4H = (u8)(PhaseTime >> 8);                         //装载 30 度角延时
            T4L = (u8)PhaseTime;
            T4T3M |=   (1<<7);                                  //Timer4 开始运行
            XiaoCiCnt = 1;                                      //1:需要消磁, 2:正在消磁, 0 已经消磁
        }
}

void Timer0_config(void)                                        //Timer0 初始化函数
{
        Timer0_16bitAutoReload();                               // T0 工作于 16 位自动重装
        Timer0_12T();
        TH0 = (65536UL-MAIN_Fosc/12 / 250) / 256; //4ms
        TL0 = (65536UL-MAIN_Fosc/12 / 250) % 256;
        TR0 = 1;                                                //  打开定时器 0

        ET0 = 1;                                                //  允许 ET0 中断
}

void Timer0_ISR(void) interrupt 1                               //Timer0 中断函数, 20us
{
        B_4ms = 1;        //4ms 定时标志
}

//========================== timer3 初始化函数 ================================================
void  Timer3_Config(void)
{
        P_SW2 |= 0x80;                                          //SFR enable
        T4T3M &= 0xf0;                                          //停止计数, 定时模式, 12T 模式, 不输出时钟
        T3H = 0;
        T3L = 0;

        T3T4PIN = 0x01;                                         //选择 IO, 0x00: T3--P0.4, T3CLKO--P0.5,
                                                                //T4--P0.6, T4CLKO--P0.7;
                                                                //0x01: T3--P0.0, T3CLKO--P0.1,
                                                                //T4--P0.2, T4CLKO--P0.3;
        IE2     |=    (1<<5);                                   //允许中断
        T4T3M |=    (1<<3);                                     //开始运行
}

//========================== timer4 初始化函数 ================================================
void  Timer4_Config(void)
{
        P_SW2 |= 0x80;                                          //SFR enable
        T4T3M &= 0x0f;                                          //停止计数, 定时模式, 12T 模式, 不输出时钟
        T4H = 0;
```

```
    T4L = 0;

    T3T4PIN = 0x01;                                 //选择 IO, 0x00: T3--P0.4, T3CLKO--P0.5,
                                                    //T4--P0.6, T4CLKO--P0.7;
                                                    //0x01: T3--P0.0, T3CLKO--P0.1,
                                                    //T4--P0.2, T4CLKO--P0.3;
    IE2     |=   (1<<6);                           //允许中断
//  T4T3M |=   (1<<7);                            //开始运行
}

//========================== timer3 中断函数 ============================================
void timer3_ISR (void) interrupt TIMER3_VECTOR
{
    B_Timer3_OverFlow = 1;                          //溢出标志
}

//========================== timer4 中断函数 ============================================
void timer4_ISR (void) interrupt TIMER4_VECTOR
{
    T4T3M &= ~(1<<7);                               //Timer4 停止运行
    if(XiaoCiCnt == 1)                              //标记需要消磁. 每次检测到过 0 事件后第一次
                                                    //中断为 30 度角延时, 设置消磁延时.
    {
        XiaoCiCnt = 2;                              //1:需要消磁, 2:正在消磁, 0 已经消磁
        if(B_RUN)                                   //电机正在运行
        {
            if(++step >= 6)   step = 0;
            StepMotor();
        }
                                                    //消磁时间, 换相后线圈(电感)电流减小到 0 的过程中,
                                                    //出现反电动势, 电流越大消磁时间越长,
                                                    //过 0 检测要在这个时间之后
                                                    //100%占空比时施加较重负载, 电机电流上升,
                                                    //可以示波器看消磁时间.
                                                    //只要在换相后延时几十 us 才检测过零, 就可以了
        T4H = (u8)((65536UL - 40*2) >> 8);          //装载消磁延时
        T4L = (u8)(65536UL - 40*2);
        T4T3M |=   (1<<7);                          //Timer4 开始运行
    }
    else if(XiaoCiCnt == 2)      XiaoCiCnt = 0;     //1:需要消磁, 2:正在消磁, 0 已经消磁
}


#define     D_START_PWM       30
/******************** 强制电机启动函数 ***************************/
void StartMotor(void)
{
    u16 timer,i;
    CMPCR1 = 0x8C;                                  // 关比较器中断

    PWM_Value   = D_START_PWM;                      // 初始占空比, 根据电机特性设置
    PWMA_CCR1L = PWM_Value;
    PWMA_CCR2L = PWM_Value;
    PWMA_CCR3L = PWM_Value;
    step = 0;    StepMotor();     Delay_n_ms(50);   //Delay_n_ms(250);// 初始位置
    timer = 200;                                    //风扇电机启动

    while(1)
    {
```

```
            for(i=0; i<timer; i++)  delay_us(100);          //根据电机加速特性，最高转速等等调整启动加速速度
            timer -= timer /16;
            if(++step >= 6)   step = 0;
            StepMotor();
            if(timer < 40)     return;
        }
}

/*********************************************/
void main(void)
{
        u8    i;
        u16   j;

        P2n_standard(0xf8);
        P3n_standard(0xbf);
        P5n_standard(0x10);

        adc11 = 0;

        PWMA_config();
        ADC_config();
        CMP_config();
        Timer0_config();                                    // Timer0 初始化函数
        Timer3_Config();                                    // Timer3 初始化函数
        Timer4_Config();                                    // Timer4 初始化函数
        PWW_Set = 0;
        TimeOut = 0;

        EA    = 1; //  打开总中断


        while (1)
        {
            if(B_4ms)                                       // 4ms 时隙
            {
                B_4ms = 0;

                if(TimeOut != 0)
                {
                    if(--TimeOut == 0)                      //堵转超时
                    {
                        B_RUN    = 0;
                        PWM_Value = 0;
                        CMPCR1 = 0x8C;                      //  关比较器中断
                        PWMA_ENO    = 0;
                        PWMA_CCR1L = 0; PWMA_CCR2L = 0; PWMA_CCR3L = 0;
                        PWM1_L=0; PWM2_L=0; PWM3_L=0;
                        Delay_n_ms(250);                    //堵转时,延时一点时间再启动
                    }
                }

                if(!B_RUN && (PWW_Set >= D_START_PWM)) //  占空比大于设定值，并且电机未运行，则启动电机
                {
                    B_start = 1;                            //启动模式
                    for(i=0; i<8; i++) PhaseTimeTmp[i] = 400;
                    StartMotor();                           //  启动电机
                    B_start = 0;
                    XiaoCiCnt = 0;                          //初始进入时
```

```
            CMPCR1 &= ~0x40;                    // 清除中断标志位
            if(step & 1)CMPCR1 = 0xAC;          //上升沿中断
            else        CMPCR1 = 0x9C;          //下降沿中断
            B_RUN = 1;
            Delay_n_ms(250);                    //延时一下, 先启动起来
            Delay_n_ms(250);
            TimeOut = 125;                      //启动超时时间 125*4 = 500ms
        }

        if(B_RUN)//正在运行中
        {
            if(PWM_Value < PWW_Set) PWM_Value++;    //油门跟随电位器
            if(PWM_Value > PWW_Set) PWM_Value--;
            if(PWM_Value < (D_START_PWM-10))    // 停转, 停转占空比 比 启动占空比 小 10/256
            {
                B_RUN = 0;
                PWM_Value = 0;
                CMPCR1 = 0x8C;                  // 关比较器中断
                PWMA_ENO   = 0;
                PWMA_CCR1L = 0; PWMA_CCR2L = 0; PWMA_CCR3L = 0;
                PWM1_L=0; PWM2_L=0; PWM3_L=0;
            }
            else
            {
                PWMA_CCR1L = PWM_Value;
                PWMA_CCR2L = PWM_Value;
                PWMA_CCR3L = PWM_Value;
            }
        }
        else
        {
            adc11 = ((adc11 *7)>>3) + Get_ADC10bitResult(11);
        }

        j = adc11;
        if(j != adc11)     j = adc11;
        PWW_Set = (u8)(j >> 5);                 //油门是 8 位的
    }
  }
}
```

25.10.25　带 HALL 三相无刷电机驱动演示程序，电位器调速，捕捉中断换相



详细代码讲解及视频请上 STC 官网论坛：三相无刷电机驱动-STC8H-带 HALL

C 语言代码

```
#include "stc8h.h"
/************ 功能说明 **************

本程序试验使用 STC8H1K28-LQFP32 来驱动带霍尔传感器的无刷三相直流电机.

PWM 的捕捉中断功能用来检测霍尔信号.
P0.3 接的电位器控制转速, 处于中间位置为停止, 逆时针旋转电位器电压降低电机为逆时针转, 顺时针旋转电位器电压升高电机为顺时针转.
关于带霍尔传感器三相无刷直流电机的原理, 用户自行学习了解, 本例不予说明.

*****************************************/


#define MAIN_Fosc       24000000L                   //定义主时钟

#include    "STC8Hxxx.h"


#define ADC_START  (1<<6)                           /*  自动清 0 */
#define ADC_FLAG   (1<<5)                           /*  软件清 0 */

#define     ADC_SPEED   1
#define     RES_FMT     (1<<5)
```

```
#define CSSETUP         (0<<7)
#define SMPDUTY         20

sbit PWM1     = P1^0;
sbit PWM1_L = P1^1;
sbit PWM2     = P1^2;
sbit PWM2_L = P1^3;
sbit PWM3     = P1^4;
sbit PWM3_L = P1^5;

u8    step;                                         //切换步骤
u8    PWM_Value;                                    //  决定 PWM 占空比的值
bit   B_RUN;                                        //运行标志
u8    PWW_Set;                                      //目标 PWM 设置
u8    YouMen;                                       //油门
bit   B_direct;                                     //转向, 0 顺时针, 1 逆时针
bit   B_4ms;                                        //4ms 定时标志

//========================================================================
//  函数: u16     Get_ADC10bitResult(u8 channel)    //channel = 0~15
//========================================================================
u16   Get_ADC10bitResult(u8 channel)                //channel = 0~15
{
    u8 i;
    ADC_RES = 0;
    ADC_RESL = 0;
    ADC_CONTR = 0x80 | ADC_START | channel;
    NOP(5);                                         //
    i = 255;                                        //超时限制
    while(i != 0)
    {
        i--;
        if((ADC_CONTR & ADC_FLAG) != 0) break;      //等待 ADC 结束
    }
    ADC_CONTR &= ~ADC_FLAG;
    return      ((u16)ADC_RES * 256 + (u16)ADC_RESL);
}


void  Delay_500ns(void)
{
    NOP(6);
}

void StepMotor(void)                                //  换相序列函数
{
    PWMB_IER    = 0;
    PWMB_CCER1 = 0;
    PWMB_CCER2 = 0;

    step = P2 & 0x07;                               //P2.0-HALL_A P2.1-HALL_B P2.2-HALL_C
    if(!B_direct)                                   //顺时针
    {
        switch(step)
        {
        case 2:   // 010, P2.0-HALL_A 下降沿   PWM3, PWM2_L=1       //顺时针
                  PWMA_ENO = 0x00; PWM1_L=0; PWM3_L=0;
                  Delay_500ns();
                  PWMA_ENO = 0x10;                  //  打开 C 相的高端 PWM
```

```
                PWM2_L = 1;                     // 打开 B 相的低端
                PWMB_CCER2 = (0x01+0x00);       //P2.2 0x01:允许输入捕获, +0x00:上升沿, +0x02: 下降沿
                PWMB_IER   = 0x08;              //P2.2 使能中断
                break;
        case 6: // 110, P2.2-HALL_C 上升沿  PWM3, PWM1_L=1
                PWMA_ENO = 0x10; PWM2_L=0; PWM3_L=0;    // 打开 C 相的高端 PWM
                Delay_500ns();
                PWM1_L = 1;                     // 打开 A 相的低端
                PWMB_CCER1 = (0x10+0x20);       //P2.1 0x10:允许输入捕获, +0x00:上升沿, +0x20: 下降沿
                PWMB_IER   = 0x04;              //P2.1 使能中断
                break;
        case 4: // 100, P2.1-HALL_B 下降沿  PWM2, PWM1_L=1
                PWMA_ENO = 0x00; PWM2_L=0; PWM3_L=0;
                Delay_500ns();
                PWMA_ENO = 0x04;                // 打开 B 相的高端 PWM
                PWM1_L = 1;                     // 打开 A 相的低端
                PWMB_CCER1 = (0x01+0x00);       //P2.0 0x01:允许输入捕获, +0x00:上升沿, +0x02: 下降沿
                PWMB_IER   = 0x02;              //P2.0 使能中断
                break;
        case 5: // 101, P2.0-HALL_A 上升沿  PWM2, PWM3_L=1
                PWMA_ENO = 0x04; PWM1_L=0; PWM2_L=0;  // 打开 B 相的高端 PWM
                Delay_500ns();
                PWM3_L = 1;                     // 打开 C 相的低端
                PWMB_CCER2 = (0x01+0x02);       //P2.2 0x01:允许输入捕获, +0x00:上升沿, +0x02: 下降沿
                PWMB_IER   = 0x08;              //P2.2 使能中断
                break;
        case 1: // 001, P2.2-HALL_C 下降沿  PWM1, PWM3_L=1
                PWMA_ENO = 0x00; PWM1_L=0; PWM2_L=0;
                Delay_500ns();
                PWMA_ENO = 0x01;                // 打开 A 相的高端 PWM
                PWM3_L = 1;                     // 打开 C 相的低端
                PWMB_CCER1 = (0x10+0x00);       //P2.1 0x10:允许输入捕获, +0x00:上升沿, +0x20: 下降沿
                PWMB_IER   = 0x04;              //P2.1 使能中断
                break;
        case 3: // 011, P2.1-HALL_B 上升沿  PWM1, PWM2_L=1
                PWMA_ENO = 0x01; PWM1_L=0; PWM3_L=0;  // 打开 A 相的高端 PWM
                Delay_500ns();
                PWM2_L = 1;                     // 打开 B 相的低端
                PWMB_CCER1 = (0x01+0x02);       //P2.0 0x01:允许输入捕获, +0x00:上升沿, +0x02: 下降沿
                PWMB_IER   = 0x02;              //P2.0 使能中断
                break;

        default:
                break;
        }
}

else  //逆时针
{
        switch(step)
        {
        case 4: // 100, P2.0-HALL_A 下降沿  PWM1, PWM2_L=1      //逆时针
                PWMA_ENO = 0x00; PWM1_L=0; PWM3_L=0;
                Delay_500ns();
                PWMA_ENO = 0x01;                // 打开 A 相的高端 PWM
                PWM2_L = 1;                     // 打开 B 相的低端
                PWMB_CCER1 = (0x10+0x00);       //P2.1 0x10:允许输入捕获, +0x00:上升沿, +0x20: 下降沿
                PWMB_IER   = 0x04;              //P2.1 使能中断
                break;
```

```
        case 6:   // 110, P2.1-HALL_B 上升沿   PWM1, PWM3_L=1
                  PWMA_ENO = 0x01; PWM1_L=0; PWM2_L=0;  // 打开 A 相的高端 PWM
                  Delay_500ns();
                  PWM3_L = 1;                          // 打开 C 相的低端
                  PWMB_CCER2 = (0x01+0x02);            //P2.2 0x01:允许输入捕获, +0x00:上升沿, +0x02: 下降沿
                  PWMB_IER   = 0x08;                   //P2.2  使能中断
                  break;
        case 2:   // 010, P2.2-HALL_C 下降沿   PWM2, PWM3_L=1
                  PWMA_ENO = 0x00; PWM1_L=0; PWM2_L=0;
                  Delay_500ns();
                  PWMA_ENO = 0x04;                     // 打开 B 相的高端 PWM
                  PWM3_L = 1;                          // 打开 C 相的低端
                  PWMB_CCER1 = (0x01+0x00);            //P2.0 0x01:允许输入捕获, +0x00:上升沿, +0x02: 下降沿
                  PWMB_IER   = 0x02;                   //P2.0  使能中断
                  break;
        case 3:   // 011, P2.0-HALL_A 上升沿   PWM2, PWM1_L=1
                  PWMA_ENO = 0x04; PWM2_L=0; PWM3_L=0;  // 打开 B 相的高端 PWM
                  Delay_500ns();
                  PWM1_L = 1;                          // 打开 A 相的低端
                  PWMB_CCER1 = (0x10+0x20);            //P2.1 0x10:允许输入捕获, +0x00:上升沿, +0x20: 下降沿
                  PWMB_IER   = 0x04;                   //P2.1  使能中断
                  break;
        case 1:   // 001, P2.1-HALL_B 下降沿   PWM3, PWM1_L=1
                  PWMA_ENO = 0x00; PWM2_L=0; PWM3_L=0;
                  Delay_500ns();
                  PWMA_ENO = 0x10;                     // 打开 C 相的高端 PWM
                  PWM1_L = 1;                          // 打开 A 相的低端
                  PWMB_CCER2 = (0x01+0x00);            //P2.2 0x01:允许输入捕获, +0x00:上升沿, +0x02: 下降沿
                  PWMB_IER   = 0x08;                   //P2.2  使能中断
                  break;
        case 5:   // 101, P2.2-HALL_C 上升沿   PWM3, PWM2_L=1
                  PWMA_ENO = 0x10; PWM1_L=0; PWM3_L=0;  // 打开 C 相的高端 PWM
                  Delay_500ns();
                  PWM2_L = 1;                          // 打开 B 相的低端
                  PWMB_CCER1 = (0x01+0x02);            //P2.0 0x01:允许输入捕获, +0x00:上升沿, +0x02: 下降沿
                  PWMB_IER   = 0x02;                   //P2.0  使能中断
                  break;

        default:
                  break;
        }
    }
}


void PWMA_config(void)
{
    P_SW2 |= 0x80;        //SFR enable

    PWM1    = 0;
    PWM1_L = 0;
    PWM2    = 0;
    PWM2_L = 0;
    PWM3    = 0;
    PWM3_L = 0;
    P1n_push_pull(0x3f);

    PWMA_PSCR = 3;                                     // 预分频寄存器,
                                                       //分频  Fck_cnt = Fck_psc/(PSCR[15:0}+1),
```

```
                                        //边沿对齐 PWM 频率 = SYSclk/((PSCR+1)*(AAR+1)),
                                        //中央对齐 PWM 频率 = SYSclk/((PSCR+1)*AAR*2).
    PWMA_DTR    = 24;                   // 死区时间配置, n=0~127: DTR= n T,
                                        //0x80 ~(0x80+n), n=0~63: DTR=(64+n)*2T,
                                        //0xc0 ~(0xc0+n), n=0~31: DTR=(32+n)*8T,
                                        //0xE0 ~(0xE0+n), n=0~31: DTR=(32+n)*16T,
    PWMA_ARR      = 255;                // 自动重装载寄存器,  控制 PWM 周期
    PWMA_CCER1    = 0;
    PWMA_CCER2    = 0;
    PWMA_SR1      = 0;
    PWMA_SR2      = 0;
    PWMA_ENO      = 0;
    PWMA_PS       = 0;
    PWMA_IER      = 0;
//  PWMA_ISR_En = 0;

    PWMA_CCMR1    = 0x68;               // 通道模式配置, PWM 模式 1,  预装载允许
    PWMA_CCR1     = 0;                  // 比较值,  控制占空比(高电平时钟数)
    PWMA_CCER1 |= 0x05;                 // 开启比较输出,  高电平有效
    PWMA_PS       |= 0;                 // 选择 IO, 0:P1.0 P1.1, 1:P2.0 P2.1, 2:P6.0 P6.1,
//  PWMA_IER      |= 0x02;              // 使能中断

    PWMA_CCMR2    = 0x68;               // 通道模式配置, PWM 模式 1,  预装载允许
    PWMA_CCR2     = 0;                  // 比较值,  控制占空比(高电平时钟数)
    PWMA_CCER1 |= 0x50;                 // 开启比较输出,  高电平有效
    PWMA_PS       |= (0<<2);            // 选择 IO, 0:P1.2 P1.3, 1:P2.2 P2.3, 2:P6.2 P6.3,
//  PWMA_IER      |= 0x04;              // 使能中断

    PWMA_CCMR3    = 0x68;               // 通道模式配置, PWM 模式 1,  预装载允许
    PWMA_CCR3     = 0;                  // 比较值,  控制占空比(高电平时钟数)
    PWMA_CCER2 |= 0x05;                 // 开启比较输出,  高电平有效
    PWMA_PS       |= (0<<4);            // 选择 IO, 0: P1.4 P1.5, 1: P2.4 P2.5, 2: P6.4 P6.5,
//  PWMA_IER      |= 0x08;              // 使能中断

    PWMA_BKR      = 0x80;               // 主输出使能  相当于总开关
    PWMA_CR1      = 0x81;               // 使能计数器,  允许自动重装载寄存器缓冲,
                                        //边沿对齐模式,
                                        // 向上计数,   bit7=1:写自动重装载寄存器缓冲
                                        //(本周期不会被打扰),
                                        //=0:直接写自动重装载寄存器本(周期可能会乱掉)
    PWMA_EGR      = 0x01;               //产生一次更新事件,  清除计数器和与分频计数器,
                                        //装载预分频寄存器的值
//  PWMA_ISR_En = PWMA_IER;             //设置标志允许通道 1~4 中断处理
}


//========================================================================
// 函数: void PWMB_config(void)
// 描述: PPWM 配置函数。
// 参数: noe.
// 返回: none.
// 版本: V1.0
// 备注:
//========================================================================
void PWMB_config(void)
{
    P_SW2 |= 0x80;                      //SFR enable

    PWMB_PSCR = 11;                     //预分频寄存器,分频 Fck_cnt = Fck_psc/(PSCR[15:0]+1),
```

```
                                                //边沿对齐 PWM 频率 = SYSclk/((PSCR+1)*(AAR+1)),
                                                //中央对齐 PWM 频率 = SYSclk/((PSCR+1)*AAR*2).
    PWMB_DTR    = 0;                            // 死区时间配置, n=0~127: DTR= n T,
                                                //0x80 ~(0x80+n), n=0~63: DTR=(64+n)*2T,
                                                //0xc0 ~(0xc0+n), n=0~31: DTR=(32+n)*8T,
                                                //0xE0 ~(0xE0+n), n=0~31: DTR=(32+n)*16T,
    PWMB_CCER1    = 0;
    PWMB_CCER2    = 0;
    PWMB_CR1      = 0;                          // 使能计数器, 允许自动重装载寄存器缓冲,
                                                //边沿对齐模式, 向上计数, bit7=1:写自动重装载
                                                //寄存器缓冲(本周期不会被打扰), =0:直接写自动
                                                //重装载寄存器本(周期可能会乱掉)
    PWMB_CR2      = 0;
    PWMB_SR1      = 0;
    PWMB_SR2      = 0;
    PWMB_ENO      = 0;
    PWMB_PS       = 0;
    PWMB_IER      = 0;

    PWMB_CCMR1    = 0x31;                       // 通道 5 模式配置, 配置成输入通道, 8 个时钟滤波
//  PWMB_CCER1 |= (0x01+0x02);                  // 0x01:允许输入捕获, +0x00:上升沿, +0x02: 下降沿
    PWMB_PS       |= 0;                         // 选择 IO, 0:P2.0, 1:P1.7, 2:P0.0, 3:P7.4,
//  PWMB_IER      |= 0x02;                      // 使能中断

    PWMB_CCMR2    = 0x31;                       // 通道 6 模式配置, 配置成输入通道, 8 个时钟滤波
//  PWMB_CCER1 |= (0x10+0x20);                  // 0x10:允许输入捕获, +0x00:上升沿, +0x20: 下降沿
    PWMB_PS       |= (0<<2);                    // 0:选择 P2.1, 1:选择 P5.4, 2:选择 P0.1, 3:选择 P7.5,
//  PWMB_IER      |= 0x04;                      // 使能中断

    PWMB_CCMR3    = 0x31;                       // 通道 7 模式配置, 配置成输入通道, 8 个时钟滤波
//  PWMB_CCER2 |= (0x01+0x02);                  // 0x01:允许输入捕获, +0x00:上升沿, +0x02: 下降沿
    PWMB_PS       |= (0<<4);                    // 选择 IO, 0:P2.2, 1:P3.3, 2:P0.2, 3:P7.6,
//  PWMB_IER      |= 0x08;                      // 使能中断

    PWMB_EGR      = 0x01;                       //产生一次更新事件, 清除计数器和预分频计数器,
                                                //装载预分频寄存器的值

    PWMB_SMCR     = 0x60;
    PWMB_BKR      = 0x00;                       //主输出使能 相当于总开关
    PWMB_CR1      |= 0x01;                      // 使能计数器, 允许自动重装载寄存器缓冲,
                                                //边沿对齐模式, 向上计数, bit7=1:写自动重
                                                //装载寄存器缓冲(本周期不会被打扰), =0:直接写自
                                                //动重装载寄存器本(周期可能会乱掉)

//  P2n_standard(0x07);                         //通道 5 6 7 输出 IO 设置为准双向口
}


//========================================================================
// 函数: void PWMB_ISR(void) interrupt PWMB_VECTOR
// 描述: PWMB 中断处理程序. 捕获数据通过 TIM1-> CCRnH / TIM1-> CCRnL 读取
// 参数: None
// 返回: none.
// 版本: V1.0,
//========================================================================
void PWMB_ISR(void) interrupt PWMB_VECTOR
{
    PWMB_SR1 = 0;                               //清除中断标志
    PWMB_SR2 = 0;                               //清除中断标志
```

```
    if(B_RUN) StepMotor();                          //换相
}


void ADC_config(void)                               //ADC 初始化函数
{
    P1n_pure_input(0xc0);                           //P1.7 P1.6 设置为高阻输入
    P0n_pure_input(0x0f);                           //P0.3~P0.0 设置为高阻输入
    ADC_CONTR = 0x80 + 6;                           //ADC on + channel
    ADCCFG = RES_FMT + ADC_SPEED;
    P_SW2 |=    0x80;                               //访问 XSFR
    ADCTIM = CSSETUP + CSHOLD + SMPDUTY;
}


void Timer0_config(void)                            //Timer0 初始化函数
{
    Timer0_16bitAutoReload();                       // T0 工作于 16 位自动重装
    Timer0_12T();
    TH0 = (65536UL-MAIN_Fosc/12 / 250) / 256;        //4ms
    TL0 = (65536UL-MAIN_Fosc/12 / 250) % 256;
    TR0 = 1;                                        //  打开定时器 0

    ET0 = 1;                                        //  允许 ET0 中断
}

void Timer0_ISR(void) interrupt 1                   //Timer0 中断函数
{
    B_4ms = 1;                                      //4ms 定时标志
//  if(B_RUN) StepMotor();                          //换相, 增加定时器里换相可以保证启动成功.
}


/*********************************************/
void main(void)
{
    P2n_standard(0xf8);
    P3n_standard(0xbf);
    P5n_standard(0x10);


    PWMA_config();
    PWMB_config();
    ADC_config();
    Timer0_config();        //Timer0 初始化函数
    PWW_Set = 0;

    EA   = 1; //  打开总中断

    while (1)
    {
        if(B_4ms)           // 4ms 时隙
        {
            B_4ms = 0;

            YouMen   = (u8)(Get_ADC10bitResult(11) >> 2);  //油门是 8 位的, P0.3 ADC11-->控制电位器输入
            if(YouMen >= 128) PWW_Set = YouMen - 128, B_direct = 0;  //顺时针
            else   PWW_Set = 127 - YouMen, B_direct = 1;     //逆时针
```

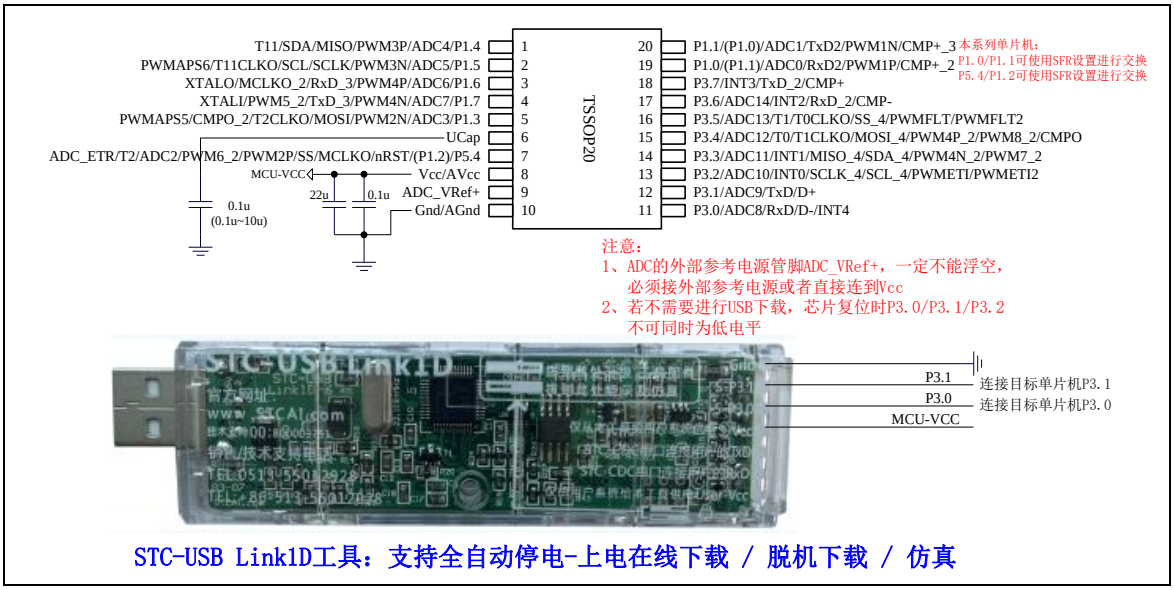
```
            PWW_Set *= 2;                           //PWM 设置值 0~254

            if(!B_RUN && (PWW_Set >= 30))           // PWM_Set >= 30，并且电机未运行，则启动电机
            {
                PWM_Value = 30;                     //启动电机的最低 PWM，根据具体电机而定
                PWMA_CCR1L = PWM_Value;             //输出 PWM
                PWMA_CCR2L = PWM_Value;
                PWMA_CCR3L = PWM_Value;
                B_RUN = 1;                          //标注运行
                StepMotor();                        //启动换相
            }

            if(B_RUN)//正在运行中
            {
                if(PWM_Value < PWW_Set) PWM_Value++;//油门跟随电位器，调速柔和
                if(PWM_Value > PWW_Set) PWM_Value--;
                if(PWM_Value < 20)                  // 停转
                {
                    B_RUN = 0;
                    PWMB_IER  = 0;
                    PWMB_CCER1 = 0;
                    PWMB_CCER2 = 0;
                    PWM_Value = 0;
                    PWMA_ENO  = 0;
                    PWMA_CCR1L = 0;
                    PWMA_CCR2L = 0;
                    PWMA_CCR3L = 0;
                    PWM1_L=0;
                    PWM2_L=0;
                    PWM3_L=0;
                }
                else
                {
                    PWMA_CCR1L = PWM_Value;
                    PWMA_CCR2L = PWM_Value;
                    PWMA_CCR3L = PWM_Value;
                }
            }
        }

    }
}
```

26　高级 PWM-硬件移相

产品线	高级 PWM 新增硬件移相功能
STC8H1K08 系列	
STC8H1K28 系列	
STC8H3K64S4 系列	
STC8H3K64S2 系列	
STC8H8K64U 系列	
STC8H4K64TL 系列	
STC8H4K64TLCD 系列 A 系列	
STC8H4K64TLCD 系列 A+系列	
STC8H1K08T 系列	
STC8H2K12U-A/B 系列	●
STC8H2K32U 系列	●
STC8G1K08-SOP8 系列	
STC8G1K08A-SOP8 系列	

26.1　PWM 硬件移相端口说明

STC8H2K12U 系列支持 PWM 硬件移相的端口为
P1.3（PWMAPS5）和 P1.5（PWMAPS6），如下管脚图所示：



STC-USB Link1D工具：支持全自动停电-上电在线下载 / 脱机下载 / 仿真

26.2 相关寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
PWMA_ENO2	PWMA 输出使能寄存器 2	F930H	-	-	-	-	-	ENO6	-	ENO5	xxxx,x0x0
PWMA_IOAUX2	PWMA 输出附加寄存器 2	F931H	-	-	-	-	-	AUX6	-	AUX6	xxxx,x0x0
PWMA_CR3	PWMA 控制寄存器 3	F932H	MMS2[3:0]				-	OIS6	-	OIS5	0000,x0x0
PWMA_SR3	PWMA 状态寄存器 3	F933H	-	-	-	-	-	-	CC6IF	CC5IF	xxxx,xx00
PWMA_CCER3	PWMA 捕获比较使能寄存器 3	F934H	-	-	CC6P	CC6E	-	-	CC5P	CC5E	xx00,xx00
PWMA_CCMR1X	PWMA 捕获比较模式寄存器 1x	F938H	-	-	-	-	-	-	-	OC1M[3]	xxx,xxx0
PWMA_CCMR2X	PWMA 捕获比较模式寄存器 2x	F939H	-	-	-	-	-	-	-	OC2M[3]	xxx,xxx0
PWMA_CCMR3X	PWMA 捕获比较模式寄存器 3x	F93AH	-	-	-	-	-	-	-	OC3M[3]	xxx,xxx0
PWMA_CCMR4X	PWMA 捕获比较模式寄存器 4x	F93BH	-	-	-	-	-	-	-	OC4M[3]	0000,0000
PWMA_CCMR5	PWMA 捕获比较模式寄存器 5	F93CH	OC5CE	OC5M[2:0]			OC5PE	-	-	-	0000,0xxx
PWMA_CCMR5X	PWMA 捕获比较模式寄存器 5x	F93DH	-	-	-	-	-	-	-	OC5M[3]	xxxx,xxx0
PWMA_CCMR6	PWMA 捕获比较模式寄存器 6	F93EH	OC6CE	OC6M[2:0]			OC6PE	-	-	-	000x,xxxx
PWMA_CCMR6X	PWMA 捕获比较模式寄存器 6x	F93FH	-	-	-	-	-	-	-	OC6M[3]	xxx,xxx0
PWMA_CCR5H	PWMA 捕获比较寄存器 5	F940H	CCR5[15:8]								0000,0000
PWMA_CCR5L	PWMA 捕获比较寄存器 5	F941H	CCR5[7:0]								0000,0000
PWMA_CCR5X	PWMA 捕获比较寄存器 5	F942H	GC5C3	GC5C2	GC5C1	-	-	-	-	-	000x,xxx
PWMA_CCR6H	PWMA 捕获比较寄存器 6	F943H	CCR6[15:8]								0000,0000
PWMA_CCR6L	PWMA 捕获比较寄存器 6	F944H	CCR6[7:0]								0000,0000

26.2.1 输出使能寄存器 2（PWMA_ENO2）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_ENO2	F930H	-	-	-	-	-	ENO6	-	ENO5

ENO6：PWMAPS6 输出控制位

0：禁止 PWMAPS6 输出

1：使能 PWMAPS6 输出

ENO5：PWMAPS5 输出控制位

0：禁止 PWMAPS5 输出

1：使能 PWMAPS5 输出

26.2.2 输出附加使能寄存器 2（PWMA_IOAUX2）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_IOAUX2	F931H	-	-	-	-	-	AUX6	-	AUX5

AUX6：PWMAPS6 输出附加控制位

0：PWMAPS6 的输出直接由 ENO6 控制

1：PWMAPS6 的输出由 ENO6 和 PWMA_BKR 共同控制

AUX5：PWMAPS5 输出附加控制位

0：PWMAPS5 的输出直接由 ENO5 控制

1：PWMAPS5 的输出由 ENO5 和 PWMA_BKR 共同控制

26.2.3 控制寄存器 3（PWMA_CR3）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CR3	F932H	MMS[3:0]				-	OIS6	-	OIS5

MMS[3:0]：主模式选择 2

MMS[3:0]	模式	说明
0000	复位	PWMA_EGR寄存器的UG位被用于作为触发输出（TRGO2）。如果复位由触发输入生成（从模式控制器配置为复位模式），则TRGO2上的信号相比实际复位会有延迟。
0001	使能	计数器使能信号CNT_EN用作触发输出(TRGO2)。该触发输出可用于同时启动多个定时器，或者控制在一段时间内使能从定时器。计数器使能信号由CEN控制位与门控模式下的触发输入的逻辑或运算组合而成。当计数器使能信号由触发输入控制时，TRGO2上会存在延迟，选择主/从模式时除外（请参见PWMA_SMCR寄存器中MSM位的说明）。
0010	更新	更新事件被选为触发输出（TRGO2）
0011	比较脉冲	CC1IF 标志置1时（即使已为高），只要发生捕获或比较匹配，触发输出(TRGO2)都会发送一个正脉冲。
0100	比较	OC1REF信号用作触发输出（TRGO2）
0101	比较	OC2REF信号用作触发输出（TRGO2）
0110	比较	OC3REF信号用作触发输出（TRGO2）
0111	比较	OC4REF信号用作触发输出（TRGO2）
1000	比较	OC5REF信号用作触发输出（TRGO2）
1001	比较	OC6REF信号用作触发输出（TRGO2）
1010	比较脉冲	OC4REF 上升沿或下降沿时，TRGO2 上生成脉冲
1011	比较脉冲	OC6REF 上升沿或下降沿时，TRGO2 上生成脉冲
1100	比较脉冲	OC4REF 或 OC6REF 上升沿时，TRGO2 上生成脉冲
1101	比较脉冲	OC4REF 上升沿或 OC6REF 下降沿时，TRGO2 上生成脉冲
1110	比较脉冲	OC5REF 或 OC6REF 上升沿时，TRGO2 上生成脉冲
1111	比较脉冲	OC5REF 上升沿或 OC6REF 下降沿时，TRGO2 上生成脉冲

注：必须先使能从定时器或 ADC 的时钟，才能从主定时器接收事件；并且从主定时器接收触发信号时，不得实时更改从定时器或 ADC 的时钟。

OIS6：空闲状态时 OC6 输出电平（请参见 OIS1 位）

OIS5：空闲状态时 OC5 输出电平（请参见 OIS1 位）

26.2.4 状态寄存器 3(PWMA_SR3)

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_SR3	F933H	-	-	-	-	-	-	CC6IF	CC5IF

CC6IF：比较6中断标记，参考CC1IF描述（注意：通道6只能配置为输出）

CC5IF：比较5中断标记，参考CC1IF描述（注意：通道5只能配置为输出）

26.2.5 捕获/比较使能寄存器 3（PWMA_CCER3）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCER3	F934H	-	-	CC6P	CC6E	-	-	CC5P	CC5E

CC6P：OC6 输入捕获/比较输出极性。参考 CC1P

CC6E：OC6 输入捕获/比较输出使能。参考 CC1E

CC5P：OC5 输入捕获/比较输出极性。参考 CC1P

CC5E：OC5 输入捕获/比较输出使能。参考 CC1E

26.2.6 捕获/比较模式扩展寄存器 1（PWMA_CCMR1X）

通道可用于捕获输入模式或比较输出模式，通道的方向由相应的 CCnS 位定义。该寄存器其它位的作用在输入和输出模式下不同。OCxx 描述了通道在输出模式下的功能，ICxx 描述了通道在输入模式下的功能。因此必须注意，同一个位在输出模式和输入模式下的功能是不同的。

通道配置为比较输出模式

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCMR1X	F938H	-	-	-	-	-	-	-	OC1M[3]

OC1M[3]：和 PWMA_CCMR1 中的 OC1M[2:0]合并为 4 位对输出模式进行扩展

OCnM[3:0]	模式	说明
0000	冻结	PWMn_CCR1与PWMn_CNT间的比较对OCnREF不起作用
0001	匹配时设置通道n的输出为有效电平	当PWMn_CCR1=PWMn_CNT时，OCnREF输出高
0010	匹配时设置通道n的输出为无效电平	当PWMn_CCR1=PWMn_CNT时，OCnREF输出低
0011	翻转	当PWMn_CCR1=PWMn_CNT时，翻转OCnREF
0100	强制为无效电平	强制OCnREF为低
0101	强制为有效电平	强制OCnREF为高
0110	PWM模式1	在向上计数时，当PWMn_CNT<PWMn_CCR1时OCnREF输出高，否则OCnREF输出低 在向下计数时，当PWMn_CNT>PWMn_CCR1时OCnREF输出低，否则OCnREF输出高
0111	PWM模式2	在向上计数时，当PWMn_CNT<PWMn_CCR1时OCnREF输出低，否则OCnREF输出高 在向下计数时，当PWMn_CNT>PWMn_CCR1时OCnREF输出高，否则OCnREF输出低
1000	可再触发OPM模式1	在递增计数模式下，通道为有效状态，直至（在TRGI信号上）检测到触发事件。然后，在PWM模式1下进行比较，通道会在下一次更新时再次变为有效状态。在递减计数模式下，通道为无效状态，直至（在 TRGI 信号上）检测到触发事件。然后，在PWM模式1下进行比较，通道会在下一次更新时再次变为无效状态。
1001	可再触发OPM模式2	在递增计数模式下，通道为无效状态，直至（在TRGI信号上）检测到触发事件。然后，在PWM模式2下进行比较，通道会在下一次更新时再次变为无效状态。在递减计数模式下，通道为有效状态，直至（在TRGI信号上）检测到触发事件。然后，在PWM模式2下进行比较，通道会在下一次更新时再次变为有效状态。

1010	保留	
1011	保留	
1100	组合PWM模式1	OC1REF与在PWM模式1下的行为相同。OC1REFC是OC1REF和OC2REF的逻辑或运算结果。
1101	组合PWM模式2	OC1REF与在PWM模式2下的行为相同。OC1REFC是OC1REF和OC2REF的逻辑与运算结果。
1110	不对称PWM模式1	OC1REF与在PWM模式1下的行为相同。计数器递增计数时，OC1REFC输出OC1REF；计数器递减计数时，OC1REFC输出OC2REF。
1111	不对称PWM模式2	OC1REF与在PWM模式2下的行为相同。计数器递增计数时，OC1REFC输出OC1REF；计数器递减计数时，OC1REFC输出OC2REF。

注 1：一旦 LOCK 级别设为 3（PWMn_BKR 寄存器中的 LOCK 位）并且 CCnS=00（该通道配置成输出）则该位不能被修改。

注 2：在 PWM 模式 1 或 PWM 模式 2 中，只有当比较结果改变了或在输出比较模式中从冻结模式切换到 PWM 模式时，OCnREF 电平才改变。

注 3：在有互补输出的通道上，这些位是预装载的。如果 PWMn_CR2 寄存器的 CCPC=1，OCM 位只有在 COM 事件发生时，才从预装载位取新值。

26.2.7 捕获/比较模式扩展寄存器 2（PWMA_CCMR2X）

通道配置为比较输出模式

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCMR2X	F939H	-	-	-	-	-	-	-	OC2M[3]

OC2M[3]：和 PWMA_CCMR2 中的 OC2M[2:0]合并为 4 位对输出模式进行扩展，详情参考 OC1M。

26.2.8 捕获/比较模式扩展寄存器 3（PWMA_CCMR3X）

通道配置为比较输出模式

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCMR3X	F93AH	-	-	-	-	-	-	-	OC3M[3]

OC3M[3]：和 PWMA_CCMR3 中的 OC3M[2:0]合并为 4 位对输出模式进行扩展，详情参考 OC1M。

26.2.9 捕获/比较模式扩展寄存器 4（PWMA_CCMR4X）

通道配置为比较输出模式

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCMR4X	F93BH	-	-	-	-	-	-	-	OC4M[3]

OC4M[3]：和 PWMA_CCMR4 中的 OC4M[2:0]合并为 4 位对输出模式进行扩展，详情参考 OC1M。

26.2.10 捕获/比较模式寄存器 5（PWMx_CCMR5）

通道配置为比较输出模式

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCMR5	F93CH	OC5CE	OC5M[2:0]			OC5PE	-	-	-

OC5CE：输出比较 5 清零使能。该位用于使能使用 PWMETI 引脚上的外部事件来清通道 5 的输出信号（OC5REF）

0：OC5REF 不受 ETRF 输入的影响；

1：一旦检测到 ETRF 输入高电平，OC5REF=0。

OC5M[2:0]：输出比较 5 模式，参考 OC1M。

OC5PE：输出比较 5 预装载使能，参考 OP5PE。

26.2.11 捕获/比较模式扩展寄存器 5（PWMA_CCMR5X）

通道配置为比较输出模式

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCMR5X	F93DH	-	-	-	-	-	-	-	OC5M[3]

OC5M[3]：和 PWMA_CCMR5 中的 OC5M[2:0]合并为 4 位对输出模式进行扩展，详情参考 OC1M。

26.2.12 捕获/比较模式寄存器 6（PWMx_CCMR6）

通道配置为比较输出模式

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCMR6	F93EH	OC6CE	OC6M[2:0]			OC6PE	-	-	-

OC6CE：输出比较 6 清零使能。该位用于使能使用 PWMETI 引脚上的外部事件来清通道 6 的输出信号（OC6REF）

0：OC6REF 不受 ETRF 输入的影响；

1：一旦检测到 ETRF 输入高电平，OC6REF=0。

OC6M[2:0]：输出比较 6 模式，参考 OC1M。

OC6PE：输出比较 6 预装载使能，参考 OP6PE。

26.2.13 捕获/比较模式扩展寄存器 6（PWMA_CCMR6X）

通道配置为比较输出模式

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCMR6X	F93FH	-	-	-	-	-	-	-	OC6M[3]

OC6M[3]：和 PWMA_CCMR6 中的 OC6M[2:0]合并为 4 位对输出模式进行扩展，详情参考 OC1M。

26.2.14 捕获/比较寄存器 5 高 8 位（PWMA_CCR5H）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCR5H	F940H	CCR5[15:8]							

CCR5[15:8]：捕获/比较 5 的高 8 位值

CCR5 包含了装入当前比较值（预装载值）。如果在 PWM5_CCMR1 寄存器（OC5PE 位）中未选择预装载功能，写入的数值会立即传输至当前寄存器中。否则只有当更新事件发生时，此预装载值才传输至当前捕获/比较 5 寄存器中。当前比较值同计数器 PWM5_CNT 的值相比较，并在 OC5 端口上产生输出信号。

26.2.15 捕获/比较寄存器 5 低 8 位（PWMA_CCR5L）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCR5L	F941H	CCR5[7:0]							

CCR5[7:0]：捕获/比较 5 的低 8 位值

26.2.16 捕获/比较扩展寄存器 5（PWMA_CCR5X）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCR5X	F942H	GC5C3	GC5C2	GC5C1	-	-	-	-	-

GC5C3：通道 5 和通道 3 组（Group Channel 5 and Channel 3）。通道 3 输出上失真：

0：OC5REF 对 OC3REFC 无影响；

1：OC3REFC 是 OC3REFC 和 OC5REF 的逻辑与运算结果。该位可以立即生效，也可预装载并在更新事件后执行（如果在 PWMA_CCMR3 中选择了预装载功能）。

注：也可在组合 PWM 信号上应用此失真。

GC5C2：通道 5 和通道 2 组（Group Channel 5 and Channel 2）。通道 2 输出上失真：

0：OC5REF 对 OC2REFC 无影响；

1：OC2REFC 是 OC2REFC 和 OC5REF 的逻辑与运算结果。该位可以立即生效，也可预装载并在更新事件后执行（如果在 PWMA_CCMR2 中选择了预装载功能）。

注：也可在组合 PWM 信号上应用此失真。

GC5C1：通道 5 和通道 1 组（Group Channel 5 and Channel 1）。通道 1 输出上失真：

0：OC5REF 对 OC1REFC 无影响；

1：OC1REFC 是 OC1REFC 和 OC5REF 的逻辑与运算结果。该位可以立即生效，也可预装载并在更新事件后执行（如果在 PWMA_CCMR1 中选择了预装载功能）。

注：也可在组合 PWM 信号上应用此失真。

26.2.17 捕获/比较寄存器 6 高 8 位（PWMA_CCR6H）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCR6H	F943H	CCR6[15:8]							

CCR6[15:8]：捕获/比较 6 的高 8 位值

CCR6 包含了装入当前比较值（预装载值）。如果在 PWM6_CCMR1 寄存器（OC6PE 位）中未选择预装载功能，写入的数值会立即传输至当前寄存器中。否则只有当更新事件发生时，此预装载值才传输至当前捕获/比较 6 寄存器中。当前比较值同计数器 PWM6_CNT 的值相比较，并在 OC6 端口上产生输出信号。

26.2.18 捕获/比较寄存器 6 低 8 位（PWMA_CCR6L）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCR6L	F944H	CCR6[7:0]							

CCR6[7:0]：捕获/比较 6 的低 8 位值

26.3 移相 PWM 输出模式

26.3.1 不对称 PWM 模式

在不对称模式下，生成的两个中心对齐 PWM 信号间允许存在可编程相移。频率由 PWMA_ARR 寄存器的值确定，而占空比和相移则由一对 PWMA_CCRx 寄存器确定。两个寄存器分别控制递增计数和递减计数期间的 PWM，这样每半个 PWM 周期便会调节一次 PWM：

- OC1REFC（或 OC2REFC）由 PWMA_CCR1 和 PWMA_CCR2 控制
- OC3REFC（或 OC4REFC）由 PWMA_CCR3 和 PWMA_CCR4 控制

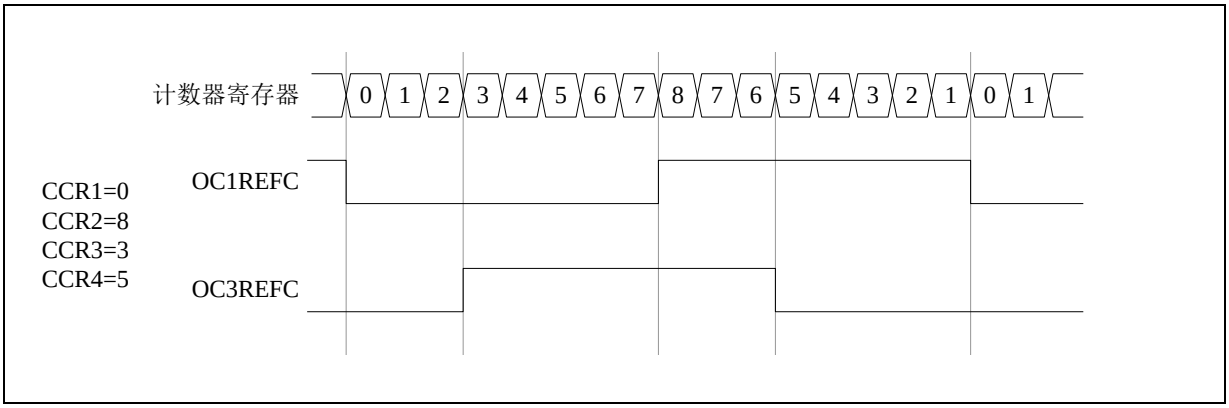
两个通道可以独立选择不对称 PWM 模式(每对 CCR 寄存器控制一个 OCx 输出)，只需向 PWMA_CCMRx 寄存器的 OCxM 位写入“1110”（不对称 PWM 模式 1）或“1111”（不对称 PWM 模式 2）。

给定通道用作不对称 PWM 通道时，也可使用其互补通道。例如，如果通道 1 上产生 OC1REFC 信号（不对称 PWM 模式 1），则由于不对称 PWM 模式 1 的原因，通道 2 上可输出 OC2REF 信号或 OC2REFC 信号。

（注：出于兼容性原因，OCxM[3:0]位域分为两部分，最高有效位与最低有效的 3 位不相邻。）

下图显示了不对称 PWM 模式下可以产生的信号示例（通道 1 到通道 4 在不对称 PWM 模式 1 下配置）。与死区发生器配合使用时，这可控制相移全桥直流到直流转换器。

（产生 2 个 50%占空比的相移 PWM 信号）



26.3.2 组合 PWM 模式

在组合 PWM 模式下，生成的两个边沿或中心对齐 PWM 信号的各个脉冲间允许存在可编程延时和相移。频率由 PWMA_ARR 寄存器的值确定，而占空比和延时则由两个 PWMA_CCRx 寄存器确定。产生的信号 OCxREFC 由两个参考 PWM 的逻辑或运算或者逻辑与运算组合组成。

- OC1REFC（或 OC2REFC）由 PWMA_CCR1 和 PWMA_CCR2 控制
- OC3REFC（或 OC4REFC）由 PWMA_CCR3 和 PWMA_CCR4 控制

两个通道可以独立选择组合 PWM 模式（每对 CCR 寄存器控制一个 OCx 输出），只需向 PWMA_CCMRx 寄存器的 OCxM 位写入“1100”（组合 PWM 模式 1）或“1101”（组合 PWM 模式 2）。

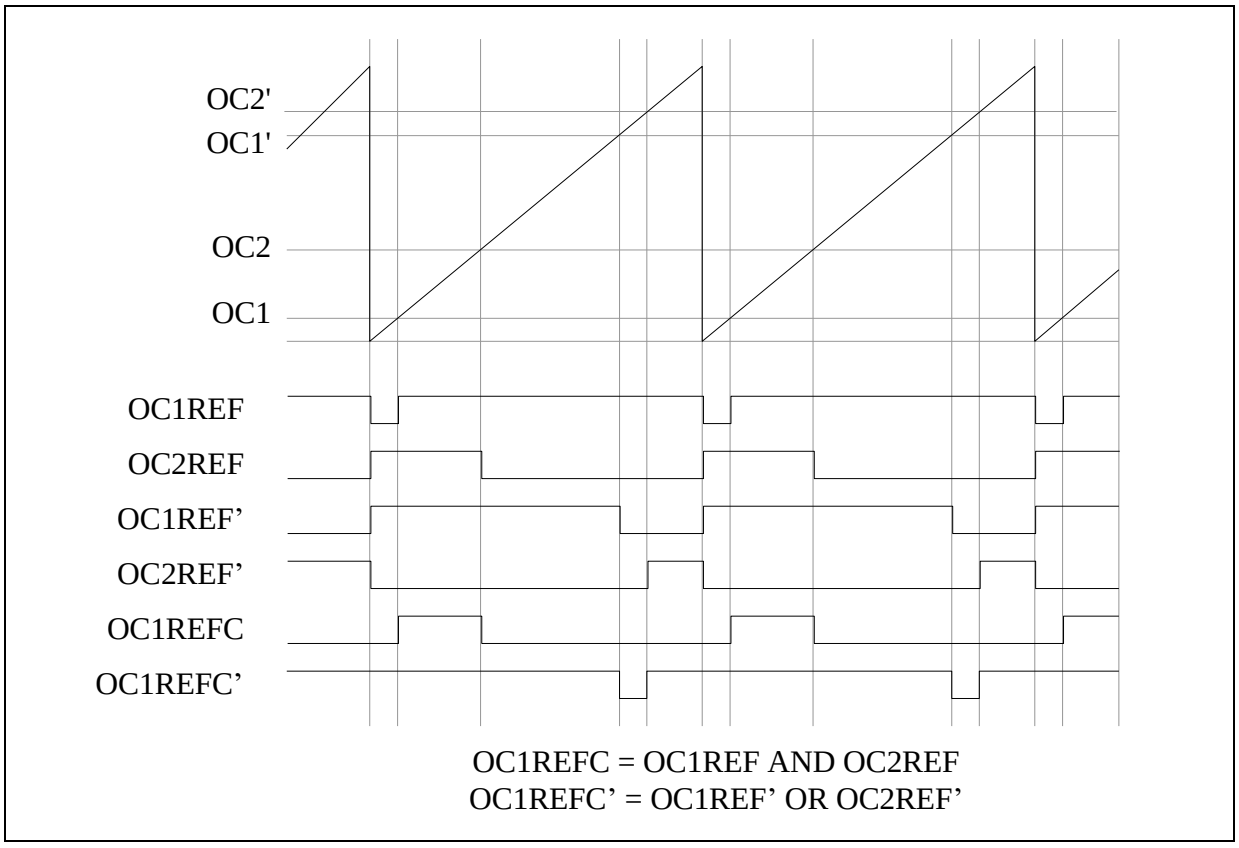
当给定通道用作组合 PWM 通道时，其互补通道必须在相反的 PWM 模式下配置。例如，一个通道在组合 PWM 模式 1 下配置，另一个通道在组合 PWM 模式 2 下配置。

（注：出于兼容性原因，OCxM[3:0]位域分为两部分，最高有效位与最低有效的 3 位不相邻。）

下图显示了不对称 PWM 模式下可以产生的信号示例，通过以下配置可获得这些信号：

- 通道 1 在组合 PWM 模式 2 下配置。
- 通道 2 在 PWM 模式 1 下配置。
- 通道 3 在组合 PWM 模式 2 下配置。
- 通道 4 在 PWM 模式 1 下配置。

（通道 1 和通道 3 上的组合 PWM 模式）



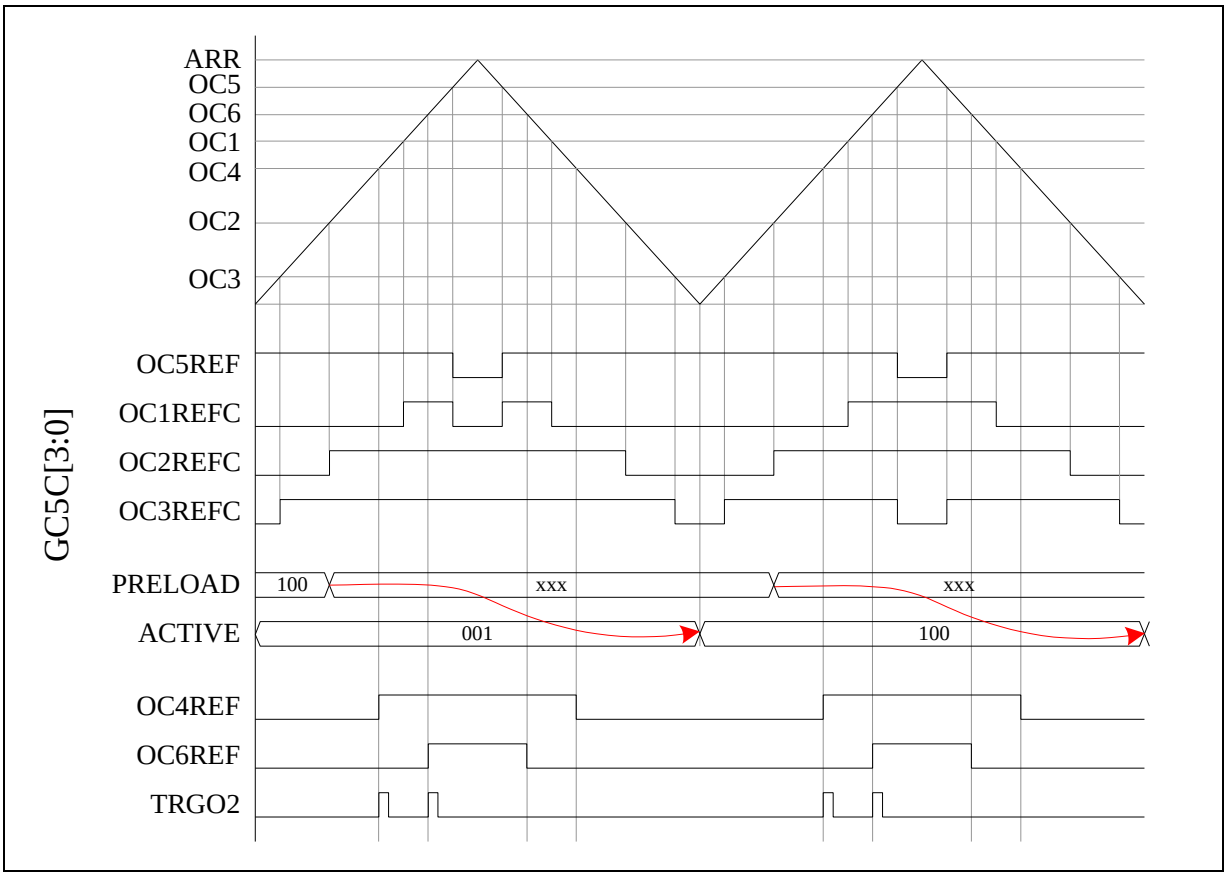
26.3.3 组合三相 PWM 模式

在组合三相 PWM 模式下，产生的一至三个中心对齐 PWM 信号与一个可编程信号间允许在脉冲中间进行逻辑与运算。OC5REF 信号用于定义产生的组合信号。凭借 PWMA_CCR5 中的 3 位 GC5C[3:1]，可以选择 OC5REF 与哪个参考信号组合。产生的信号 OCxREFC 由两个参考 PWM 的逻辑与运算组合组成。

- 如果 GC5C1 置 1，则 OC1REFC 由 PWMA_CCR1 和 PWMA_CCR5 控制
- 如果 GC5C2 置 1，则 OC2REFC 由 PWMA_CCR2 和 PWMA_CCR5 控制
- 如果 GC5C3 置 1，则 OC3REFC 由 PWMA_CCR3 和 PWMA_CCR5 控制

通道 1 到通道 3 可独立选择组合三相 PWM 模式，只需将 3 位 GC5C[3:1]中的至少一位置 1。

（三相组合 PWM 信号，每个周期多个触发脉冲）



（TRGO2 波形说明了如何根据给定的三相 PWM 信号同步 ADC）

26.4 PWM 硬件移相范例程序

请参考论坛如下的帖子

https://www.stcaimcu.com/forum.php?mod=viewthread&tid=4707&page=3#pid50750

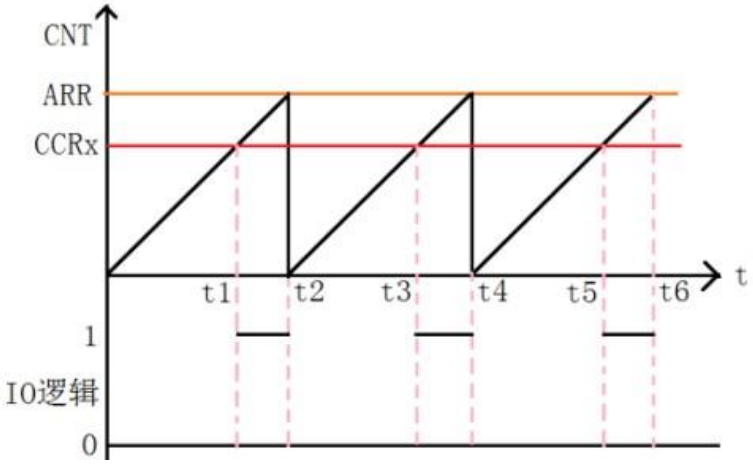
26.5 利用不对称 PWM 实现高速正交编码信号输出（热心网友冲哥提供）

前言：

众所周知，STC 早早的就出了带硬件移相的 PWM，但是好多人还不理解这个硬件移相 PWM 有什么作用，这篇帖子带大家仔细研究下这个硬件移相 PWM 的一部分——不对称 PWM。

一、普通 PWM 模式

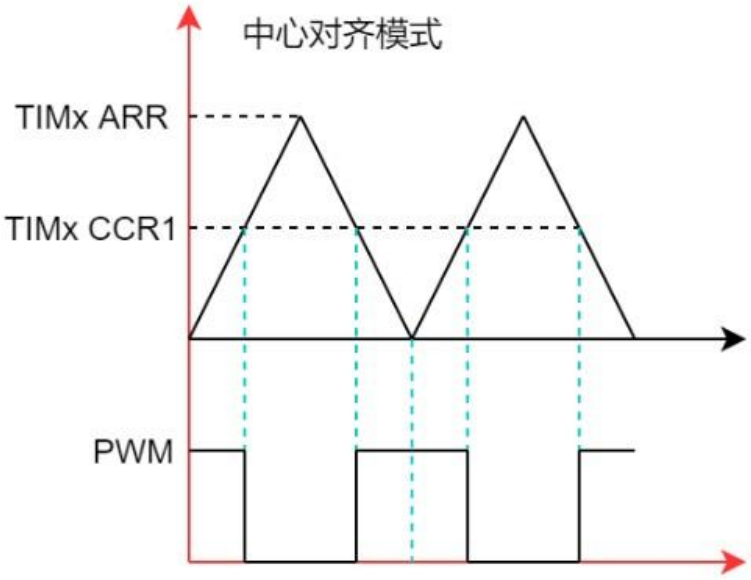
常见的 PWM 模式如下图所示，通过一个 ARR 寄存器设定最大计数值（向上计数模式），CNT 表示当前的计数，当计数超过 ARR 的时候被清 0，当 CNT 数值小于当前通道的 CCRx 的时候输出低电平，反之输出高电平，这个是常规的 PWM 模式 2（不开启极性反相的时候！）。



再来看下手册这部分，可以看到 PWM 模式 1 其实就是 PWM 模式 2 的反过来的波形

110	PWM模式1	在向上计数时，当PWMn_CNT<PWMn_CCR1时 OCnREF输出高，否则OCnREF输出低 在向下计数时，当PWMn_CNT>PWMn_CCR1时 OCnREF输出低，否则OCnREF输出高
111	PWM模式2	在向上计数时，当PWMn_CNT<PWMn_CCR1时 OCnREF输出低，否则OCnREF输出高 在向下计数时，当PWMn_CNT>PWMn_CCR1时 OCnREF输出高，否则OCnREF输出低

当然上面的都是边沿计数，还有一种中心对齐的计数



但是这种模式就是只能生成中心对称的波形，如果单纯的只想做几路不同占空比输出的波形，那确实没一点问题。可以思考下，像上面的几种模式，总的规划出来就分为四大类：

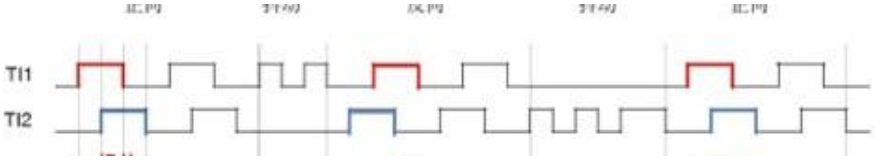
1.前面高电平后面低电平

2.前面低电平后面高电平

3.两边低电平中间高电平

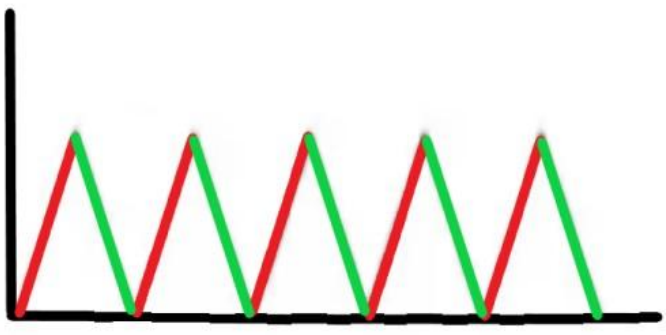
4.两边高电平中间低电平

上面 1 和 2 是同一种计数方式，3 和 4 是一种计数方式，同一时间只能选择一种计数方式。但是这时候，我们如果想要生成一个如下图的正交编码信号的 PWM 波形该如何实现的？

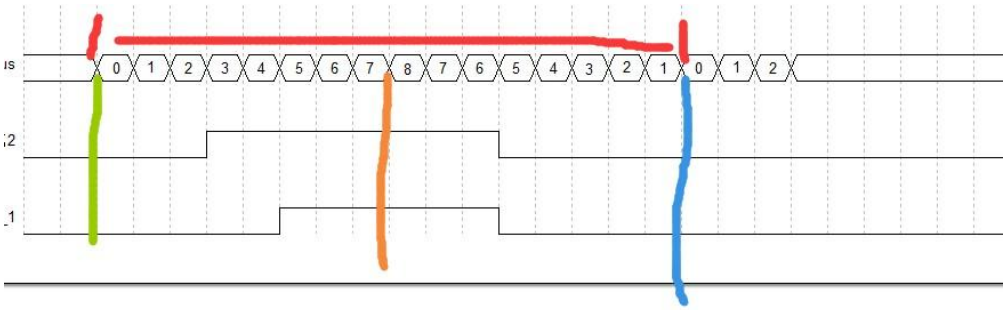


二、不对称互补 PWM

按照我个人的理解，不对称 PWM 的本质就是起始和结束电平不受约束，按照手册的说法他必须使用中心对齐计数模式，开始电平是可以在上升计数段里任意定义，结束电平是在下降计数段里任意定义的。在下面的图里，一个红色线和一个绿色线为一个中心计数模式的一个周期（从 0 计数到最大值 arr 再到 0 的过程），开始电平可以在红线里的任意位置，结束电平可以是绿线的任意位置，这样是不是就很好分配了。



再画个简单的示意图：



假设红色的是整个计数周期，将他对半的分为绿-橙区域和橙-蓝区域，开始的电平可以在这个绿-橙区域任意位置，结束电平也可以定义在这个橙-蓝区域的任意位置，此时你想要这个波形横移多少位置不就是分分钟的了。

当然在开始之前我们还需要了解一些基本知识：

24.3.1　不对称 PWM 模式

在不对称模式下，生成的两个中心对齐 PWM 信号间允许存在可编程相移。频率由 PWMA_ARR 寄存器的值确定，而占空比和相移则由一对 PWMA_CCRx 寄存器确定。两个寄存器分别控制递增计数和递减计数期间的 PWM，这样每半个 PWM 周期便会调节一次 PWM：

- OC1REFC（或 OC2REFC）由 PWMA_CCR1 和 PWMA_CCR2 控制
- OC3REFC（或 OC4REFC）由 PWMA_CCR3 和 PWMA_CCR4 控制

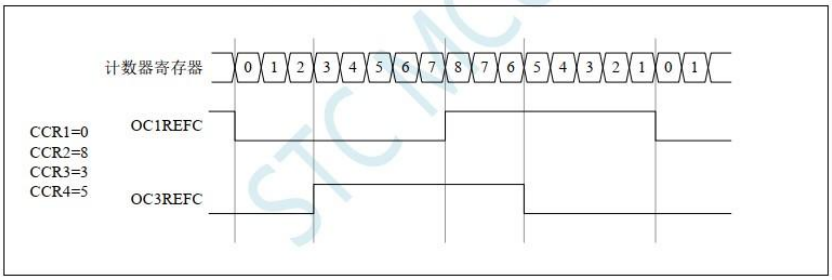
两个通道可以独立选择不对称 PWM 模式(每对 CCR 寄存器控制一个 OCx 输出)，只需向 PWMA_CCMRx 寄存器的 OCxM 位写入“1110”（不对称 PWM 模式 1）或“1111”（不对称 PWM 模式 2）。

给定通道用作不对称 PWM 通道时，也可使用其互补通道。例如，如果通道 1 上产生 OC1REFC 信号（不对称 PWM 模式 1），则由于不对称 PWM 模式 1 的原因，通道 2 上可输出 OC2REF 信号或 OC2REFC 信号。

（注：出于兼容性原因，OCxM[3:0]位域分为两部分，最高有效位与最低有效的 3 位不相邻。）

下图显示了不对称 PWM 模式下可以产生的信号示例（通道 1 到通道 4 在不对称 PWM 模式 1 下配置）。与死区发生器配合使用时，这可控制相移全桥直流到直流转换器。

（产生 2 个 50%占空比的相移 PWM 信号）



手册上关于这个模式的描述就这么一页，我给他归纳总结了一下：

1.需要在中心对齐计数模式下使用

2.必须手动设置为不对称模式才能使用这个输出（**切记这个寄存器的位置不连续，要重点注意**）

3.两个通道才能组合成为一个完整的不对称输出通道，且通道 1（又名 OC1REFC 或 OC2REFC）由 CCR1 和 CCR2 控制，通道 2（又名 OC3REFC 或 OC4REFC）由 CCR3 和 CCR4 控制。

4.1P 端口设置为不对称输出时起始电平时间点为 CCR1,结束时间点为 CCR2;　2P 端口设置为不对称输出时起始电平时间点为 CCR2,结束时间点为 CCR1;

5.别的配置都可以参考普通 PWM 的配置

6.一个通道会占用两个端口的 CCR 寄存器，其中一个作为不对称输出的时候，另一个还可以作为普通的 pwm 模式输出。

三、寄存器描述

1.设置为中心对齐计数

23.9.4 控制寄存器 1（PWMx_CR1）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CR1	FEC0H	ARPEA	CMSA[1:0]		DIRA	OPMA	URSA	UDISA	CENA
PWMB_CR1	FEE0H	ARPEB	CMSB[1:0]		DIRB	OPMB	URSB	UDISB	CENB

ARPEn：自动预装载允许位（n=A,B）

0：PWMn_ARR 寄存器没有缓冲，它可以被直接写入

1：PWMn_ARR 寄存器由预装载缓冲器缓冲

CMSn[1:0]：选择对齐模式（n=A,B）

CMSn[1:0]	对齐模式	说明
00	边沿对齐模式	计数器依据方向位（DIR）向上或向下计数
01	中央对齐模式1	计数器交替地向上和向下计数。 配置为输出的通道的输出比较中断标志位(CCnIF)只在计数器向下计数时被置1。
10	中央对齐模式2	计数器交替地向上和向下计数。 配置为输出的通道的输出比较中断标志位(CCnIF)只在计数器向上计数时被置1。
11	中央对齐模式3	计数器交替地向上和向下计数。 配置为输出的通道的输出比较中断标志位(CCnIF)在计数器向上和向下计数时均被置1。

注 1：在计数器开启时（CEN=1），不允许从边沿对齐模式转换到中央对齐模式。

2.手动使能不对称模式

		和OC2REF的逻辑与运算结果。
1110	不对称PWM模式1	OC1REF与在PWM模式1下的行为相同。计数器递增计数时，OC1REFC输出OC1REF；计数器递减计数时，OC1REFC输出OC2REF。
1111	不对称PWM模式2	OC1REF与在PWM模式2下的行为相同。计数器递增计数时，OC1REFC输出OC1REF；计数器递减计数时，OC1REFC输出OC2REF。

注 1：一旦 LOCK 级别设为 3（PWMn_BKR 寄存器中的 LOCK 位）并且 CCnS=00（该通道配置成

这里尤其要注意 OCxM[bit2:0]在 CCMx 寄存器，但是最高位 OCxM[bit3]在 CCMxX 寄存器（小写的 x 表示哪个端口）；

3.通道比较数值设置

23.9.25 捕获/比较寄存器 1/5 高 8 位（PWMx_CCR1H）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWMA_CCR1H	FED5H	CCR1[15:8]							
PWMB_CCR5H	FEF5H	CCR5[15:8]							

CCRn[15:8]：捕获/比较 n 的高 8 位值（n=1,5）

若 CCn 通道配置为输出：CCRn 包含了装入当前比较值（预装载值）。如果在 PWMn_CCMR1 寄存器（OCnPE 位）中未选择预装载功能，写入的数值会立即传输至当前寄存器中。否则只有当更新事件发生时，此预装载值才传输至当前捕获/比较 n 寄存器中。当前比较值同计数器 PWMn_CNT 的值相比较，并在 OCn 端口上产生输出信号。

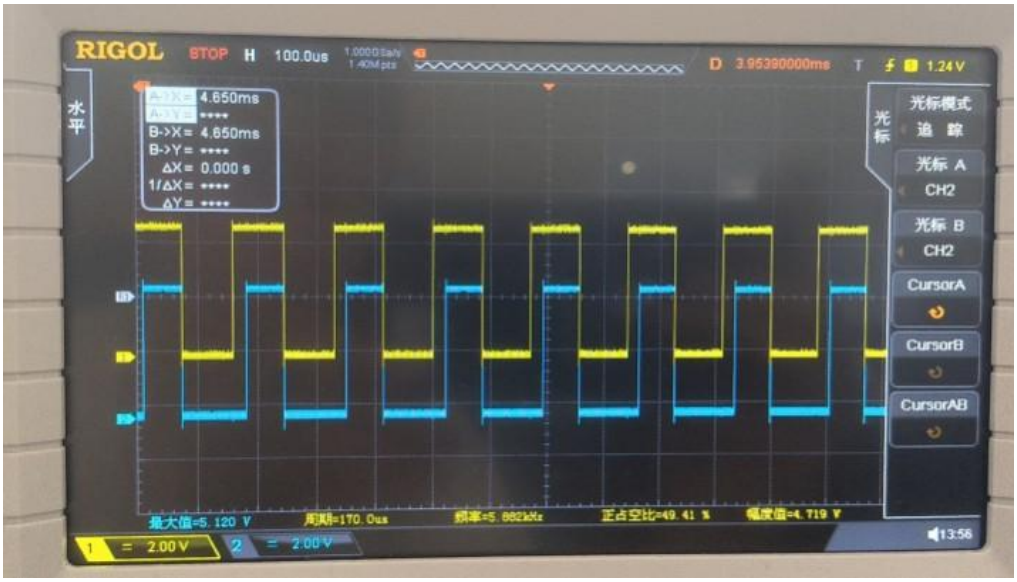
若 CCn 通道配置为输入：CCRn 包含了上一次输入捕获事件发生时的计数器值（此时该寄存器为只读）。

四、代码实战

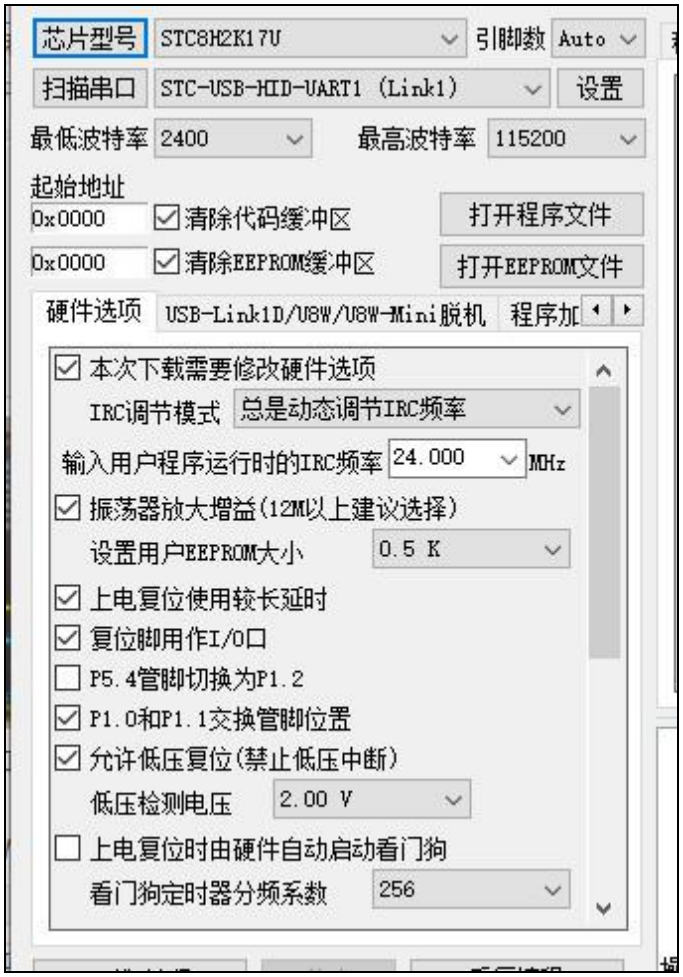
首先我们可以看下来验证一下第二章里的第四点 4.1P 端口设置为不对称输出时起始电平时间点为 CCR1, 结束时间点为 CCR2; 2P 端口设置为不对称输出时起始电平时间点为 CCR2,结束时间点为 CCR1）是什么意思：

1）首先配置 1P 端口为不对称 PWM 模式 2 输出，2P 端口 PWM 模式 2 输出

```
PWM1_Duty = 3;
PWM2_Duty = 5;
PWMA_PSCRH = 0;
PWMA_PSCRL = 255;
PWMA_CCER1 = 0x00; //写 CCMRx 前必须先清零 CCxE 关闭通道
PWMA_CCER2 = 0x00;
PWMA_CCMR1X = 0x01;//端口 1:不对称 PWM2 模式
PWMA_CCMR1 = 0x70;
PWMA_CCMR2X = 0x00;//端口 2:PWM2 模式
PWMA_CCMR2 = 0x70;
PWMA_CCER1 = 0x55; //配置通道输出使能和极性
PWMA_CCER2 = 0x55;
PWMA_ARRH = (u8)(PWM_PERIOD >> 8); //设置周期时间
PWMA_ARRL = (u8)PWM_PERIOD;
PWMA_ENO = 0x00;
PWMA_ENO |= ENO1P; //使能输出
PWMA_ENO |= ENO2P; //使能输出
PWMA_PS = 0x00; //高级 PWM 通道输出脚选择位
PWMA_PS |= PWM1_0; //选择 PWM1_0 通道
PWMA_PS |= PWM2_0; //选择 PWM2_0 通道
```

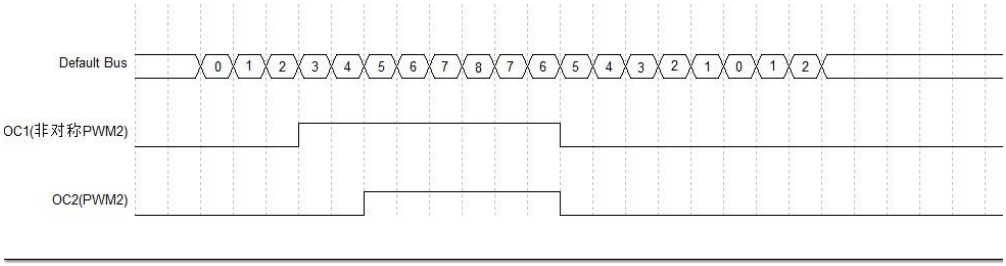


效果图如上，黄色线条为 1P 端口，即不对称 PWM2 模式。蓝色线条为 2P 端口，即 PWM2 模式。下载选项和使用的硬件如下图所示



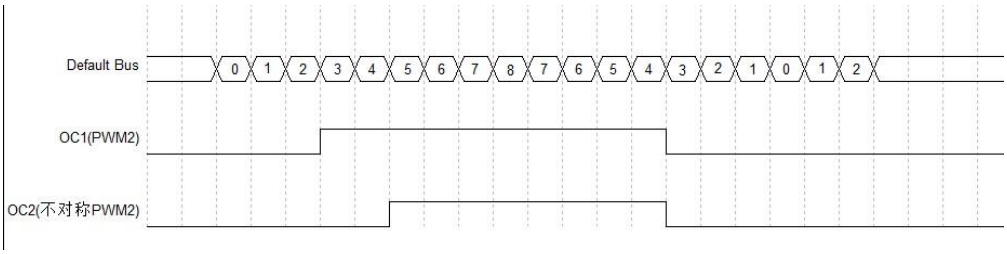
1.首先 PWM 频率 = 时钟/(分频+1)/ARR/2 = 24M / 256/8/2 ≈ 5.86Khz，和示波器测的频率基本一致

2.按照我们的分析，实际上端口 1 应该在上升计数为 3 的时候变成高电平，下降计数为 5 的时候变为低电平，占空比为 50%。端口 2 应该在上升计数为 5 的时候变成高电平，下降计数为 5 的时候变为低电平，例如下图所示，可以看到最终测量到的结果和这个一致，可见这个分析是正确的。

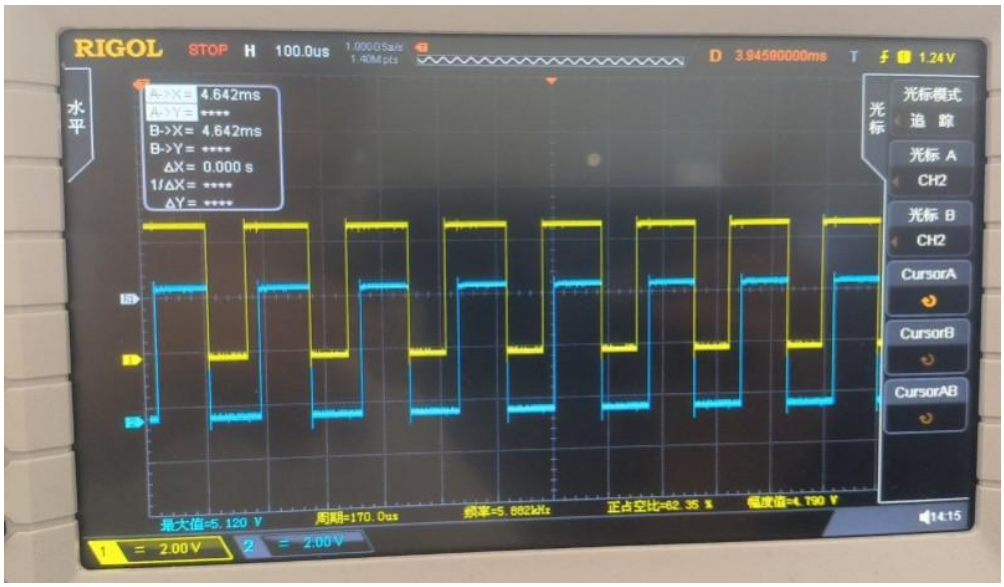


2）首先配置 1P 端口为不对称 PWM 模式 2 输出，2P 端口 PWM 模式 2 输出

```
PWM1_Duty = 3;
PWM2_Duty = 5;
PWMA_PSCRH = 0;
PWMA_PSCRL = 255;
PWMA_CCER1 = 0x00; //写 CCMRx 前必须先清零 CCxE 关闭通道
PWMA_CCER2 = 0x00;
PWMA_CCMR1X = 0x00;//端口 1:称 PWM2 模式
PWMA_CCMR1 = 0x70;
PWMA_CCMR2X = 0x01;//端口 2:不对 PWM2 模式
PWMA_CCMR2 = 0x70;
PWMA_CCER1 = 0x55; //配置通道输出使能和极性
PWMA_CCER2 = 0x55;
PWMA_ARRH = (u8)(PWM_PERIOD >> 8); //设置周期时间
PWMA_ARRL = (u8)PWM_PERIOD;
PWMA_ENO = 0x00;
PWMA_ENO |= ENO1P; //使能输出
PWMA_ENO |= ENO2P; //使能输出
PWMA_PS = 0x00;  //高级 PWM 通道输出脚选择位
PWMA_PS |= PWM1_0; //选择 PWM1_0 通道
PWMA_PS |= PWM2_0; //选择 PWM2_0 通道
```



我们下载进去实测一下：



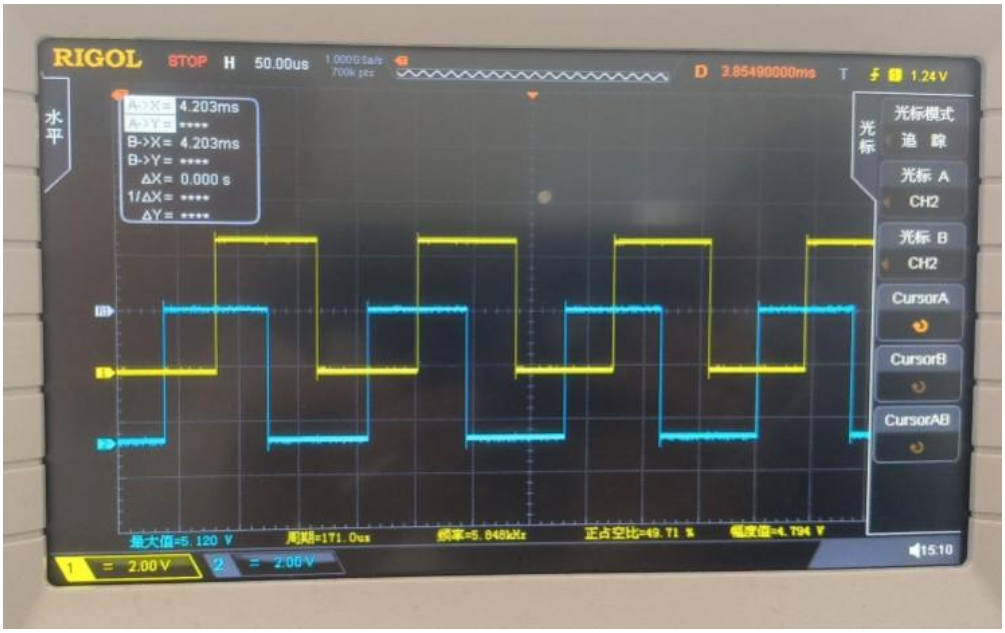
可以看到这个波形就是我们预期的波形

3）最后我们来实现一下正交编码信号的波形。
通道 1 使用 PWM2 不对称模式，通道 3 使用不对称 PWM2 模式

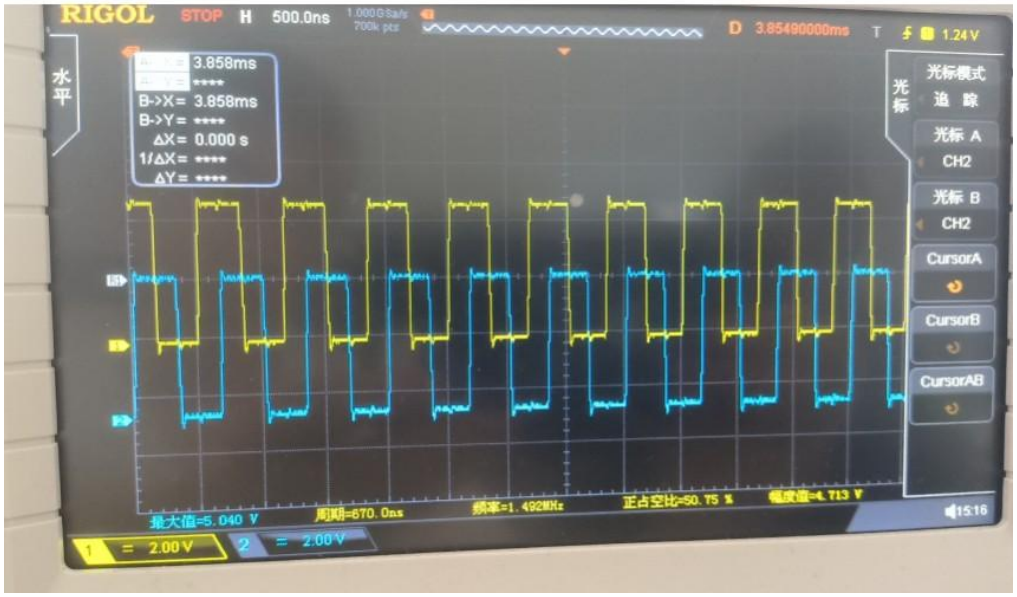
```
PWM1_Duty = 0;
PWM2_Duty = 8;
PWM3_Duty = 4;
PWM4_Duty = 4;
PWMA_PSCRH = 0;
PWMA_PSCRL = 255;
PWMA_CCER1 = 0x00; //写 CCMRx 前必须先清零 CCxE 关闭通道
PWMA_CCER2 = 0x00;
PWMA_CCMR1X = 0x01;//通道 1:PWM2 不对称模式
PWMA_CCMR1 = 0x70;
PWMA_CCMR2X = 0x00;//通道 2:PWM2 模式,极性反转
PWMA_CCMR2 = 0x70;
PWMA_CCMR3X = 0x01;//通道 3:不对称 PWM2 模式
PWMA_CCMR3 = 0x70;
PWMA_CCMR4X = 0x00;//通道 4:PWM2 模式
PWMA_CCMR4 = 0x70;
PWMA_CCER1 = 0x55; //配置通道输出使能和极性
PWMA_CCER2 = 0x55;
PWMA_ARRH = (u8)(PWM_PERIOD >> 8); //设置周期时间
PWMA_ARRL = (u8)PWM_PERIOD;
PWMA_ENO = 0x00;
PWMA_ENO |= ENO1P; //使能输出
PWMA_ENO |= ENO3P; //使能输出
PWMA_PS = 0x00;  //高级 PWM 通道输出脚选择位
PWMA_PS |= PWM1_0; //选择 PWM1_0 通道
PWMA_PS |= PWM2_0; //选择 PWM2_0 通道
```

```
PWMA_PS |= PWM3_0; //选择 PWM3_0 通道
PWMA_PS |= PWM4_0; //选择 PWM4_0 通道
```

最后下载到单片机看下最终的波形效果。



之所以这里要用硬件，就是频率可以很快，且不需要进中断即可实现。（代码里为了方便测量将 PSC 分频寄存器配置了 255，实际上这个改为 0，即不分频速度分分钟能到 Mhz 级别，效果如下）



27 高速高级 PWM（HSPWM）

产品线	高速高级 PWM
STC8H1K08 系列	
STC8H1K28 系列	
STC8H3K64S4 系列	
STC8H3K64S2 系列	
STC8H8K64U 系列	
STC8H4K64TL 系列	
STC8H4K64TLCD 系列 A 系列	
STC8H4K64TLCD 系列 A+系列	
STC8H1K08T 系列	●
STC8H2K12U-A/B 系列	●
STC8H2K32U 系列	●
STC8G1K08-SOP8 系列	
STC8G1K08A-SOP8 系列	

STC8H 的部分系列单片机为高级 PWMA 和高级 PWMB 提供了高速模式（HSPWMA 和 HSPWMB）。高速高级 PWM 是以高级 PWMA 和高级 PWMB 为基础，增加了高速模式。

当系统运行在较低工作频率时，高速高级 PWM 可工作在高达 144M~192M 的频率下。从而可以达到降低内核功耗，提升外设性能的目的

27.1 相关寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
HSPWMA_CFG	高速 PWMA 配置寄存器	FBF0H	-	-	-	-	-	INTEN	ASYNCEN	1	xxxx,0001
HSPWMA_ADR	高速 PWMA 地址寄存器	FBF1H	RW/BUSY	ADDR[6:0]							0000,0000
HSPWMA_DAT	高速 PWMA 数据寄存器	FBF2H	DATA[7:0]								0000,0000
HSPWMB_CFG	高速 PWMB 配置寄存器	FBF4H	-	-	-	-	-	INTEN	ASYNCEN	1	xxxx,0001
HSPWMB_ADR	高速 PWMB 地址寄存器	FBF5H	RW/BUSY	ADDR[6:0]							0000,0000
HSPWMB_DAT	高速 PWMB 数据寄存器	FBF6H	DATA[7:0]								0000,0000

27.1.1 HSPWM 配置寄存器（HSPWMn_CFG）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
HSPWMA_CFG	FBF0H	-	-	-	EXTN	-	INTEN	ASYNCEN	1
HSPWMB_CFG	FBF4H	-	-	-	-	-	INTEN	ASYNCEN	1

EXTN：异步控制模式访问寄存器区选择

0：使用异步控制模式访问 XFR 区域为 FEC0H~FEDFH 的高级 PWM 相关寄存器。

1：使用异步控制模式访问 XFR 区域为 F930H~F947H 的高级 PWM 硬件移相相关寄存器。

ASYNCEN：异步控制模式使能位

0：关闭异步控制。

1：使能异步控制模式。

注：当关闭异步控制时，高级 PWMA/高级 PWMB 为传统模式，此时高级 PWM 会自动选择系统工作频率，PWM 工作频率与系统工作频率相同；若需要时 PWM 工作在高速模式，则需要使能异步控制模式，此时 PWM 时钟可选择主时钟（MCLK）或者 PLL 输出时钟

INTEN：异步模式中断使能位

0：关闭异步模式下的 PWM 中断。

1：使能异步模式下的 PWM 中断。

注：异步模式下，若需要响应高级 PWMA/高级 PWMB 的中断，则必须使能 INTEN 位

27.1.2 HSPWM 地址寄存器（HSPWMn_ADR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
HSPWMA_ADR	FBF1H	RW/BUSY	ADDR[6:0]						
HSPWMB_ADR	FBF5H	RW/BUSY	ADDR[6:0]						

ADDR[6:0]：高级 PWMA/PWMB 的特殊功能寄存器地址低 7 位

RW/BUSY：读写控制位、状态位

写 0：异步方式**写** PWMA/PWMB 的特殊功能寄存器。

写 1：异步方式**读** PWMA/PWMB 的特殊功能寄存器。

读 0：异步读写已经完成

读 1：异步读写正在进行，处于忙状态

27.1.3 HSPWM 数据寄存器（HSPWMn_DAT）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
HSPWMA_DAT	FBF2H	DATA[7:0]							
HSPWMB_DAT	FBF6H	DATA[7:0]							

DATA[7:0]：高级 PWMA/PWMB 的特殊功能寄存器数据

写：写数据到高级 PWMA/PWMB 的特殊功能寄存器。

读：从高级 PWMA/PWMB 的特殊功能寄存器读取数据。

异步读取 PWMA 的特殊功能寄存器步骤：（PWMB 类似）

1、读取 HSPWMA_ADR，等待 BUSY 为 0，确定前一个异步读写已完成

2、将 PWMA 的特殊功能寄存器的低 7 位写入 HSPWMA_ADR，同时置“1”HSPWMA_ADR.7

3、读取 HSPWMA_ADR，等待 BUSY 为 0

4、读取 HSPWMA_DAT

异步写 PWMA 的特殊功能寄存器步骤：（PWMB 类似）

1、读取 HSPWMA_ADR，等待 BUSY 为 0，确定前一个异步读写已完成

2、将需要写入 PWMA 的特殊功能寄存器的数据写入 HSPWMA_DAT

3、将 PWMA 的特殊功能寄存器的低 7 位写入 HSPWMA_ADR，同时清“0”HSPWMA_ADR.7

4、读取 HSPWMA_ADR，等待 BUSY 为 0。（可跳过此步继续执行其他代码，以提高系统效率）

特别注意：特殊功能寄存器 PWMA_PS 和 PWMB_PS 属于端口控制寄存器，不属于 PWMA 和 PWMB 寄存器组，所以无论是否启动 PWM 的异步控制模式，PWMA_PS 和 PWMB_PS 寄存器都只能使用普通同步模式进行读写

27.2 范例程序

27.2.1 使能高级 PWM 的高速模式（异步模式）

```
//测试工作频率为24MHz

#include "stc8h.h"
#include "intrins.h"

#define    FOSC              24000000UL

#define    HSCK_MCLK         0
#define    HSCK_PLL          1
#define    HSCK_SEL          HSCK_PLL

#define    PLL_96M           0
#define    PLL_144M          1
#define    PLL_SEL           PLL_144M

#define    CKMS              0x80
#define    HSIOCK            0x40
#define    MCK2SEL_MSK       0x0c
#define    MCK2SEL_SEL1      0x00
#define    MCK2SEL_PLLD2     0x04
#define    MCK2SEL_PLLD4     0x08
#define    MCKSEL_MSK        0x03
#define    MCKSEL_HIRC       0x00
#define    MCKSEL_XOSC       0x01
#define    MCKSEL_X32K       0x02
#define    MCKSEL_IRC32K     0x03

#define    ENCKM             0x80
#define    PCKI_MSK          0x60
#define    PCKI_D1           0x00
#define    PCKI_D2           0x20
#define    PCKI_D3           0x40
#define    PCKI_D4           0x60

void delay()
{
    int    i;

    for    (i=0; i<100; i++);
}

char ReadPWMA(char addr)
{
    char dat;

    while (HSPWMA_ADR & 0x80);                  //等待前一个异步读写完成
    HSPWMA_ADR = addr | 0x80;                   //设置间接访问地址,只需要设置原 XFR 地址的低 7 位
                                                //HSPWMA_ADR 寄存器的最高位写 1,表示读数据
    while (HSPWMA_ADR & 0x80);                  //等待当前异步读取完成
    dat    = HSPWMA_DAT;                        //读取异步数据
```

```
    return dat;
}

void WritePWMA(char addr, char dat)
{
    while (HSPWMA_ADR & 0x80);              //等待前一个异步读写完成
    HSPWMA_DAT = dat;                       //准备需要写入的数据
    HSPWMA_ADR = addr & 0x7f;               //设置间接访问地址,只需要设置原 XFR 地址的低 7 位
                                            //HSPWMA_ADR 寄存器的最高位写 0,表示写数据
}

void main()
{
    P_SW2 |= 0x80;                          //使能访问 XFR，没有冲突不用关闭

    //选择 PLL 输出时钟
#if  (PLL_SEL == PLL_96M)
    CLKSEL &= ~CKMS;                        //选择 PLL 的 96M 作为 PLL 的输出时钟
#elif (PLL_SEL == PLL_144M)
    CLKSEL |= CKMS;                         //选择 PLL 的 144M 作为 PLL 的输出时钟
#else
    CLKSEL &= ~CKMS;                        //默认选择 PLL 的 96M 作为 PLL 的输出时钟
#endif

    //选择 PLL 输入时钟分频,保证输入时钟为 12M
    PLLCR &= ~PCKI_MSK;
#if  (FOSC == 12000000UL)
    PLLCR |= PCKI_D1;                       //PLL 输入时钟 1 分频
#elif (FOSC == 24000000UL)
    PLLCR |= PCKI_D2;                       //PLL 输入时钟 2 分频
#elif (FOSC == 36000000UL)
    PLLCR |= PCKI_D3;                       //PLL 输入时钟 3 分频
#elif (FOSC == 48000000UL)
    PLLCR |= PCKI_D4;                       //PLL 输入时钟 4 分频
#else
    PLLCR |= PCKI_D1;                       //默认 PLL 输入时钟 1 分频
#endif

    //启动 PLL
    PLLCR |= ENCKM;                         //使能 PLL 倍频

    delay();                                //等待 PLL 锁频

    //选择 HSPWM/HSSPI 时钟
#if  (HSCK_SEL == HSCK_MCLK)
    CLKSEL &= ~HSIOCK;                      //HSPWM/HSSPI 选择主时钟为时钟源
#elif (HSCK_SEL == HSCK_PLL)
    CLKSEL |= HSIOCK;                       //HSPWM/HSSPI 选择 PLL 输出时钟为时钟源
#else
    CLKSEL &= ~HSIOCK;                      //默认 HSPWM/HSSPI 选择主时钟为时钟源
#endif

    HSCLKDIV = 0;                           //HSPWM/HSSPI 时钟源不分频

    HSPWMA_CFG = 0x03;                      //使能 PWMA 相关寄存器异步访问功能

    //通过异步方式设置 PWMA 的相关寄存器
    WritePWMA((char)&PWMA_CCER1, 0x00);
    WritePWMA((char)&PWMA_CCMR1, 0x00);     //CC1 为输出模式
```

```
    WritePWMA((char)&PWMA_CCMR1, 0x60);             //OC1REF 输出 PWM1(CNT<CCR 时输出有效电平 1)
    WritePWMA((char)&PWMA_CCER1, 0x05);             //使能 CC1/CC1N 上的输出功能
    WritePWMA((char)&PWMA_ENO, 0x03);               //使能 PWM 信号输出到端口
    WritePWMA((char)&PWMA_BKR, 0x80);               //使能主输出
    WritePWMA((char)&PWMA_CCR1H, 200 >> 8);         //设置输出 PWM 的占空比
    WritePWMA((char)&PWMA_CCR1L, 200);
    WritePWMA((char)&PWMA_ARRH, 1000 >> 8);         //设置输出 PWM 的周期
    WritePWMA((char)&PWMA_ARRL, 1000);
    WritePWMA((char)&PWMA_DTR, 10);                 //设置互补对称输出 PWM 的死区
    WritePWMA((char)&PWMA_CR1, 0x01);               //开始 PWM 计数

    P2M0 = 0;
    P2M1 = 0;
    P3M0 = 0;
    P3M1 = 0;

    P2 = ReadPWMA((char)&PWMA_ARRH);                //异步方式读取寄存器
    P3 = ReadPWMA((char)&PWMA_ARRL);

    while (1);
}
```

27.2.2 使用异步模式控制高级 PWM 移相功能

```
//测试工作频率为 24MHz

#include "STC8H.h"
#include "stdio.h"

typedef     unsigned char   u8;
typedef     unsigned int    u16;
typedef     unsigned long   u32;

/*****************************  用户定义宏  ***********************************/

#define     MAIN_Fosc       24000000L               //定义主时钟

/*****************************************************************************/
/*
P1.0 和 P1.1 输出 50%占空比的互补波形
P1.3 和 P1.4 生成移相 180 的波形。
*/

/*************  本地变量声明  **************/
u16   pscr=0 ;                                      //分频系数
u16   arr=287;                                      //需为奇数
u16   ccr=72;                                       //通道比较数值


/*************  函数声明  **************/
void PWM_Config(void);


/**********************  主函数  *************************/
void main(void)
{
```

```
    P_SW2 = 0x80;                                   //扩展寄存器访问使能

    P0M1 = 0x00;    P0M0 = 0x00;                    //设置为准双向口
    P1M1 = 0x00;    P1M0 = 0x00;                    //设置为准双向口
    P2M1 = 0x00;    P2M0 = 0x00;                    //设置为准双向口
    P3M1 = 0x00;    P3M0 = 0x00;                    //设置为准双向口
    P4M1 = 0x00;    P4M0 = 0x00;                    //设置为准双向口
    P5M1 = 0x00;    P5M0 = 0x00;                    //设置为准双向口
    P6M1 = 0x00;    P6M0 = 0x00;                    //设置为准双向口
    P7M1 = 0x00;    P7M0 = 0x00;                    //设置为准双向口
    PWM_Config();
    EA = 1;                                         //打开总中断

    while (1)
    {
    }
}

void delay(void)
{
    int    i;
    for(i=0;i<100;i++);
}

void WritePWMA( char addr,char dat )
{
    while(HSPWMA_ADR & 0X80);
    HSPWMA_DAT = dat;
    HSPWMA_ADR = addr &  0x7f;
}

//========================================================================
// 函数: void PWM_Config(void)
// 描述: PWM 功能配置.
// 参数: none.
// 返回: none.
// 版本: V1.0
//========================================================================
void PWM_Config(void)
{
#if  0
    PWMA_CCER1 = 0x00;                              //写 CCMRx 前必须先清零 CCxE  关闭通道
    PWMA_CCER2 = 0x00;

    //组合模式测试-1
//  PWMA_CCMR1X = 0x00;                             //通道 1:组合模式 2
//  PWMA_CCMR1 = 0x60;
//  PWMA_CCMR2X = 0x00;                             //通道 2:PWM 模式 2
//  PWMA_CCMR2 = 0x60;
    PWMA_CCMR3X = 0x01;                             //通道 3:组合模式 2
    PWMA_CCMR3 = 0x50;
    PWMA_CCMR4X = 0x00;                             //通道 4:PWM 模式 1
    PWMA_CCMR4 = 0x60;

//  PWMA_CCER1 = 0x55;                              //配置通道输出使能和极性
    PWMA_CCER2 = 0x55;

    PWMA_ARRH = (u8)(arr >> 8);                     //设置周期时间
    PWMA_ARRL = (u8)arr;
```

```
        PWMA_ENO = 0x50;

        PWMA_PS = 0x00;                                 //高级 PWM 通道输出脚选择位

        PWMA_CCR1H = (u8)(((arr+1)/2) >> 8);
        PWMA_CCR1L = (u8)(((arr+1)/2));

        PWMA_CCR2H = (u8)(ccr >> 8);
        PWMA_CCR2L = (u8)(ccr);

        PWMA_CCR3H = (u8)(((arr+1)/2) >> 8);
        PWMA_CCR3L = (u8)(((arr+1)/2));

        PWMA_CCR4H = (u8)(((arr+1)/2 + ccr) >> 8);
        PWMA_CCR4L = (u8)(((arr+1)/2 + ccr ));

        PWMA_BKR = 0x80;                                //使能主输出
        PWMA_CR1 |= 0x01;                               //开始计时
#else

        CLKSEL |= 0X80;                                 //选择 144MHz 作为主时钟
        PLLCR |= 0X20;
        PLLCR |= 0X80;                                  //启动 PLL
        delay();

        CLKSEL |= 0X40;                                 //选择 PLL
        HSCLKDIV = 0;                                   //不分频
        HSPWMA_CFG = 0X03;

//      PWMA_PSCR =pscr;
//      WritePWMA((char)&PWMA_PSCRH,pscr>>8);           //写分频系数
//      WritePWMA((char)&PWMA_PSCRL,pscr);

        WritePWMA((char)&PWMA_CCER1,0x00);              //写 CCMRx 前必须先清零 CCxE 关闭通道
        WritePWMA((char)&PWMA_CCER2,0x00);

//      WritePWMA((char)&PWMA_CCMR1X,0x10);             //写 端口 1 写入 PWM 模式 1
//      WritePWMA((char)&PWMA_CCMR1,0x50);
//
//      WritePWMA((char)&PWMA_CCMR2X,0x00);             //写 端口 2 写入 PWM 模式 1
//      WritePWMA((char)&PWMA_CCMR2,0x70);

        HSPWMA_CFG = 0X13;                              //新增的移相寄存器使用间接定址访问前
                                                        //需要设置新增的使能位(bit4)
        WritePWMA((char)&PWMA_CCMR3X,0x01);             //写 端口 3 写入组合 PWM 模式 2
        HSPWMA_CFG = 0X03;                              //间接寻址旧的寄存器则要清除这个使能位(bit4)
        WritePWMA((char)&PWMA_CCMR3,0x50);
//      WritePWMA((char)&PWMA_CCMR3X,0x00);             //写端口 3 写入 PWM 模式 2
//      WritePWMA((char)&PWMA_CCMR3,0x70);

//      WritePWMA((char)&PWMA_CCMR4X,0x00);             //写端口 3 写入 PWM 模式 1
        WritePWMA((char)&PWMA_CCMR4,0x60);              //

//      WritePWMA((char)&PWMA_CCER1,0x55);              //写使能
        WritePWMA((char)&PWMA_CCER2,0x55);

        WritePWMA((char)&PWMA_ARRH,(u8)(arr >> 8));     //写计数周期
        WritePWMA((char)&PWMA_ARRL,(u8)arr);
```

```
    WritePWMA((char)&PWMA_ENO,(u8)0x50);                //写端口输出使能


    WritePWMA((char)&PWMA_CCR1H,(u8)(((arr+1)/2) >> 8));
    WritePWMA((char)&PWMA_CCR1L,(u8)(((arr+1)/2)));

    WritePWMA((char)&PWMA_CCR2H,(u8)(ccr >> 8));
    WritePWMA((char)&PWMA_CCR2L,(u8)(ccr));

    WritePWMA((char)&PWMA_CCR3H,(u8)(((arr+1)/2) >> 8));
    WritePWMA((char)&PWMA_CCR3L,(u8)(((arr+1)/2)));

    WritePWMA((char)&PWMA_CCR4H,(u8)(((arr+1)/2 + ccr) >> 8));
    WritePWMA((char)&PWMA_CCR4L,(u8)(((arr+1)/2 + ccr )));

    WritePWMA((char)&PWMA_BKR,(u8)0x80);                //写
    WritePWMA((char)&PWMA_CR1,(u8)0x01);                //写启动

#endif
}
```