Application Note

SPC11X8/SPD11X8 PWM 使用指南

2021 年 11 月 – 版本 2

目录

表格列表

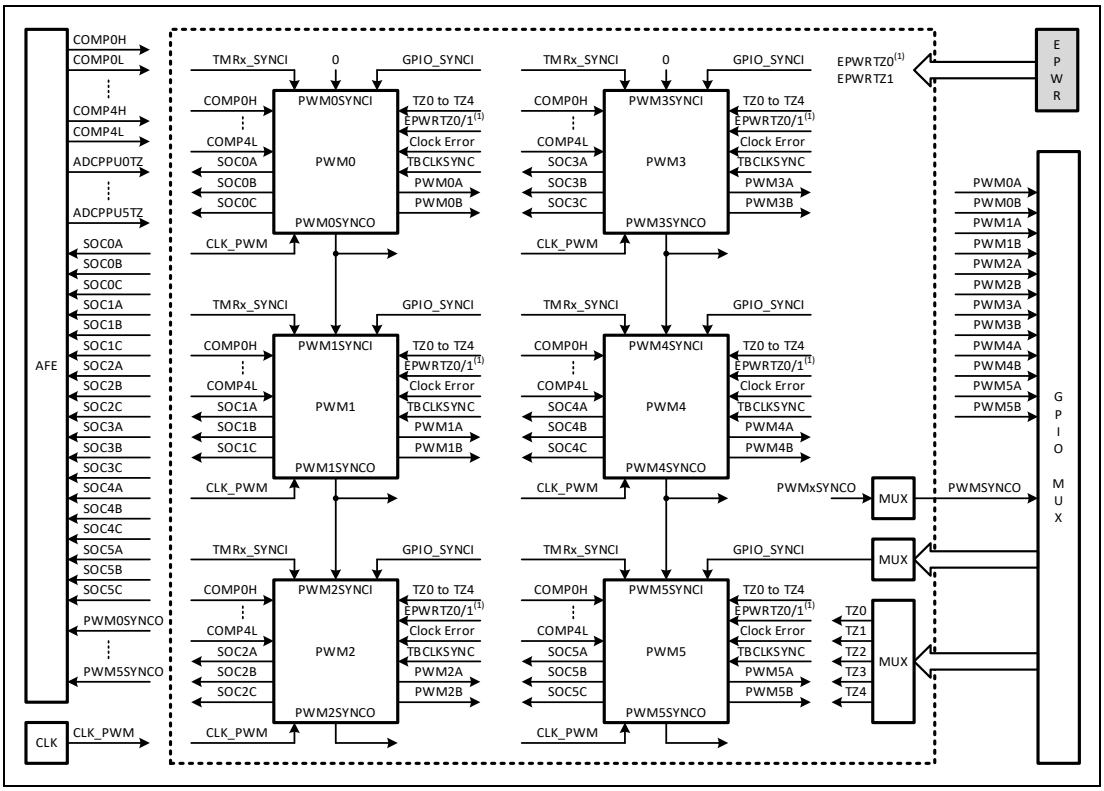
图片列表

1 PWM 单元

1.1 概述

PWM 在电力电子系统中有着至关重要的作用，广泛地应用在电机控制、开关电源及 UPS 中。如图 1-1 所示，SPC11X8/SPD11X8 中带有 6 个独立的 PWM 单元，每个 PWM 单元有两路输出。同时各个 PWM 单元可以连接在一起，产生同步的 PWM 组。

图 1-1: PWM 总框图



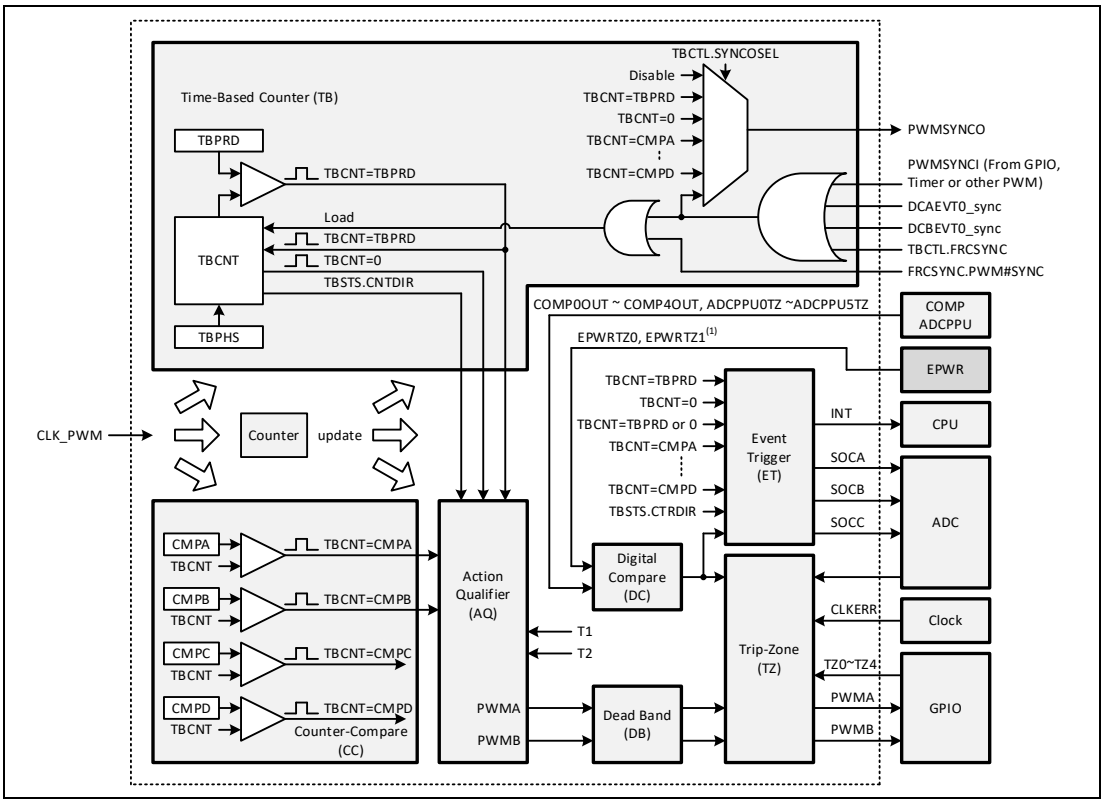
（1） 在 SPC11X8/SPD11X8 中，EPWR 模块处于未连接状态，EPWRTZ0 和 EPWRTZ1 始终为 0。

1.2 PWM 单元特性

- 六组 PWM 单元，每组提供两个互补 PWM 输出，亦可编程为双独立输出，提供共 12 组 PWM 输出管脚；
- PWM 的 Clock 频率最高可达 200MHz，可提供高精度 PWM 控制；
- 功能强大的 PWM 触发 ADC 功能，所有的 PWM 事件都可以触发 CPU 中断或者触发 ADC 转换（SOC）；
- 可用软件强制 PWM 输出为特定电平；
- 支持 Cycle-by-cycle 关断 PWM，支持 One-shot 连续关断 PWM；
- 灵活独立的上升沿和下降沿死区控制，可实现双通道非交叠互补波形或者固定相移波形；
- 内部比较器 Comparator 可关断任意 PWM，只需简单配置；
- 跨单元的软硬件波形同步和配置同步。

1.3 PWM 单元关键信号

图 1-2: PWM 单元功能框图



（1） 在 SPC11X8/SPD11X8 中，EPWR 模块处于未连接状态，EPWRTZ0 和 EPWRTZ1 始终为 0。

图 1-2 给出了 PWM 单元内部功能框图。其中关键的信号包括：

- 同步信号输入 PWMSYNCI 和同步信号输出 PWMSYNCO
  同步信号用于设定不同 PWM 单元之间的相位关系。SPC11X8/SPD11X8 中，PWMSYNCI 输入信号可以配置为来自于 GPIO 输入、Timer 输出的同步信号、前级 PWM 单元的 PWMSYNCO 输出。

- Trip-zone 封锁信号
  封锁信号用于在异常事件发生时，立即或者安全地停止 PWM。在 SPC11X8/SPD11X8 中，每个 PWM 有 5 个来自 GPIO 的封锁信号（TZ0 到 TZ4）用于接收外部触发事件。同时，CLKDET 时钟监测模块输出的时钟异常事件、PLL 失锁事件以及 CPU 的 Lockup 和 Halted 事件也可以独立配置为每个 PWM 的封锁信号。

- 比较器（COMP）和 ADC 后处理（ADCPPU）输出
  SPC11X8/SPD11X8 中提供了 10 个比较器，用于过载检测。同时，提供了 6 个 ADC 硬件后处理单元实现过高、过低和过零检测。PWM 可以将这些信号用作封锁，并配置内部的数字比较单元，在已知的可能出现尖峰封锁信号的时间窗内对原始信号设置盲区，实现尖峰信号的消隐。

- ADC 触发转换信号 PWMxSOCA、PWMxSOCB 和 PWMxSOCC

- PWM 输出信号 PWMxA 和 PWMxB

2 PWM 主要原理

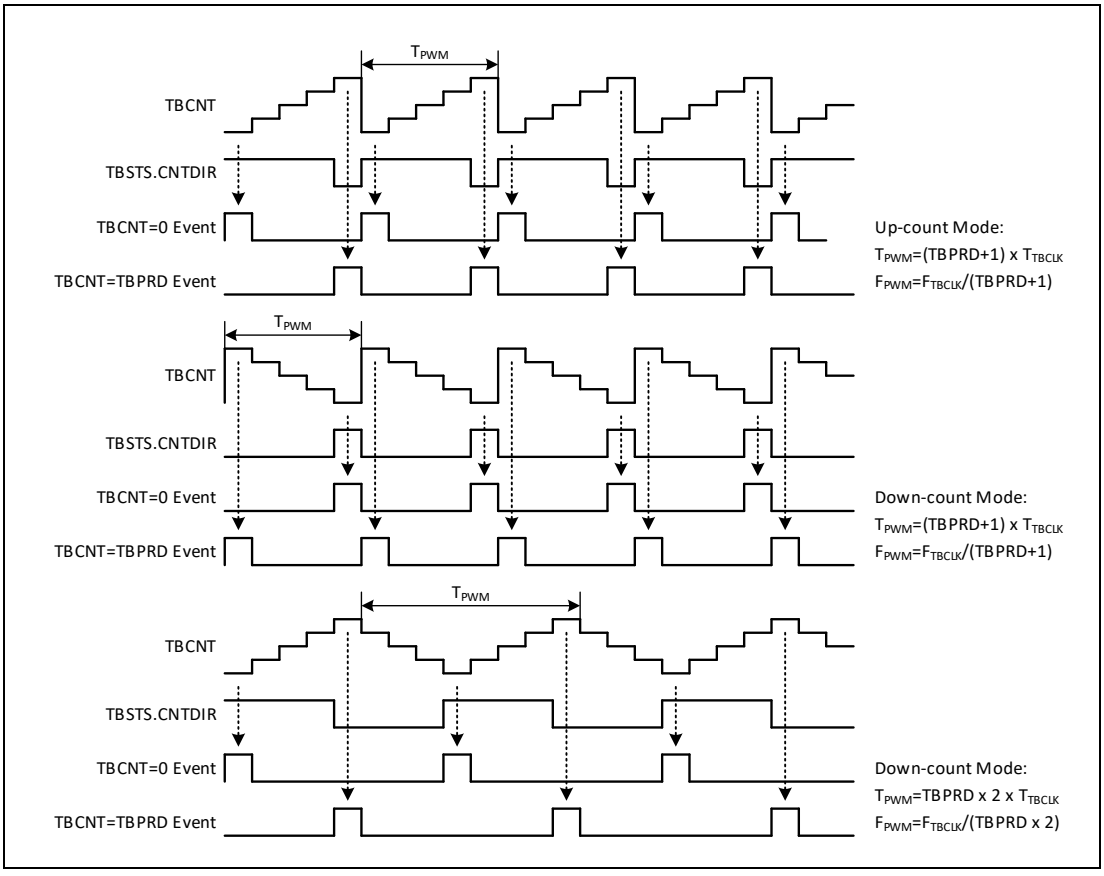
2.1 计数模式及事件

如图 2-1 所示，PWM 单元内部的计数器 TBCNT 可以按照以下三种方式进行计数：

- 向上计数
- 向下计数
- 先向上再向下

计数过程中，TBCNT 的最小值为 0，最大值是周期寄存器 TBPRD 中保存的值。TBSTS.CNTDIR 指示了计数方向。

图 2-1: PWM 三种模式计数波形



计数比较（Counter Compare）模块将 TBCNT 当前计数值与 0，TBPRD，CMPx 寄存器的值相比较，产生相应的事件。结合其他模块的输出或者外部输入，共有如下事件被送至行为限定（Action Qualifier）模块。

- TBCNT=TBPRD
- TBCNT=0
- TBCNT=CMPA
- TBCNT=CMPB
- T0/T1（可配置为数字比较事件、外部封锁信号或者输入同步信号）
- 软件强制控制

表 2-1 到表 2-3 列举了不同计数方式和场景下各个事件的优先级。值得注意的是，为了便于通过调整 CMP 的值就可以实现从 0%到 100%的波形占空比，向上计数模式时的 CAU/CBU 事件和向下计数模式时的 CAD/CBD 事件，不受图 2-1 中所示的 TBSTS.CNTDIR 在 TBCNT=TBPRD 和 TBCNT=0 时的变化的影响。

- 向上计数模式
  当 TBCNT 计数到 TBPRD，会产生 PRD 事件。此时，即便 TBSTS.CNTDIR=0（指示即将向下计数至 0），只要 CMPA/CMPB 的值被配置为和 TBPRD 相等，就可以同时产生 CAU/CBU 事件，但优先级低于 PRD。
- 向下计数模式
  当 TBCNT 计数到 0，会产生 ZRO 事件。此时，即便 TBSTS.CNTDIR=1（指示即将向上计数至 TBPRD），只要 CMPA/CMPB 的值被配置为 0，就可以同时产生 CAD/CBD 事件，但优先级低于 ZRO。

表 2-1: 向上计数时各事件优先级

优先级	事件
1（最高）	软件强制控制
2	TBCNT=TBPRD (PRD)
3	T0 on up-count (T0U)
4	T1 on up-count (T1U)
5	TBCNT=CMPB on up-count (CBU)
6	TBCNT=CMPA on up-count (CAU)
7（最低）	TBCNT=0 (ZRO)

表 2-2: 向下计数时各事件优先级

优先级	事件
1（最高）	软件强制控制
2	TBCNT=0 (ZRO)
3	T0 on down-count (T0D)
4	T1 on down-count (T1D)
5	TBCNT=CMPB on down-count (CBD)
6	TBCNT=CMPA on down-count (CAD)
7（最低）	TBCNT=TBPRD (PRD)

表 2-3: 上下计数时各事件优先级

优先级	事件	
	从 0 向上计数到 TBPRD-1 时	从 TBPRD 向下计数到 1 时
1（最高）	软件强制控制	软件强制控制
2	T0 on up-count (T0U)	T0 on down-count (T0D)
3	T1 on up-count (T1U)	T1 on down-count (T1D)
4	TBCNT=CMPB on up-count (CBU)	TBCNT=CMPB on down-count (CBD)
5	TBCNT=CMPA on up-count (CAU)	TBCNT=CMPA on down-count (CAD)
6（最低）	TBCNT=0 (ZRO)	TBCNT=TBPRD (PRD)

行为限定（Action Qualifier）模块可以配置每个事件发生时 PWMxA/PWMxB 的行为：

- 不操作（不受事件影响）
- 输出置低（Clear）
- 输出置高（Set）
- 输出翻转（Toggle）

图 2-2 示范了上下计数模式时如何产生对称 PWM 波形输出：当向上计数至 TBCNT=CMPA/CMPB 时，触发 CAU/CBU 事件，将 PWMA/PWMB 置高；当向下计数至 TBCNT=CMPA/CMPB 时，触发 CAD/CBD 事件，将 PWMA/PWMB 置低。

波形的占空比变化如下：

- 当 CMPA/CMPB=0，则 TBCNT=0 时，由于 TBSTS.CNTDIR=1（指示向上计数），故只能产生 CAU/CBU 事件，永远无法产生 CAD/CBD 事件，导致 PWMA/PWMB 的输出为常高，对应于 100%的占空比。
- 随着 CMPA/CMPB 增大，占空比逐渐减小。
- 当 CMPA/CMPB=TBPRD，则 TBCNT=TBPRD 时，由于 TBSTS.CNTDIR=0（指示向下计数），故只能产生 CAD/CBD 事件，永远无法产生 CAU/CBU 事件，导致 PWMA/PWMB 的输出为常低，对应于 0%的占空比。

图 2-2: 上下计数模式时的对称 PWM 输出

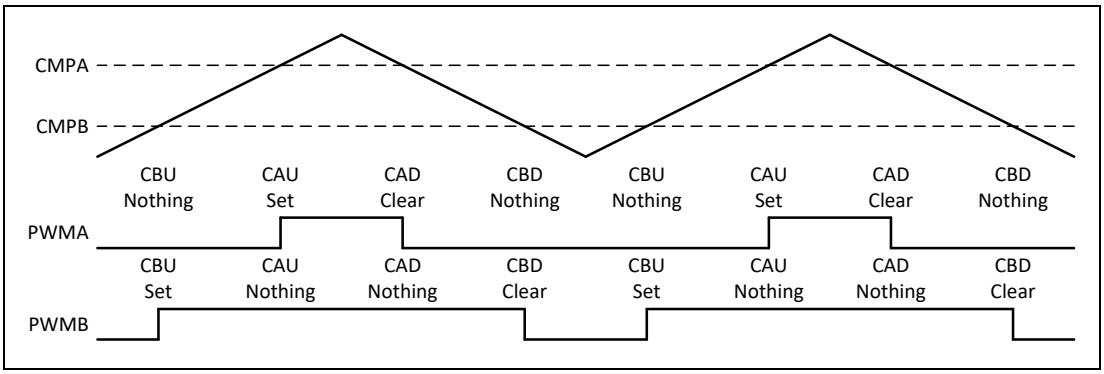


图 2-3 示范了上下计数模式时如何产生非对称 PWM 波形输出。

对于 PWMA 的输出，当向上计数至 TBCNT=CMPA 时，触发 CAU 事件，将 PWMA 置高；当向下计数至 TBCNT=CMPB 时，触发 CBD 事件，将 PWMA 置低。

对于 PWMB 的输出，当向上计数至 TBCNT=CMPB 时，触发 CBU 事件，将 PWMB 置高；当向下计数至 TBCNT=CMPA 时，触发 CAD 事件，将 PWMB 置低。

图 2-3: 上下计数模式时的非对称 PWM 输出

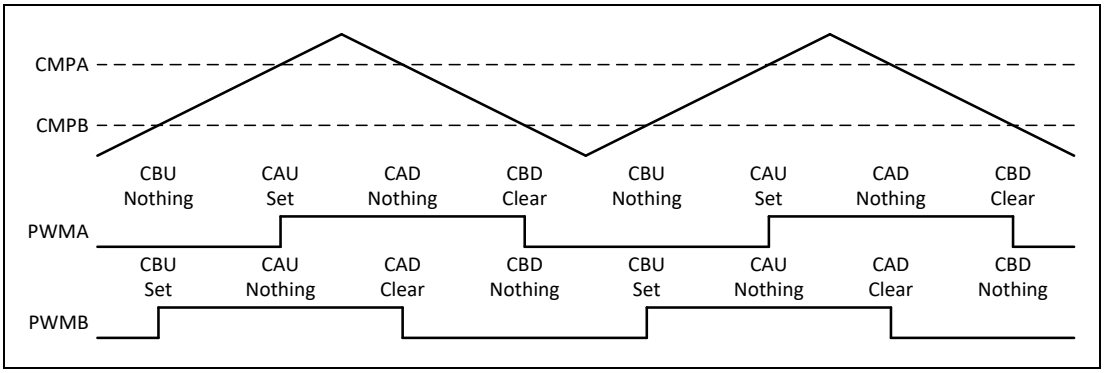


图 2-4 示范了向上计数模式时如何产生占空比连续可调的 PWM 波形：当 TBCNT=0 时，触发 ZRO 事件，将 PWMA 置高；当 TBCNT=CMPA 时，触发 CAU 事件，将 PWMA 置低。

随着 CMPA 从 0 到 TBPRD+1，波形的占空比可以实现从 0%到 100%的变化，具体如下：

- 当 CMPA=0，则 TBCNT=0 时，由于产生的 CAU 事件比 ZRO 事件优先级高，导致输出常低，对应于 0%的占空比。
- 当 0<CMPA≤TBPRD，则 TBCNT=0 时，产生 ZRO 事件，将输出置高；TBCNT=CMPA 时，产生 CAU 事件，将输出置低。占空比随着 CMPA 的增加而增加
- 当 CMPA>TBPRD，则仅能产生 ZRO 事件，无法产生 CAU 事件，导致输出常高，对应于 100%的占空比。

图 2-4: 向上计数模式时的 PWM 输出

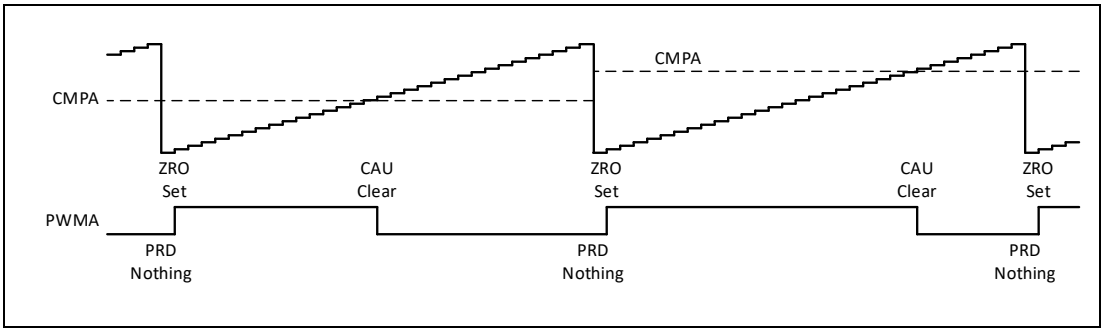
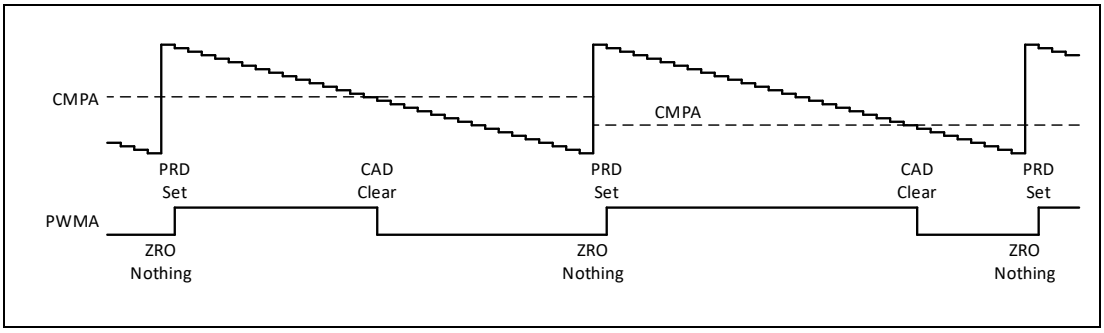


图 2-5 示范了向下计数模式时如何产生占空比连续可调的 PWM 波形：当 TBCNT=TBPRD 时，触发 PRD 事件，将 PWMA 置高；当 TBCNT=CMPA 时，触发 CAD 事件，将 PWMA 置低。

随着 CMPA 从 TBPRD 到 0，再到 TBPRD+1（可等效视作-1 以便理解），波形的占空比可以实现从 0 到 100%的变化，具体如下：

- 当 CMPA=TBPRD，则 TBCNT=TBPRD 时，由于产生的 CAD 事件比 PRD 事件优先级高，导致输出常低，对应于 0%的占空比。
- 当 TBPRD>CMPA≥0，则 TBCNT=TBPRD 时，产生 PRD 事件，将输出置高；TBCNT=CMPA 时，产生 CAD 事件，将输出置低。占空比随着 CMPA 的减小而增加。
- 当 CMPA>TBPRD，则仅能产生 PRD 事件，无法产生 CAD 事件，导致输出常高，对应于 100%的占空比。

图 2-5: 向下计数模式时的 PWM 输出

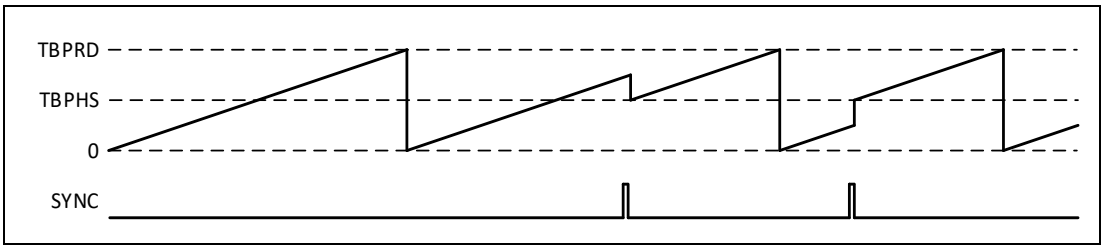


2.2 计数同步

在实际应用中，通常需要多个 PWM 协同控制一个电机，这就要求所用到的 PWM 模块间信号需要同步。SPC11X8/SPD11X8 的 PWM 单元在接收到同步信号时，会把 TBPHS 中的数值载入到计数器（TBCNT）中，并根据计数模式和相关配置来决定同步后的计数方向。

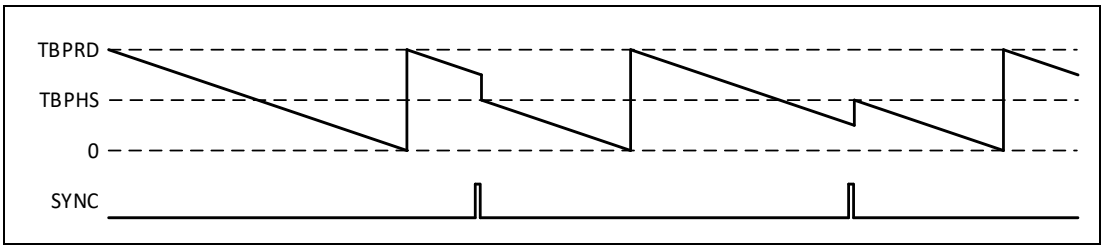
- 向上计数模式（图 2-6）
  PWM 在同步后从 TBPHS 开始继续往上计数。若 TBPHS=TBPRD，则下一拍计数为 0。

图 2-6: 向上计数模式时的 PWM 同步



- 向下计数模式（图 2-7）
  PWM 在同步后从 TBPHS 开始继续往下计数。若 TBPHS=0，则下一拍计数为 TBPRD。

图 2-7: 向下计数模式时的 PWM 同步



- 上下计数模式

  PWM 在同步后回到 TBPHS，若 TBCTL.PHSDIR=1（图 2-8）或 TBPHS=0，则下一拍向上计数；若 TBCTL.PHSDIR=0（图 2-9）或者 TBPHS=TBPRD，则下一拍向下计数。

图 2-8:　上下计数模式时的 PWM 同步（TBCTL.PHSDIR=1）

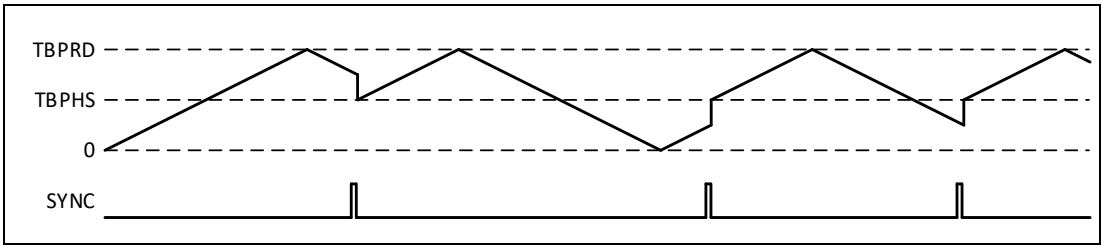
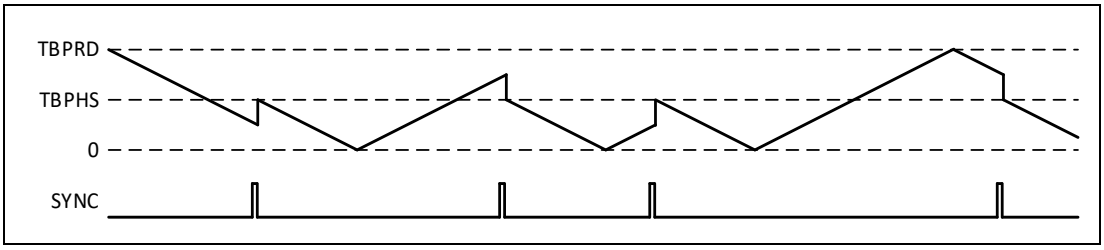


图 2-9:　上下计数模式时的 PWM 同步（TBCTL.PHSDIR=0）



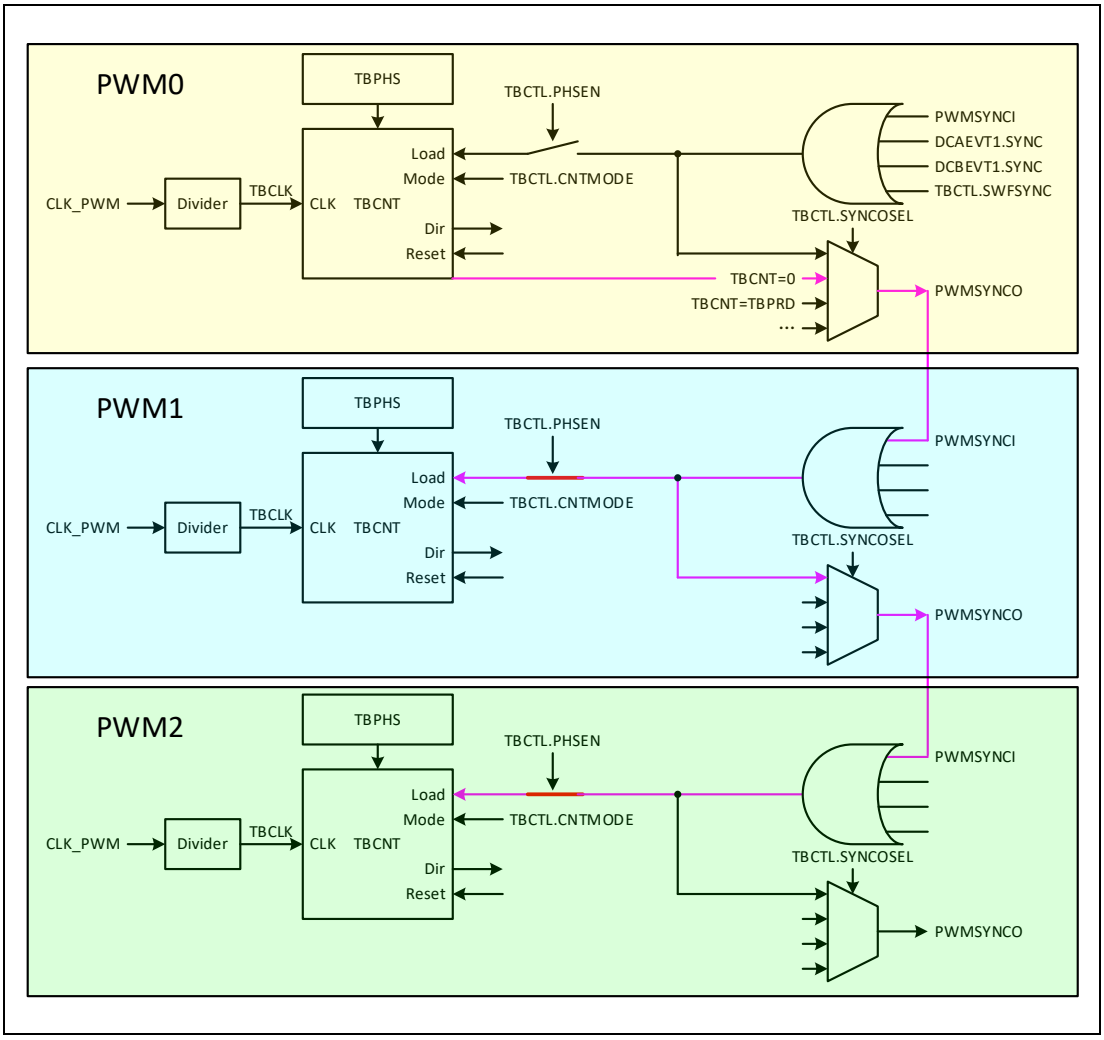
PWM 同步信号的来源包括以下几种：

- 来自 GPIO 管脚的全局硬件同步
- 来自 Timer0、Timer1 或 Timer2 的全局硬件同步
- 来自全局寄存器的全局软件同步
- 来自前级 PWM 的级间硬件同步
- 来自本级 DCAEVT1 事件的本地硬件同步
- 来自本级 DCBEVT1 事件的本地硬件同步
- 来自本级寄存器的本地软件同步

图 2-10 演示了如何通过级间同步信号流，实现从 PWM0 到 PWM2 的同步。在实际应用中还有如下几点注意事项：

- 需要同步的几个 PWM 配置为同样的计数时钟分频比，确保 TBCLK 同频；
- 调用 PWM_ForceClockSync()同步各个 PWM 的计数时钟；
- 同步信号传递到计数器会有一个 PWM 时钟（CLK_PWM）的延时。因此，如果 PWM 时钟（CLK_PWM）到计数时钟（TBCLK）的分频比为 1，并且前级 PWMSYNCO 被配置为诸如 TBCNT=0、TBCNT=TBPRD、TBCNT=CMPA、TBCNT=CMPB、TBCNT=CMPC 或 TBCNT=CMPD 这几个和计数值有关的选项时，后级的 TBPHS 寄存器在设置时需要考虑一拍计数偏差。

图 2-10: 多个 PWM 同步信号流



2.3 死区控制（Deadband）

在实际应用中，通常要求用于控制半桥、全桥电路的上下桥臂的 PWM 互补波形需要有一定的非交叠死区时间。SPC11X8/SPD11X8 的 PWM 通过如图 2-11 所示的死区控制电路来实现这一点。

- 死区控制总体上分为上升沿延时和下降沿延时两个通路。开关 S1 和 S2 分别控制了两个通路的输入，可选项是行为限定（Action Qualifier）模块的 PWMA 输出或 PWMB 输出。
- DBRED 和 DBFED 两个寄存器控制了两个通路的延时量。
- 开关 S3 和 S4 控制了两个通路的输出极性。
- 开关 S5 和 S6 控制了两个通路的使能，即输出可以配置为延时后的信号，或者是延时前通路输入的原始信号。
- 开关 S7 和 S8 控制了死区控制电路最后输出的 PWMA 和 PWMB 的来源，可以配置为上升沿延时通路输出或者下降沿延时通路输出。
- 开关 S9 可以把经过上升沿延时后的信号作为下降沿延时通路的输入，从而实现双边沿延时。

图 2-11: 死区控制电路

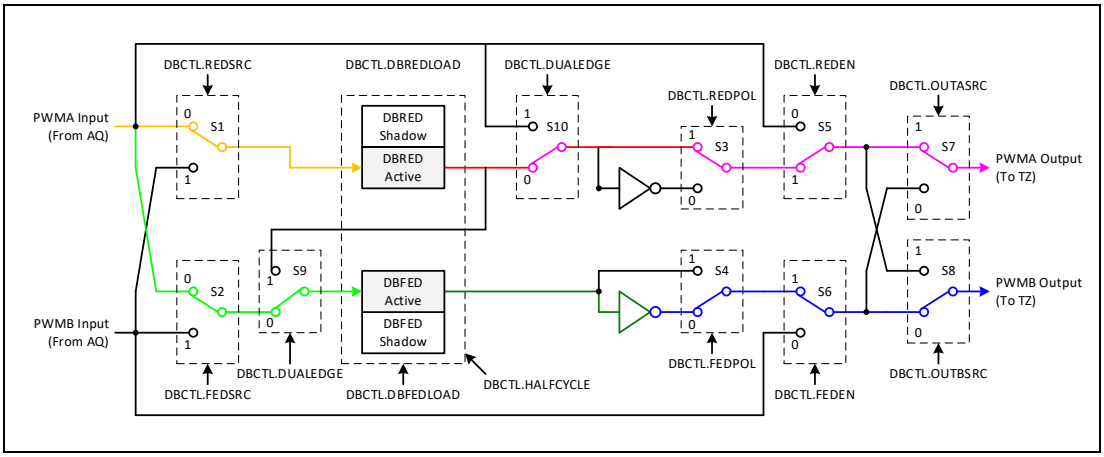
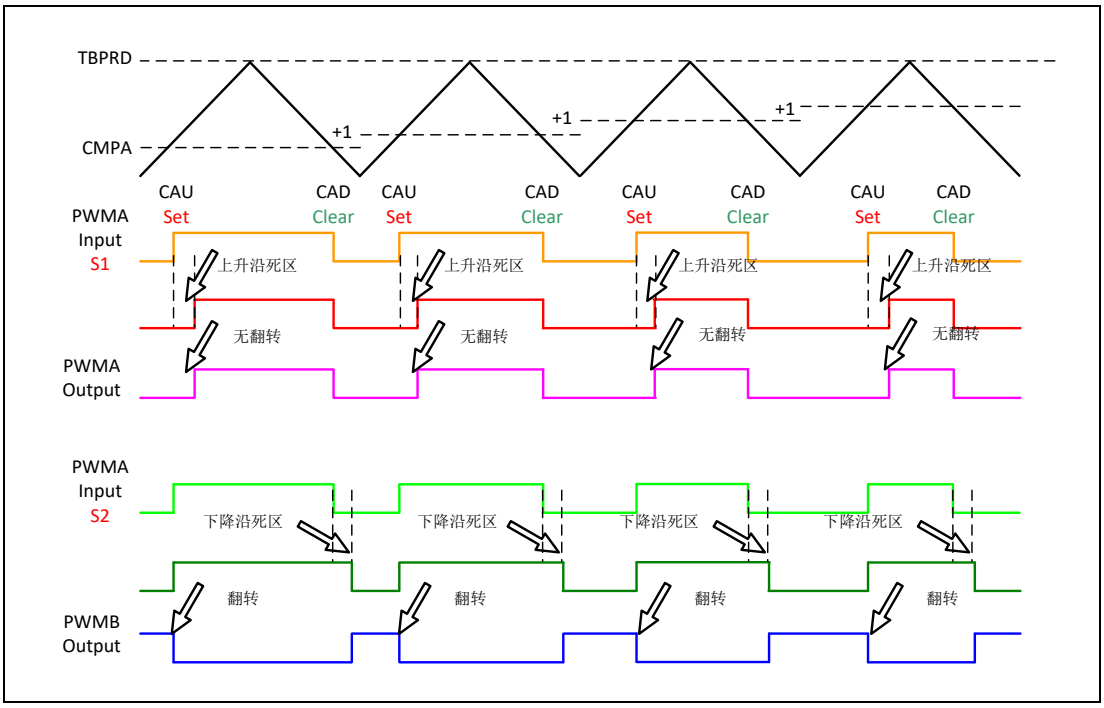


图 2-12 是带死区的互补波形的示例。

- 首先采用上下计数方式，并通过调整 CMPA 的值，由 CAU 和 CAD 事件触发占空比可变的对称波形。
- 选择行为限定（Action Qualifier）模块的 PWMA 输出作为两个延时通路的输入。
- 上升沿延时通路的输出，保持极性不变，作为死区控制模块的 PWMA 输出。
- 下降沿延时通路的输出，经过极性反相，作为死区控制模块的 PWMB 输出。
- 输出的 PWMA 和 PWMB 两个信号互补，且存在两个非交叠死区，其长度为 DBRED 和 DBFED 指定的延时。

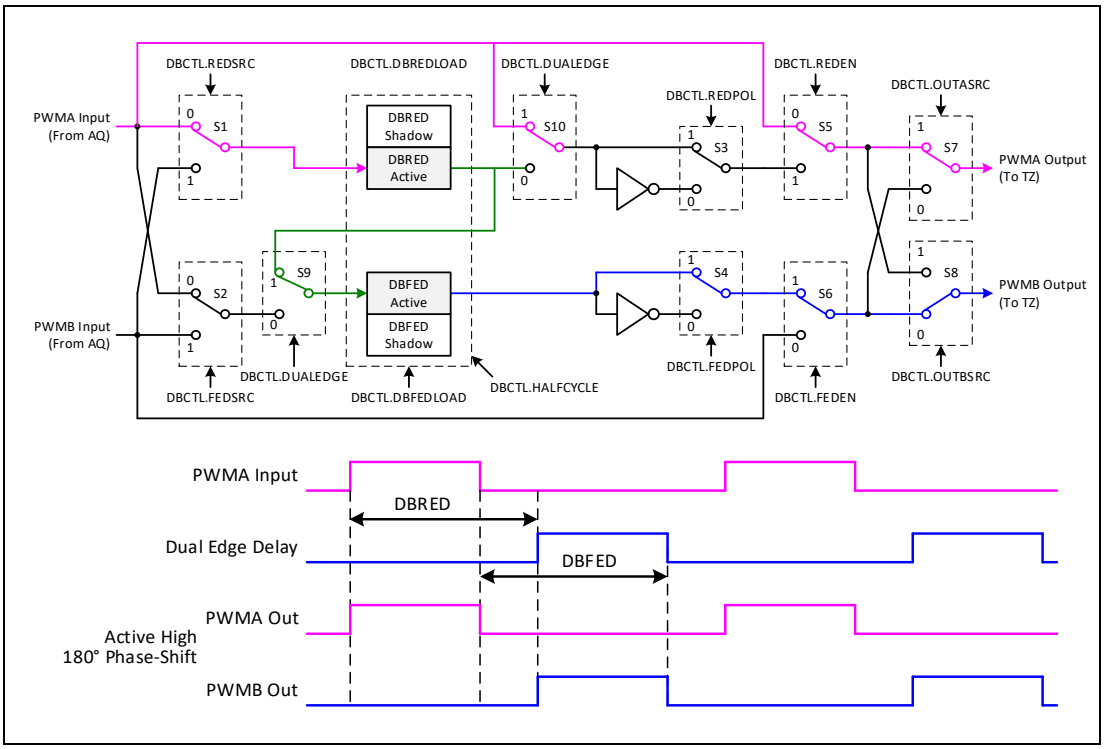
图 2-12: 带死区的互补波形



在某些应用场合中，对于双通道信号的要求不是两者互补，而是有一定的相位差。图 2-13 示范了如何实现这样的波形。

- 开关 S1 选择行为限定（Action Qualifier）模块的 PWMA 输出作为上升沿延时通路的输入。
- 开关 S9 选择经过上升沿延时后的信号作为下降沿延时通路的输入；
- 开关 S5 和 S7 选择原始信号作为最终 PWMA 输出；
- 开关 S4、S6 和 S8 选择经过双边沿延时后的信号作为最终 PWMB 输出；
- 当设定 DBRED=DBFED，则输出的两路信号仅仅存在相位差。

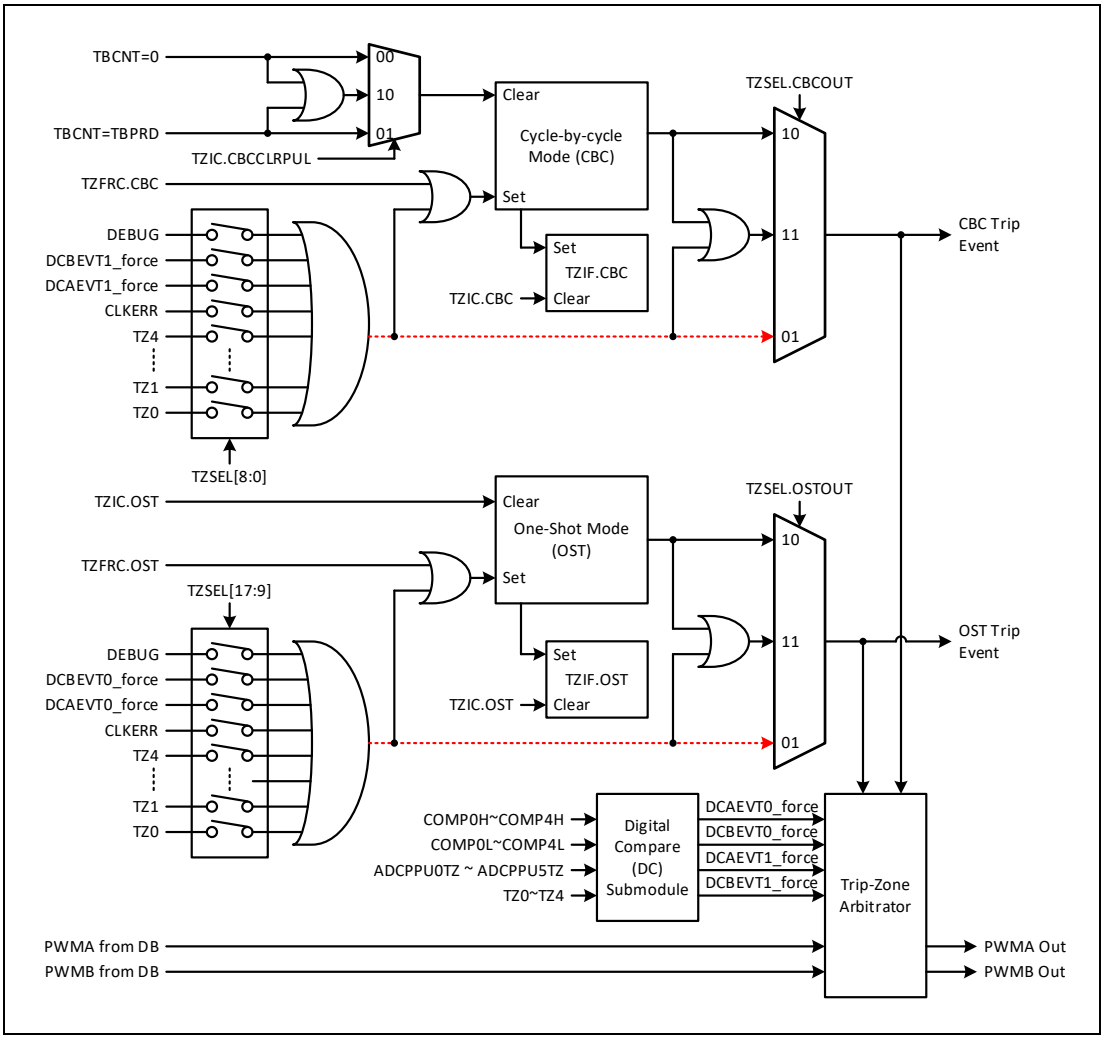
图 2-13: 产生带相位差的两路波形



2.4 信号封锁（Trip-zone）

PWM 封锁功能泛指在设定的特殊状况发生时，在第一时间关闭 PWM，并等待适当的时间重新启动 PWM 波形。如图 2-14 所示，触发封锁的事件包括来自 GPIO 的 TZ0 到 TZ4、来自时钟模块的时钟错误信号、来自 CPU 的调试信号和来自数字比较模块（Digital Compare）的封锁信号。其中 TZ0 到 TZ4 信号可以通过 TZ0SRCCTL 到 TZ4SRCCTL 寄存器来配置是来自于哪个 GPIO 管脚的输入以及对应的信号极性。此外，也可以通过软件方式触发封锁，这通常用于软件调试。

图 2-14: 信号封锁的逻辑框图



在收到封锁信号后，PWM 的输出可以配置为：

- 不操作（不受事件影响）
- 输出置低（Clear）
- 输出置高（Set）
- 输出翻转（Toggle）

如图 2-15 所示，SPC11X8/SPD11X8 的 PWM 支持如下两种封锁方式：

- 周期性封锁（Cycle-by-cycle trip）
  PWM 波形停止后，会在下一个 PWM 周期重新启动，不需要人为介入。这种方式适合用于恒流斩波，如电源控制中的定电流控制，或是步进电机中的恒流细分控制。

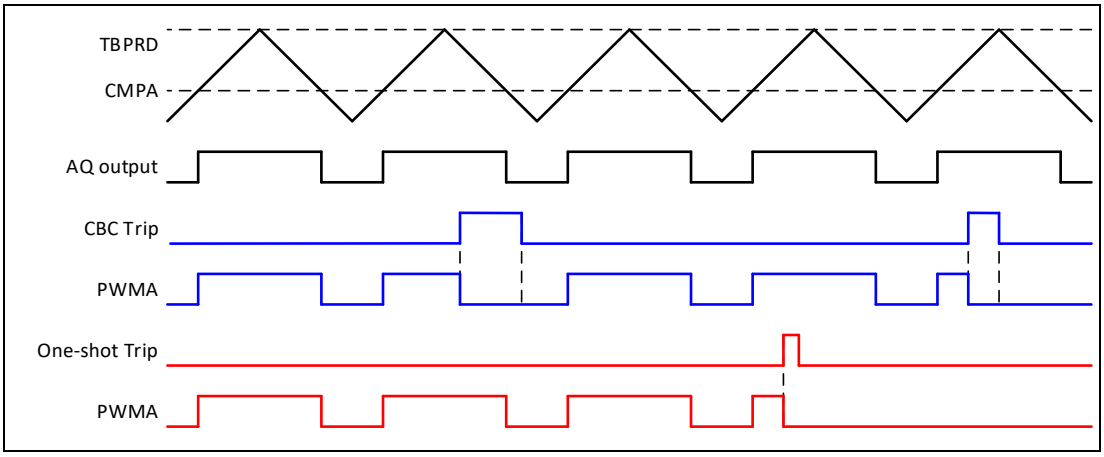
- 一次性持续封锁（One-shot trip）
  PWM 波形停止后，直到使用者介入，清除相关标志之后才会重新输出。这种方式适合用于异常或者错误发生时的处理，可以安全地停止电机。

从图 2-14 和图 2-15 中还可以看出

- 如果封锁信号选择图 2-14 中虚线所表示的未经锁存的通路，则电路实际行为变为只要触发封锁的事件消失，则 PWM 输出立刻恢复正常。因此，该信号通路主要用于调试时观测原始触发信号。要真正实现前述封锁功能，**实际使用中必须选择经过锁存的触发信号，即选择另外两路之一。**

- 如果同样的触发事件被同时配置为用于产生周期性封锁和用于产生一次性封锁，则作用在 PWM 信号上的最终效果等效于产生一次性封锁。

图 2-15: 信号封锁的两种方式



如表 2-4 所示，封锁事件直接作用于 PWM 的最终输出，具有最高的优先级。

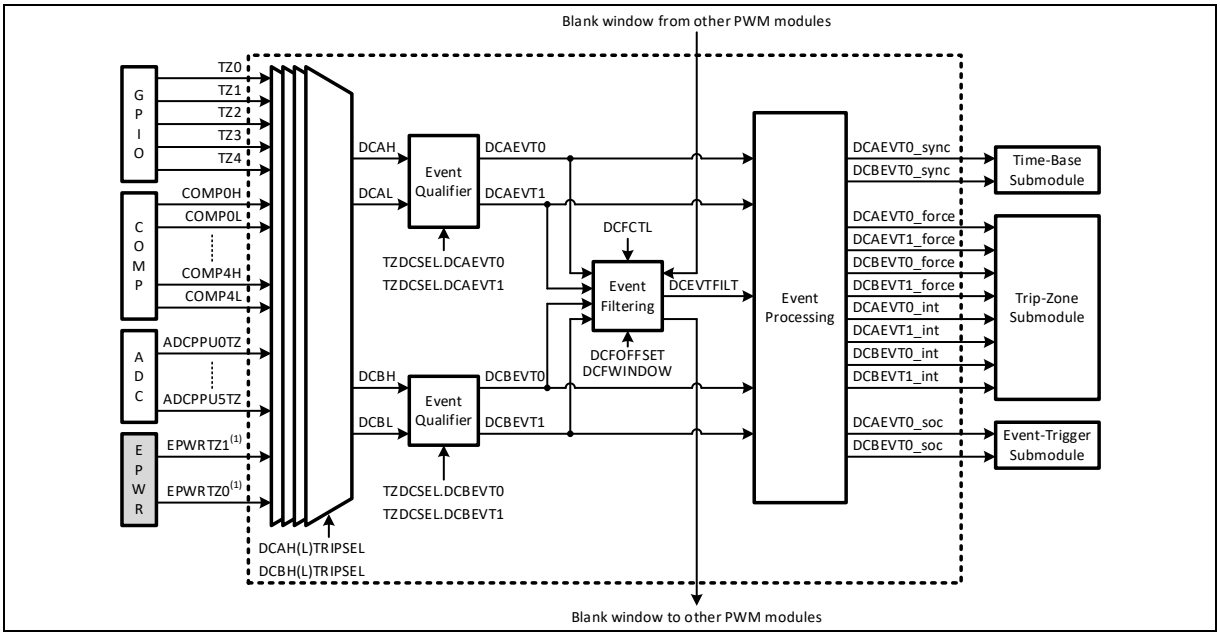
表 2-4: 影响 PWM 最终波形的各事件的优先级

优先级	事件
1（最高）	封锁事件（Trip-zone）
2	通过 AQCSFRC 寄存器的软件持续强制
3	死区控制（Dead-band）逻辑
4	通过 AQSFRC 寄存器的软件单次强制
5（最低）	行为限定（Action Qualifier）逻辑，具体见表 2-1 到表 2-3

2.5 数字比较（Digital-Compare）

如图 2-16 所示，数字比较模块将来自 GPIO 的 TZ0~TZ4 事件、来自比较器（COMP）的 COMPxL 和 COMPxH 事件以及来自 ADC 后处理单元的 ADCPPU0TZ~ADCPPU7TZ 事件作为输入，根据用户配置产生 DCAL、DCAH、DCBL、DCBH 信号，再从这四个信号的各种组合中产生原始的 DCAEVT0、DCAEVT1、DCBEVT0、DCBEVT1 事件，其中一个事件可以经过滤波加以消隐，由用户选择最终的四个事件输出，并可以用来同步计数器、触发 ADC 的 SOC 或者封锁 PWM。

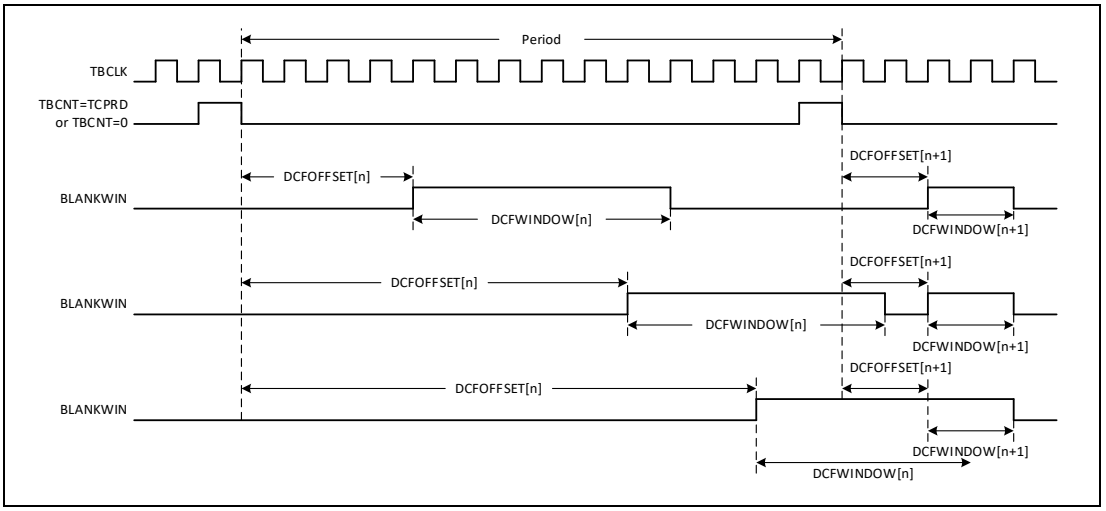
图 2-16: 数字比较模块的逻辑框图



（1） 在 SPC11X8/SPD11X8 中，EPWR 模块处于未连接状态，EPWRTZ0 和 EPWRTZ1 始终为 0。

在实际应用中，ADC 或者 COMP 等模数转换模块负责检测环路反馈回来的电压或者电流信号，判别是否超出阈值并产生相应的封锁信号。在 PWM 输出翻转等时刻附近，这些模数转换模块容易受环境突变的影响做出误操作，产生尖峰信号。如图 2-17 所示，SPC11X8/SPD11X8 的 PWM 中，在数字比较模块中引入了消隐功能，可以在已知的可能出现尖峰封锁信号的时间窗内对原始封锁信号设置盲区，并且可以用任意多个 PWM 产生的时间窗对本 PWM 设置盲区。

图 2-17: 封锁信号的消隐



3 PWM 函数和代码示例

3.1 宏和函数

如表 3-1 和表 3-2 所示，SPC11X8/SPD11X8 的 SDK 中提供了丰富的宏和函数，以便于用户更简易地使用 PWM。表格中的 PWMx 为 PWM 编号，取值范围为 PWM0~PWM7。

表 3-1: PWM 相关的宏

宏名	功能及参数说明
PWM_LinkTBPRD(PWMx, ePWMLink)	设置本地 TBPRD 寄存器所连接的目标 PWM ePWMLink：写操作会引起本地寄存器更新的 PWM（PWM_SelEnum 枚举变量）
PWM_LinkCMPA(PWMx, ePWMLinked)	设置本地 CMPA 寄存器链接的目标 PWM（参数同上）
PWM_LinkCMPB(PWMx, ePWMLinked)	设置本地 CMPB 寄存器链接的目标 PWM（参数同上）
PWM_LinkCMPC(PWMx, ePWMLinked)	设置本地 CMPC 寄存器链接的目标 PWM（参数同上）
PWM_LinkCMPD(PWMx, ePWMLinked)	设置本地 CMPD 寄存器链接的目标 PWM（参数同上）
PWM_LinkDBRED(PWMx, ePWMLinked)	设置本地 DBRED 寄存器链接的目标 PWM（参数同上）
PWM_LinkDBFED(PWMx, ePWMLinked)	设置本地 DBFED 寄存器链接的目标 PWM（参数同上）
PWM_LinkGLDCTL1(PWMx,ePWMLinked)	设置本地 GLDLD1 寄存器链接的目标 PWM（参数同上）
PWM_GetClockDiv(PWMx)	获取 PWM 的计数时钟分频比
PWM_SetClockDiv(PWMx, eDiv0, eDiv1)	设置 PWM 的计数时钟分频比 eDiv0：第一级分频（PWM_ClockDiv0Enum 枚举变量） eDiv1：第二级分频（PWM_ClockDiv1Enum 枚举变量）
PWM_GetCounterValue(PWMx)	获取 PWM 当前计数值（16 位无符号数）
PWM_GetPeriod(PWMx) PWM_GetPRD(PWMx)	获取 PWM 有效的周期值 TBPRDA（16 位无符号数）
PWM_SetPeriod(PWMx, u16Val) PWM_SetPRD(PWMx, u16Val)	设置 PWM 周期值（TBPRD） u16Val：周期值（16 位无符号数），以计数时钟为单位
PWM_SetSyncReloadValue(PWMx, u16Val)	设置同步事件发生后计数器的重置值 u16Val：重置值（16 位无符号数）
PWM_EnableSync(PWMx)	设定 PWM 计数被同步信号影响
PWM_DisableSync(PWMx)	设定 PWM 计数不被同步信号影响
PWM_SetSyncOutEvent (PWMx, eSYNCO)	设置 PWM 输出的同步信号 eSYNCO：输出选择（PWM_SyncOutputEnum 枚举变量）
PWM_SetCounterDirAfterSync (PWMx, eDir)	设置同步时间发生后的计数方向 eDir：COUNT_UP（向上）或 COUNT_DOWN（向下）
PWM_StopCounterInDebug(PWMx)	设置调试模式下 PWM 完成一个计数周期后停止

宏名	功能及参数说明
PWM_SetCounterMode(PWMx, eMode)	设置计数模式 eMode：计数模式（PWM_CounterModeEnum 枚举变量）
PWM_RunCounter(PWMx)	运行计数
PWM_StopCounter(PWMx)	停止计数
PWM_GetCountingDirection(PWMx)	查询当前计数方向
PWM_IsCountingUp(PWMx)	判断当前是否正在向上计数
PWM_IsCountingDown(PWMx)	判断当前是否正在向下计数
PWM_SetCMPA(PWMx, u16Val)	设置 CMPA 的值 u16Val：16 位无符号数
PWM_SetCMPB(PWMx, u16Val)	设置 CMPB 的值（参数同上）
PWM_SetCMPC(PWMx, u16Val)	设置 CMPC 的值（参数同上）
PWM_SetCMPD(PWMx, u16Val)	设置 CMPD 的值（参数同上）
PWM_GetCMPA(PWMx)	获取 CMPA 的有效值（16 位无符号数）
PWM_GetCMPB(PWMx)	获取 CMPB 的有效值（16 位无符号数）
PWM_GetCMPC(PWMx)	获取 CMPC 的有效值（16 位无符号数）
PWM_GetCMPD(PWMx)	获取 CMPD 的有效值（16 位无符号数）
PWM_SetCMPALoadTiming(PWMx, eMode)	设置 CMPA 有效值更新方式 eMode：更新方式（PWM_LoadTimingEnum 枚举变量）
PWM_SetCMPBLoadTiming(PWMx, eMode)	设置 CMPB 有效值更新方式 eMode：更新方式（PWM_LoadTimingEnum 枚举变量）
PWM_SetCMPCLoadTiming(PWMx, eMode)	设置 CMPC 有效值更新方式 eMode：更新方式（PWM_LoadTimingEnum 枚举变量）
PWM_SetCMPDLoadTiming(PWMx, eMode)	设置 CMPD 有效值更新方式 eMode：更新方式（PWM_LoadTimingEnum 枚举变量）
PWM_UnlockCMPA(PWMx)	允许更新 CMPA
PWM_UnlockCMPB(PWMx)	允许更新 CMPB
PWM_UnlockCMPC(PWMx)	允许更新 CMPC
PWM_UnlockCMPD(PWMx)	允许更新 CMPD
PWM_LockCMPA(PWMx)	禁止更新 CMPA
PWM_LockCMPB(PWMx)	禁止更新 CMPB
PWM_LockCMPC(PWMx)	禁止更新 CMPC
PWM_LockCMPD(PWMx)	禁止更新 CMPD
PWM_SetAQCTLALoadTiming(PWMx, eMode)	设置 AQCTLA 有效值更新方式 eMode：更新方式（PWM_LoadTimingEnum 枚举变量）
PWM_SetAQCTLBLoadTiming(PWMx, eMode)	设置 AQCTLB 有效值更新方式 eMode：更新方式（PWM_LoadTimingEnum 枚举变量）
PWM_UnlockAQCTLA(PWMx)	允许更新 AQCTLA
PWM_UnlockAQCTLB(PWMx)	允许更新 AQCTLB
PWM_LockAQCTLA(PWMx)	禁止更新 AQCTLA

宏名	功能及参数说明
PWM_LockAQCTLB(PWMx)	禁止更新 AQCTLB
PWM_ActionQualifierCHA(PWMx, eAction)	设置 PWMxA 的行为限定 eAction：行为（PWM_ActionQualifierEnum 枚举变量）
PWM_ActionQualifierCHB(PWMx, eAction)	设置 PWMxB 的行为限定 eAction：行为（PWM_ActionQualifierEnum 枚举变量）
PWM_SetT0Event(PWMx, eEvent)	配置 T0 事件 eEvent：事件选择（PWM_TxEventEnum 枚举变量）
PWM_SetT1Event(PWMx, eEvent)	配置 T1 事件 eEvent：事件选择（PWM_TxEventEnum 枚举变量）
PWM_ForceCHALow(PWMx)	软件强制 PWMxA 持续为低
PWM_ForceCHBLow(PWMx)	软件强制 PWMxB 持续为低
PWM_ForceCHAandCHBLow(PWMx)	软件强制 PWMxA 和 PWMxB 持续为低
PWM_ForceCHAHigh(PWMx)	软件强制 PWMxA 持续为高
PWM_ForceCHBHigh(PWMx)	软件强制 PWMxB 持续为高
PWM_ForceCHAandCHBHigh(PWMx)	软件强制 PWMxA 和 PWMxB 持续为高
PWM_DisableForceCHA(PWMx)	撤销对 PWMxA 的软件持续强制
PWM_DisableForceCHB(PWMx)	撤销对 PWMxB 的软件持续强制
PWM_DisableForceCHAandCHB(PWMx)	撤销对 PWMxA 和 PWMxB 的软件持续强制
PWM_SetDBCTLLoadTiming(PWMx, eMode)	设置 DBCTL 有效值更新方式 eMode：更新方式（PWM_LoadTimingEnum 枚举变量）
PWM_SetDBREDLoadTiming(PWMx, eMode)	设置 DBRED 有效值更新方式 eMode：更新方式（PWM_LoadTimingEnum 枚举变量）
PWM_SetDBFEDLoadTiming(PWMx, eMode)	设置 DBFED 有效值更新方式 eMode：更新方式（PWM_LoadTimingEnum 枚举变量）
PWM_UnlockDBCTL (PWMx)	允许更新 DBCTL
PWM_UnlockDBRED(PWMx)	允许更新 DBRED
PWM_UnlockDBFED(PWMx)	允许更新 DBFED
PWM_LockDBCTL (PWMx)	禁止更新 DBCTL
PWM_LockDBRED(PWMx)	禁止更新 DBRED
PWM_LockDBFED(PWMx)	禁止更新 DBFED
PWM_SetDeadBandRisingDelay(PWMx, u16Delay)	设定死区控制模块上升沿通路的延时 u16Delay：延时值（16 位无符号数），单位为计数时钟
PWM_SetDeadBandFallingDelay(PWMx, u16Delay)	设定死区控制模块下降沿通路的延时 u16Delay：延时值（16 位无符号数），单位为计数时钟
PWM_GetDeadBandRisingDelay(PWMx)	获取死区控制模块上升沿通路的延时，单位为计数时钟
PWM_GetDeadBandFallingDelay(PWMx)	获取死区控制模块下降沿通路的延时，单位为计数时钟
PWM_EnableTripWhenDebug(PWMx)	PWM 输出受调试模式信号周期性封锁使能
PWM_SetOneShotTripEvent(PWMx, eEvent, eType)	设置 PWM 一次性封锁信号的触发事件 eEvent：事件选择（PWM_TripEventEnum 枚举变量） eType：事件类型（PWM_TripOutputEnum 枚举变量）

宏名	功能及参数说明
PWM_SetCBCTripEvent(PWMx, eEvent, eType)	设置 PWM 周期性封锁信号的触发事件 eEvent：事件选择（PWM_TripEventEnum 枚举变量） eType：事件类型（PWM_TripOutputEnum 枚举变量）
PWM_SetCHAOutputWhenTrip(PWMx, eAction)	设置 PWMxA 输出在接收到封锁信号时的行为 eAction：行为选择（PWM_TripActionEnum 枚举变量）
PWM_SetCHBOutputWhenTrip(PWMx, eAction)	设置 PWMxB 输出在接收到封锁信号时的行为 eAction：行为选择（PWM_TripActionEnum 枚举变量）
PWM_EnableOneShotTripInt(PWMx)	允许 PWM 一次性封锁事件触发中断
PWM_DisableOneShotTripInt(PWMx)	禁止 PWM 一次性封锁事件触发中断
PWM_EnableCBCTripInt(PWMx)	允许 PWM 周期性封锁事件触发中断
PWM_DisableCBCTripInt(PWMx)	禁止 PWM 周期性封锁事件触发中断
PWM_EnableDCAEVT0TripInt(PWMx)	允许 PWM DCAEVT0 封锁事件触发中断
PWM_DisableDCAEVT0TripInt(PWMx)	禁止 PWM DCAEVT0 封锁事件触发中断
PWM_EnableDCAEVT1TripInt(PWMx)	允许 PWM DCAEVT1 封锁事件触发中断
PWM_DisableDCAEVT1TripInt(PWMx)	禁止 PWM DCAEVT1 封锁事件触发中断
PWM_EnableDCBEVT0TripInt(PWMx)	允许 PWM DCBEVT0 封锁事件触发中断
PWM_DisableDCBEVT0TripInt(PWMx)	禁止 PWM DCBEVT0 封锁事件触发中断
PWM_EnableDCBEVT1TripInt(PWMx)	允许 PWM DCBEVT1 封锁事件触发中断
PWM_DisableDCBEVT1TripInt(PWMx)	禁止 PWM DCBEVT1 封锁事件触发中断
PWM_GetOneShotTripIntFlag(PWMx)	查询 PWM 一次性封锁中断标志
PWM_GetCBCTripIntFlag(PWMx)	查询 PWM 周期性封锁中断标志
PWM_GetDCAEVT0TripIntFlag(PWMx)	查询 PWM DCAEVT0 封锁中断标志
PWM_GetDCAEVT1TripIntFlag(PWMx)	查询 PWM DCAEVT1 封锁中断标志
PWM_GetDCBEVT0TripIntFlag(PWMx)	查询 PWM DCBEVT0 封锁中断标志
PWM_GetDCBEVT1TripIntFlag(PWMx)	查询 PWM DCBEVT1 封锁中断标志
PWM_GetTripGlobalIntFlag(PWMx)	查询 PWM 全局封锁中断标志
PWM_ClearOneShotTripInt(PWMx)	清除 PWM 一次性封锁中断标志
PWM_ClearCBCTripInt(PWMx)	清除 PWM 周期性封锁中断标志
PWM_ClearDCAEVT0TripInt(PWMx)	清除 PWM DCAEVT0 封锁中断标志
PWM_ClearDCAEVT1TripInt(PWMx)	清除 PWM DCAEVT1 封锁中断标志
PWM_ClearDCBEVT0TripInt(PWMx)	清除 PWM DCBEVT0 封锁中断标志
PWM_ClearDCBEVT1TripInt(PWMx)	清除 PWM DCBEVT1 封锁中断标志
PWM_ClearTripGlobalInt(PWMx)	清除 PWM 全局封锁中断标志
PWM_EnableDCAHTripEvent(PWMx, eEvent)	设置触发 DCAH 封锁的事件 eEvent：事件选择（PWM_DCTripEventEnum 枚举变量,事件之间可进行与操作,使多事件可触发 DCAH）
PWM_EnableDCALTripEvent(PWMx, eEvent)	设置触发 DCAL 封锁的事件 eEvent：事件选择（PWM_DCTripEventEnum 枚举变量,事件之间可进行与操作,使多事件可触发 DCAL）

宏名	功能及参数说明
PWM_EnableDCBHTripEvent(PWMx, eEvent)	设置触发 DCBH 封锁的事件 eEvent：事件选择（PWM_DCTripEventEnum 枚举变量,事件之间可进行与操作,使多事件可触发 DCBH）
PWM_EnableDCBLTripEvent(PWMx, eEvent)	设置触发 DCBL 封锁的事件 eEvent：事件选择（PWM_DCTripEventEnum 枚举变量,事件之间可进行与操作,使多事件可触发 DCBL）
PWM_DisableDCAHTripEvent(PWMx, eEvent)	删除触发 DCAH 封锁的事件 eEvent：事件选择（PWM_DCTripEventEnum 枚举变量）
PWM_DisableDCALTripEvent(PWMx, eEvent)	删除触发 DCAL 封锁的事件 eEvent：事件选择（PWM_DCTripEventEnum 枚举变量）
PWM_DisableDCBHTripEvent(PWMx, eEvent)	删除触发 DCBH 封锁的事件 eEvent：事件选择（PWM_DCTripEventEnum 枚举变量）
PWM_DisableDCBLTripEvent(PWMx, eEvent)	删除触发 DCBL 封锁的事件 eEvent：事件选择（PWM_DCTripEventEnum 枚举变量）
PWM_SetRawDCAEVT0(PWMx, eEvent)	设定原始 DCAEVT0 事件 eEvent：事件选择（PWM_RawDCEventEnum 枚举变量）
PWM_SetRawDCAEVT1(PWMx, eEvent)	设定原始 DCAEVT1 事件 eEvent：事件选择（PWM_RawDCEventEnum 枚举变量）
PWM_SetRawDCBEVT0(PWMx, eEvent)	设定原始 DCABVT0 事件 eEvent：事件选择（PWM_RawDCEventEnum 枚举变量）
PWM_SetRawDCBEVT1(PWMx, eEvent)	设定原始 DCABVT1 事件 eEvent：事件选择（PWM_RawDCEventEnum 枚举变量）
PWM_SetDCAEVT0(PWMx, eEvent)	设定最终 DCAEVT0 事件 eEvent：事件选择（PWM_DCEventEnum 枚举变量）
PWM_SetDCAEVT1(PWMx, eEvent)	设定最终 DCAEVT1 事件 eEvent：事件选择（PWM_DCEventEnum 枚举变量）
PWM_SetDCBEVT0(PWMx, eEvent)	设定最终 DCBEVT0 事件 eEvent：事件选择（PWM_DCEventEnum 枚举变量）
PWM_SetDCBEVT1(PWMx, eEvent)	设定最终 DCBEVT1 事件 eEvent：事件选择（PWM_DCEventEnum 枚举变量）
PWM_EnableDCAEVT0TriggerSync(PWMx)	允许 DCAEVT0 事件触发计数同步
PWM_DisableDCAEVT0TriggerSync(PWMx)	禁止 DCAEVT0 事件触发计数同步
PWM_EnableDCBEVT0TriggerSync(PWMx)	允许 DCBEVT0 事件触发计数同步
PWM_DisableDCBEVT0TriggerSync(PWMx)	禁止 DCBEVT0 事件触发计数同步
PWM_EnableDCAEVT0TriggerSOC(PWMx)	允许 DCAEVT0 事件触发 ADC SOC
PWM_DisableDCAEVT0TriggerSOC(PWMx)	禁止 DCAEVT0 事件触发 ADC SOC
PWM_EnableDCBEVT0TriggerSOC(PWMx)	允许 DCBEVT0 事件触发 ADC SOC
PWM_DisableDCBEVT0TriggerSOC(PWMx)	禁止 DCBEVT0 事件触发 ADC SOC

宏名	功能及参数说明
PWM_SetDCFilter(PWMx, eEvent, eAlign)	配置 PWM 的 DC 事件滤波器 eEvent：事件输入（PWM_DCFilterInputEnum 枚举变量） eAlign：对齐方式（PWM_DCFilterAlignEnum 枚举变量）
PWM_EnableDCFilterFromOtherPWM(PWMx, ePWMs)	允许其他 PWM 模块的 DC 事件滤波器作用于当前 PWM 模块
PWM_DisableDCFilterFromOtherPWM(PWMx ,ePWMs)	禁止其他 PWM 模块的 DC 事件滤波器作用于当前 PWM 模块
PWM_EnableDCFilterBlank(PWMx)	允许 PWM 的 DC 事件盲区窗口
PWM_DisableDCFilterBlank(PWMx)	禁止 PWM 的 DC 事件盲区窗口
PWM_EnableDCFilterBlankInvert(PWMx)	允许 PWM 的 DC 事件盲区窗口反转
PWM_DisableDCFilterBlankInvert(PWMx)	禁止 PWM 的 DC 事件盲区窗口反转
PWM_SetDCFilterBlankWindow(PWMx, u16Size, u16Offset)	配置 PWM 的 DC 事件盲区窗口大小和偏移 u16Size：盲区窗口的大小（16 位无符号数） u16Offset：盲区窗口的偏移（16 位无符号数）
PWM_EnableSOCA(PWMx)	允许 PWM 产生 SOCA 请求
PWM_DisableSOCA(PWMx)	禁止 PWM 产生 SOCA 请求
PWM_SetSOCATiming(PWMx, eTiming)	设置 PWM 在何时产生 SOCA 请求 eTiming：时机选择（PWM_EventTimingEnum 枚举变量）
PWM_SetSOCAPeriod (PWMx, ePeriod)	设置 PWM 产生 SOCA 请求的周期 ePeriod：周期选择（PWM_EventPeriodEnum 枚举变量）
PWM_EnableSOCB(PWMx)	允许 PWM 产生 SOCB 请求
PWM_DisableSOCB(PWMx)	禁止 PWM 产生 SOCB 请求
PWM_SetSOCBTiming(PWMx, eTiming)	设置 PWM 在何时产生 SOCB 请求 eTiming：时机选择（PWM_EventTimingEnum 枚举变量）
PWM_SetSOCBPeriod (PWMx, ePeriod)	设置 PWM 产生 SOCB 请求的周期 ePeriod：周期选择（PWM_EventPeriodEnum 枚举变量）
PWM_EnableSOCC(PWMx)	允许 PWM 产生 SOCC 请求
PWM_DisableSOCC(PWMx)	禁止 PWM 产生 SOCC 请求
PWM_SetSOCCTiming(PWMx, eTiming)	设置 PWM 在何时产生 SOCC 请求 eTiming：时机选择（PWM_EventTimingEnum 枚举变量）
PWM_SetSOCCPeriod (PWMx, ePeriod)	设置 PWM 产生 SOCC 请求的周期 ePeriod：周期选择（PWM_EventPeriodEnum 枚举变量）
PWM_EnableTimeEventInt(PWMx)	允许 PWM 产生中断请求
PWM_DisableTimeEventInt(PWMx)	禁止 PWM 产生中断请求
PWM_SetTimeEventTiming(PWMx, eTiming)	设置 PWM 在何时产生中断请求 eTiming：时机选择（PWM_EventTimingEnum 枚举变量）
PWM_SetTimeEventPeriod(PWMx, ePeriod)	设置 PWM 产生中断请求的周期 ePeriod：周期选择（PWM_EventPeriodEnum 枚举变量）
PWM_GetTimeEventIntFlag(PWMx)	查询 PWM 中断标志
PWM_ClearTimeEventIntFlag(PWMx)	清除 PWM 中断标志

宏名	功能及参数说明
PWM_GetSOCAEventFlag(PWMx)	查询 SOCA 产生标志
PWM_GetSOCBEventFlag(PWMx)	查询 SOCB 产生标志
PWM_GetSOCCEventFlag(PWMx)	查询 SOCC 产生标志
PWM_ClearSOCAEvent(PWMx)	清除 SOCA 产生标志
PWM_ClearSOCBEvent(PWMx)	清除 SOCB 产生标志
PWM_ClearSOCCEvent(PWMx)	清除 SOCC 产生标志
PWM_WALLOW(PWMx)	允许写入受保护的 PWM 寄存器
PWM_WDIS(PWMx)	禁止写入受保护的 PWM 寄存器
PWM_ForceClockSync()	对所有 PWM 模块的时钟进行软件强制同步
PWM_ForceSync(ePWMIdx)	对指定的 PWM 模块的计数进行软件强制同步 ePWMIdx：PWM 选择（PWM_IncEnum 枚举变量）
PWM_EnableSyncFromGPIO(ePWMIdx)	指定会被来自 GPIO 的 PWMSYNC 信号同步的 PWM ePWMIdx：PWM 选择（PWM_IncEnum 枚举变量）
PWM_DisableSyncFromGPIO(ePWMIdx)	撤销会被来自 GPIO 的 PWMSYNC 信号同步的 PWM ePWMIdx：PWM 选择（PWM_IncEnum 枚举变量）
PWM_EnableSyncFromTIMER0(ePWMIdx)	指定会被来自 TIMER0 的 PWMSYNC 信号同步的 PWM ePWMIdx：PWM 选择（PWM_IncEnum 枚举变量）
PWM_DisableSyncFrom TIMER0(ePWMIdx)	撤销会被来自 TIMER0 的 PWMSYNC 信号同步的 PWM ePWMIdx：PWM 选择（PWM_IncEnum 枚举变量）
PWM_EnableSyncFromTIMER1(ePWMIdx)	指定会被来自 TIMER1 的 PWMSYNC 信号同步的 PWM ePWMIdx：PWM 选择（PWM_IncEnum 枚举变量）
PWM_DisableSyncFrom TIMER1(ePWMIdx)	撤销会被来自 TIMER1 的 PWMSYNC 信号同步的 PWM ePWMIdx：PWM 选择（PWM_IncEnum 枚举变量）
PWM_EnableSyncFromTIMER2(ePWMIdx)	指定会被来自 TIMER2 的 PWMSYNC 信号同步的 PWM ePWMIdx：PWM 选择（PWM_IncEnum 枚举变量）
PWM_DisableSyncFrom TIMER2(ePWMIdx)	撤销会被来自 TIMER2 的 PWMSYNC 信号同步的 PWM ePWMIdx：PWM 选择（PWM_IncEnum 枚举变量）
PWMCFG_WALLOW()	允许写入受保护的 PWMCFG（PWM 全局配置）寄存器
PWMCFG_WDIS ()	禁止写入受保护的 PWMCFG（PWM 全局配置）寄存器

表 3-2: PWM 相关的函数

函数名	功能及参数说明
void PWM_SingleChannelInit(PWM_REGS* PWMx, PWM_ChannelEnum ePWM_CH, uint32_t u32PWMFreqHz)	快速配置单信道独立 PWM 波形 PWMx：PWM_REGS 指针（PWM0~PWM7） ePWM_CH：输出通道（PWM_CHA 或 PWM_CHB） u32PWMFreqHz：输出波形的周期，单位为 Hz
void PWM_ComplementaryPairChannelInit(PWM_REGS* PWMx, uint32_t u32PWMFreqHz, uint32_t u32DeadTimeNs)	快速配置双信道互补 PWM 波形 PWMx：PWM_REGS 指针（PWM0~PWM7） u32PWMFreqHz：输出波形的周期，单位为 Hz u32DeadTimeNs：死区时间，单位为 ns

函数名	功能及参数说明
PWM_SetTZ0FromGPIO(ePinNum, ePinLevel);	设置触发 TZ0 的外部管脚和电平 ePinNum：由 GPIO_PinEnum 枚举变量指定的管脚号 ePinLevel：由 GPIO_LevelEnum 枚举变量指定的管脚电平
PWM_SetTZ1FromGPIO (ePinNum, ePinLevel)	设置触发 TZ1 的外部管脚和电平 ePinNum：由 GPIO_PinEnum 枚举变量指定的管脚号 ePinLevel：由 GPIO_LevelEnum 枚举变量指定的管脚电平
PWM_SetTZ2FromGPIO (ePinNum, ePinLevel)	设置触发 TZ2 的外部管脚和电平 ePinNum：由 GPIO_PinEnum 枚举变量指定的管脚号 ePinLevel：由 GPIO_LevelEnum 枚举变量指定的管脚电平
PWM_SetTZ3FromGPIO (ePinNum, ePinLevel)	设置触发 TZ3 的外部管脚和电平 ePinNum：由 GPIO_PinEnum 枚举变量指定的管脚号 ePinLevel：由 GPIO_LevelEnum 枚举变量指定的管脚电平
PWM_SetTZ4FromGPIO (ePinNum, ePinLevel)	设置触发 TZ4 的外部管脚和电平 ePinNum：由 GPIO_PinEnum 枚举变量指定的管脚号 ePinLevel：由 GPIO_LevelEnum 枚举变量指定的管脚电平
PWM_SetGPIOSync (ePinNum, ePinLevel)	设置触发 PWMSYNCI 的外部管脚和电平 ePinNum：由 GPIO_PinEnum 枚举变量指定的管脚号 ePinLevel：由 GPIO_LevelEnum 枚举变量指定的管脚电平

3.2 设定单通道独立波形

建议按照如下步骤设定 PWM 独立单通道：

（1） 务必先调用 CLOCK_InitWithRCO()或 CLOCK_Init()初始化时钟，否则 PWM 时钟可能有误；

（2） 调用 PWM_SingleChannelInit()，初始化基本波形，默认配置 PWM 为上下计数的中央对称波形；

（3） 可以调用 PWM_SetCounterMode()重新设定 PWM 计数方式为如下四种之一；

- 向上计数（Count Up）
- 向下计数（Count Down）
- 上下计数（Count Up and Down）
- 停止计数（Counter Stop）

（4） 默认的 PWM 周期参数按照上下计数方式被配置为：

TBPRD = PWM_Clock_Freq / PWM_Freq/2

若重新设定 PWM 计数方式为向上计数或者向下计数，则要调用 PWM_SetPeriodValue()函数，按照如下公式重新设置 PWM 的计数周期：

TBPRD = PWM_Clock_Freq/PWM_Freq - 1

（5） 调用 PWM_RunCounter()计数使能；

（6） 调整 AQ 配置。默认的 AQ 按照上下计数被配置为：向上计数过 CMP 时，输出翻转为高；向下计数过 CMP 时，输出翻转为低，故 CMP 越高则占空越小。

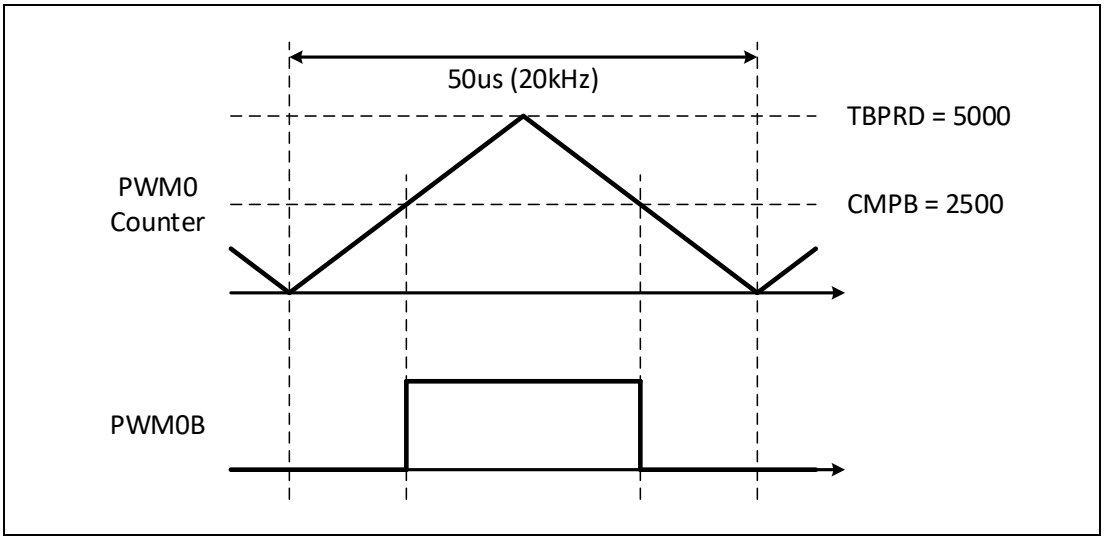
（7） 调整占空比。

（8） 将管脚配置由 GPIO 改为 PWM 输出，真正将 PWM 波形输出于管脚上。

示例代码 3-1：配置 PWM0B 输出 50%占空比的 20kHz 上下计数中央对称波形

```
void main()
{
  FLASH_WALLOW();
  FLASH_SetTiming(200000000);
  FLASH_WDIS();

  CLOCK_InitWithRCO(CLOCK_HCLK_200MHZ);

  Delay_Init();

  /*
   * Init the UART
   *
   * 1.Set the GPIO44/45 as UART FUNC
   *
   * 2.Enable the UART CLK
   *
   * 3.Set the rest para
   */
  GPIO_SetPinChannel(GPIO_34,GPIO34_UART_TXD);
  GPIO_SetPinChannel(GPIO_35,GPIO35_UART_RXD);
  CLOCK_EnableModule(UART_MODULE);
  UART_Init(UART,38400);

   /*** Step 2: PWM initial on PWM0B ***/
  PWM_SingleChannelInit(PWM0,PWM_CHB,20000);

  /*** Step 3: Set counter mode ***/
  /*** Counter mode is already set to up-down counting ***/
  /*** Step 4: Set Period ***/
  /*** Period is already set to 200000000/200000/2 = 5000 ***/
  /*** Step 5: Start counting ***/
  PWM_RunCounter(PWM0);

  /*** Step 6: Configure the AQ ***/
  /*** AQ is already configured as set on CBU and clear on CBD ***/
  /*** Step 7: Drive PWM0B output 50% Duty ***/
  PWM_SetCMPB(PWM0,2500);

  /*** Step 8: Select GPIO19 as PWM0B ***/
  GPIO_SetPinChannel(GPIO_19,GPIO19_PWM0B);
}
```

图 3-1: PWM 单通道中央对称独立波形示例

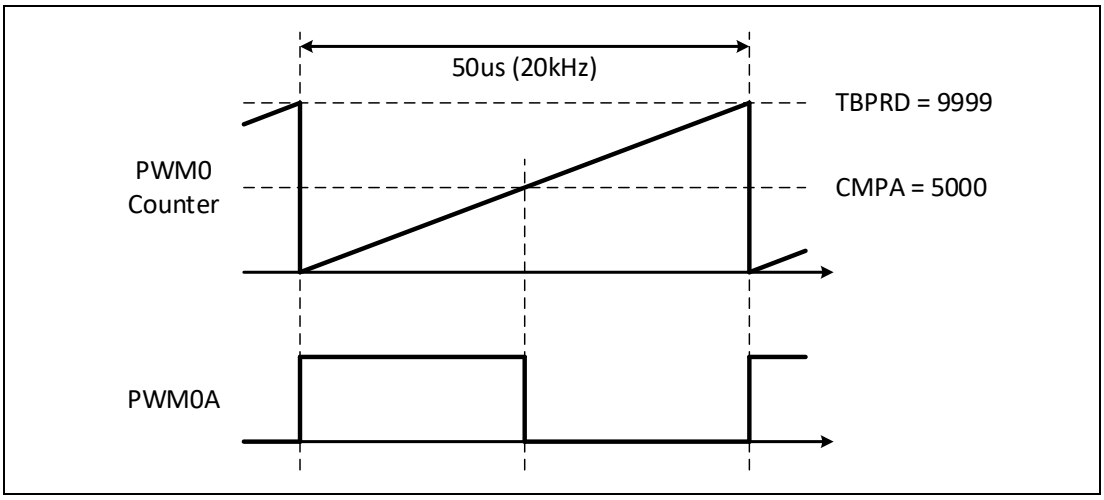


示例代码 3-2：配置 PWM0A 输出向上计数的 50%占空比的 20kHz 波形

```
void main()
{
  FLASH_WALLOW();
  FLASH_SetTiming(200000000);
  FLASH_WDIS();

  CLOCK_InitWithRCO(CLOCK_HCLK_200MHZ);

  Delay_Init();

  /*
   * Init the UART
   *
   * 1.Set the GPIO44/45 as UART FUNC
   *
   * 2.Enable the UART CLK
   *
   * 3.Set the rest para
   */
  GPIO_SetPinChannel(GPIO_34,GPIO34_UART_TXD);
  GPIO_SetPinChannel(GPIO_35,GPIO35_UART_RXD);
  CLOCK_EnableModule(UART_MODULE);
  UART_Init(UART,38400);

   /*** Step 2: PWM initial on PWM0B ***/
  /*** Period is set to 200000000/200000/2 = 5000 ***/
  PWM_SingleChannelInit(PWM0,PWM_CHA,20000);
```

```
  /*** Step 3: Set up-counting mode ***/
  PWM_SetCounterMode(PWM0, COUNT_UP);

  /*** Step 4: Set Period as 200000000/200000 - 1 = 9999 ***/
  PWM_SetPeriodValue(PWM0,9999);

  /*** Step 5: Start counting ***/
  PWM_RunCounter(PWM0);

  /*** Step 6: Configure the AQ ***/
  PWM_ActionQualifierCHA(PWM0, AQCTLA_ALL_ZRO_SET_HIGH
                              |AQCTLA_ALL_CAU_SET_LOW
                              |AQCTLA_ALL_PRD_DO_NOTHING
                              |AQCTLA_ALL_CAD_DO_NOTHING);

  /*** Step 7: Drive PWM0A output 50% Duty ***/
  PWM_SetCMPA(PWM0,5000); /* 0 for 0% duty and 10000 for 100% duty
*/
  /*** Step 8: Select GPIO18 as PWM0A ***/
  GPIO_SetPinChannel(GPIO_18,GPIO18_PWM0A);
}
```

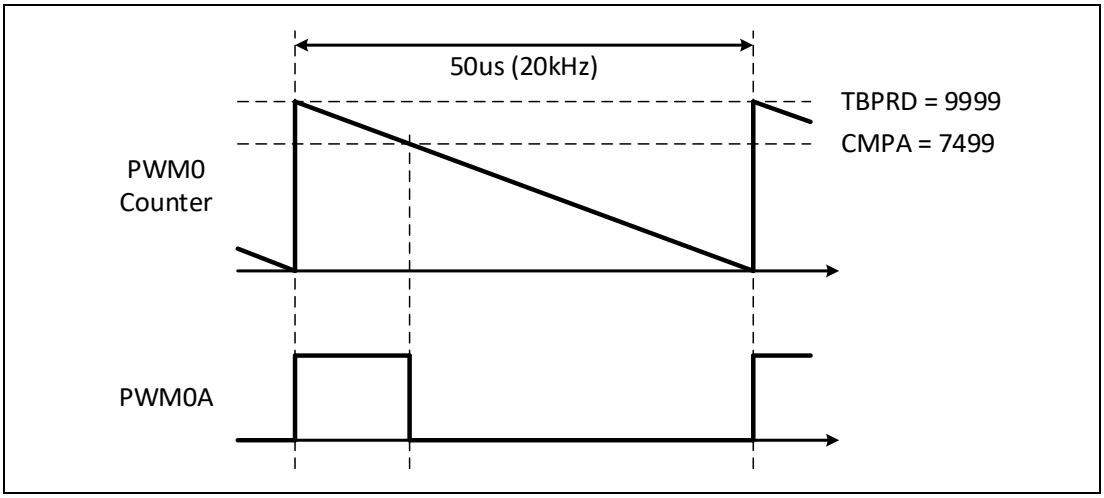
图 3-2: PWM 单通道向上计数独立波形示例



示例代码 3-3：配置 PWM0A 输出向下计数的 25%占空比的 20kHz 波形

```
void main()
{
  FLASH_WALLOW();
  FLASH_SetTiming(200000000);
  FLASH_WDIS();

  CLOCK_InitWithRCO(CLOCK_HCLK_200MHZ);

  Delay_Init();

  /*
   * Init the UART
   *
   * 1.Set the GPIO44/45 as UART FUNC
   *
   * 2.Enable the UART CLK
   *
   * 3.Set the rest para
   */
  GPIO_SetPinChannel(GPIO_34,GPIO34_UART_TXD);
  GPIO_SetPinChannel(GPIO_35,GPIO35_UART_RXD);
  CLOCK_EnableModule(UART_MODULE);
  UART_Init(UART,38400);

   /*** Step 2: PWM initial on PWM0B ***/
  /*** Period is set to 200000000/200000/2 = 5000 ***/
  PWM_SingleChannelInit(PWM0,PWM_CHA,20000);
  /*** Step 3: Set down-counting mode ***/
  PWM_SetCounterMode(PWM0, COUNT_DOWN);
  /*** Step 4: Set Period as 200000000/200000 - 1 = 9999 ***/
  PWM_SetPeriodValue(PWM0,9999);
  /*** Step 5: Start counting ***/
  PWM_RunCounter(PWM0);
  /*** Step 6: Configure the AQ ***/
  PWM_ActionQualifierCHA(PWM0, AQCTLA_ALL_ZRO_DO_NOTHING
                              |AQCTLA_ALL_CAU_DO_NOTHING
                              |AQCTLA_ALL_PRD_SET_HIGH
                              |AQCTLA_ALL_CAD_SET_LOW);
  /*** Step 7: Drive PWM0A output 25% Duty ***/
  PWM_SetCMPA(PWM0,7499);
  /*** Step 8: Select GPIO18 as PWM0A ***/
  GPIO_SetPinChannel(GPIO_18,GPIO18_PWM0A);
}
```

图 3-3:　PWM 单通道向下计数独立波形示例



3.3 设定双道路互补 PWM 波形

建议按照如下步骤设定 PWM 互补对：

（1） 务必先调用 CLOCK_InitWithRCO()或 CLOCK_Init()初始化时钟，否则 PWM 时钟可能有误；

（2） 调用 PWM_ComplementaryPairChannelInit()，初始化基本波形，设置死区时间；

（3） 默认配置 PWM 为上下计数的中央对称波形；

（4） 默认的 PWM 周期参数按照上下计数方式被配置为：

TBPRD = PWM_Clock_Freq / PWM_Freq/2

（5） 调用 PWM_RunCounter()计数使能；

（6） 预设向上计数过 CMPA 时，PWMxA 输出翻转为高；向下计数过 CMPA 时，PWMxA 输出翻转为低；

（7） 通过 CMPA 调整占空，CMPA 越大则占空越小；

（8） 将管脚配置由 GPIO 改为 PWM 输出，真正将 PWM 波形输出于管脚上。

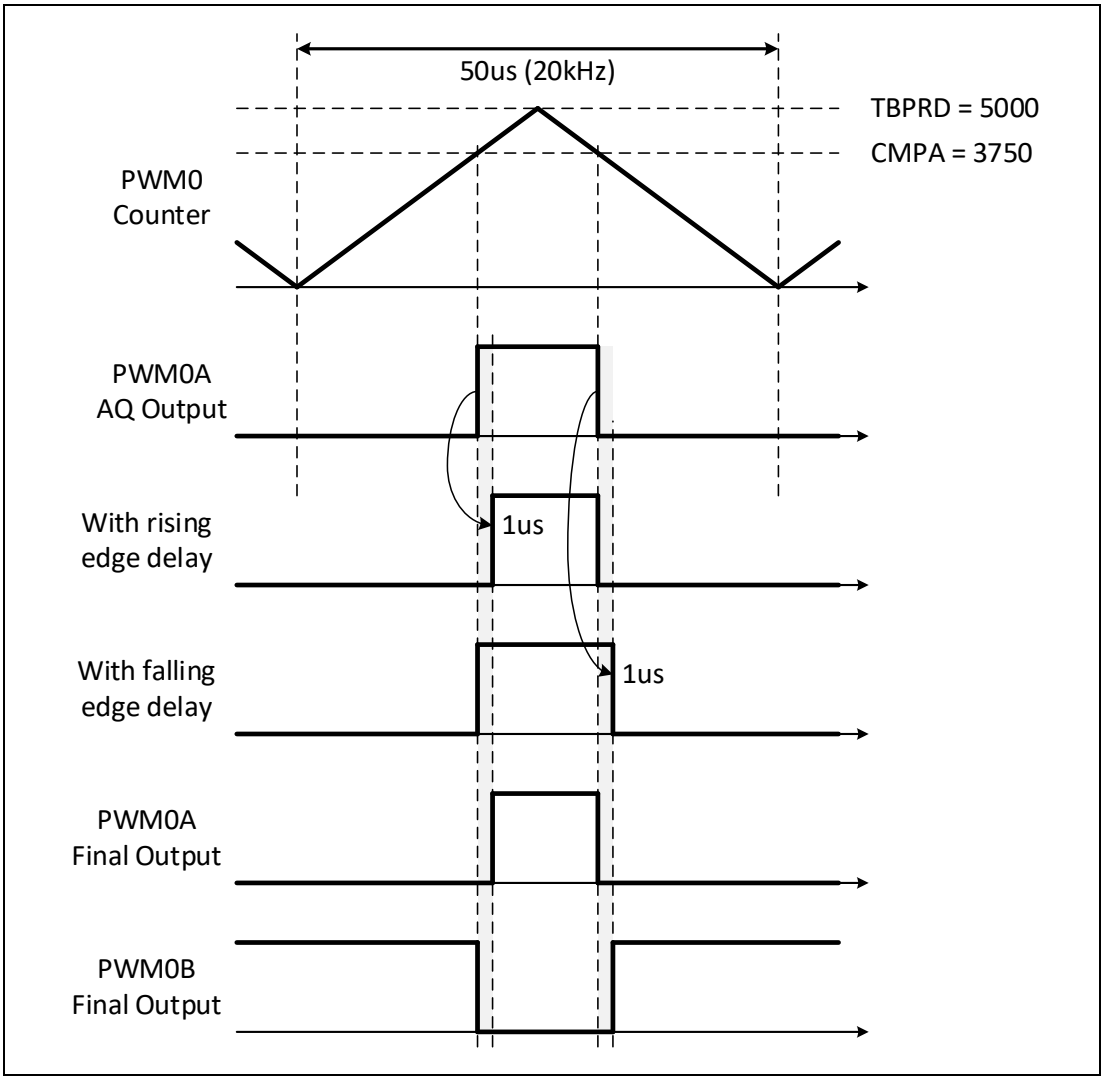
示例代码 3-4 和图 3-4 示范了如何产生一个 25%占空比、死区时间 1us 的 20kHz 互补波形。

示例代码 3-4：配置 PWM0 输出 25%占空比、死区时间 1us 的 20kHz 互补波形

```
void main()
{
  FLASH_WALLOW();
  FLASH_SetTiming(200000000);
  FLASH_WDIS();

  CLOCK_InitWithRCO(CLOCK_HCLK_200MHZ);

  Delay_Init();

  /*
   * Init the UART
   *
   * 1.Set the GPIO44/45 as UART FUNC
   *
   * 2.Enable the UART CLK
   *
   * 3.Set the rest para
   */
  GPIO_SetPinChannel(GPIO_34,GPIO34_UART_TXD);
  GPIO_SetPinChannel(GPIO_35,GPIO35_UART_RXD);
  CLOCK_EnableModule(UART_MODULE);
  UART_Init(UART,38400);

  /*** Step 2: PWM initial***/
  PWM_ComplementaryPairChannelInit(PWM0, 20000, 1000);
```

```
  /*** Step 3: Drive PWM0A output 25% Duty ***/
  PWM_SetCMPA(PWM0,3750);

  /*** Step 4: Start counting ***/
  PWM_RunCounter(PWM0);

  /*** Step 5: Set GPIO18 as PWM0A and GPIO19 as PWM0B ***/
  GPIO_SetPinChannel(GPIO_18,GPIO18_PWM0A);
  GPIO_SetPinChannel(GPIO_19,GPIO19_PWM0B);
}
```

图 3-4: PWM 双通道互补对波形示例



3.4 调整波形占空比

PWM 模块提供了影子寄存器（Shadow Register），便于在完成一个完整周期的波形后，修改参数以调整下一个周期的波形。

图 3-5: 更新 CMPA 调整波形占空比

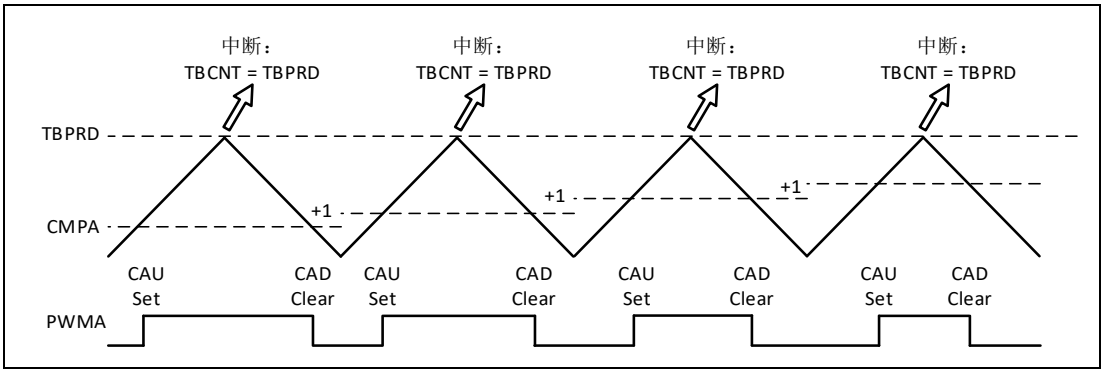


图 3-5 演示了如何通过在中断服务程序中更新影子寄存器来调整波形占空比，对应的代码见示例代码 3-5。

- 当 TBCNT=TBPRD 时，触发 CPU 中断；
- 中断服务代码中，更新 CMPA 影子寄存器的值；
- 当 TBCNT=0 时，完成一个周期的计数，此时根据 CMPA 的配置会从影子寄存器中更新 CMPA 的有效值；
- 新的周期开始后，更新后的 CMPA 使得占空比发生变化。

示例代码 3-5：配置 PWM0 输出 20kHz 单通道波形并控制占空比

```
void main()
{
  FLASH_WALLOW();
  FLASH_SetTiming(200000000);
  FLASH_WDIS();

  CLOCK_InitWithRCO(CLOCK_HCLK_200MHZ);

  Delay_Init();

  /*
   * Init the UART
   *
   * 1.Set the GPIO44/45 as UART FUNC
   *
   * 2.Enable the UART CLK
   *
   * 3.Set the rest para
   */
```

```
  GPIO_SetPinChannel(GPIO_34,GPIO34_UART_TXD);
  GPIO_SetPinChannel(GPIO_35,GPIO35_UART_RXD);
  CLOCK_EnableModule(UART_MODULE);
  UART_Init(UART,38400);

  /*** PWM initial on PWM0A ***/
  PWM_SingleChannelInit(PWM0,PWM_CHA,20000);
  /*** Load CMPA when TBCNT=ZERO ***/
  PWM_SetCMPALoadTiming(PWM0,ON_ZERO);
  /*** Run counter ***/
  PWM_RunCounter(PWM0);
  /*** Initialize 50% duty cycle ***/
  PWM_SetCMPA(PWM0,2500);
  /*** Set GPIO18 as PWM0A ***/
  GPIO_SetPinChannel(GPIO_18,GPIO18_PWM0A);
  /*** Generate interrupt whenever TBCNT=TBPRD ***/
  PWM_SetTimeEventTiming(PWM0,EQU_PERIOD);
  PWM_SetTimeEventPeriod(PWM0,ON_1ST_EVENT);
  PWM_EnableTimeEventInt(PWM0);
  /*** Enable PWM0 Interrupt in CPU side ***/
  NVIC_EnableIRQ(PWM0_IRQn);
  while(1) {
  }
}

void PWM0_IRQHandler(void)
{
  uint16_t u16CMPANextVal = PWM_GetCMPA(PWM0) + 1;
  /*** Update CMPA ***/
  if(u16CMPANextVal>PWM_GetPeriodValue(PWM1))
    PWM_SetCMPA(PWM0, 0);
  else
    PWM_SetCMPA(PWM0, u16CMPANextVal);
  /*** Clear interrupt flag ***/
  PWM_ClearTimeEventInt(PWM0);
}
```

3.5 多个 PWM 的计数同步

示例代码 3-6 中，PWM0 将 TBCNT=0 作为同步事件，送给 PWM1，并由后者再传给 PWM2。三个 PWM 均配置为 20kHz 上下计数模式，因此 PWM1 和 PWM2 在同步后的计数方向配置为向上。由于从 PWM 时钟到计数时钟的分频比为 1，考虑到同步信号传到计数器有一个 PWM 时钟延时，PWM1 和 PWM2 的同步复位值设置为 1 而不是 0。此外，分频比为 1 已经保证了 PWM 间的计数时钟保持同步，所以不需要额外调用 PWM_ForceClockSync()。

示例代码 3-6：PWM0 用 TBCNT=0 来同步 PWM1 和 PWM2，计数分频比=1

```
void main()
{
  uint32_t u32PWMPeriod;
  uint32_t u32ActureReLoadVal1;

  FLASH_WALLOW();
  FLASH_SetTiming(200000000);
  FLASH_WDIS();

  CLOCK_InitWithRCO(CLOCK_HCLK_200MHZ);

  Delay_Init();

  /*
   * Init the UART
   *
   * 1.Set the GPIO44/45 as UART FUNC
   *
   * 2.Enable the UART CLK
   *
   * 3.Set the rest para
   */
  GPIO_SetPinChannel(GPIO_34,GPIO34_UART_TXD);
  GPIO_SetPinChannel(GPIO_35,GPIO35_UART_RXD);
  CLOCK_EnableModule(UART_MODULE);
  UART_Init(UART,38400);

  /*** Step 2: PWM initial***/
  /*** Period is set to 200000000/200000/2 = 5000 ***/
  PWM_ComplementaryPairChannelInit(PWM0, 20000, 1000);
  PWM_ComplementaryPairChannelInit(PWM1, 20000, 1000);
  PWM_ComplementaryPairChannelInit(PWM2, 20000, 1000);

  /*** Step 3: set waveform 25% duty ***/
  u32PWMPeriod = ((CLOCK_GetModuleClk(PWM_MODULE))/ 20000)/2;
```

```
  PWM_SetCMPA(PWM0, (u32PWMPeriod * 3)/ 4);
  PWM_SetCMPA(PWM1, (u32PWMPeriod * 3)/ 4);
  PWM_SetCMPA(PWM2, (u32PWMPeriod * 3)/ 4);

  /*** Step 4: Start counting ***/
  PWM_RunCounter(PWM0);
  PWM_RunCounter(PWM1);
  PWM_RunCounter(PWM2);

  /*** Step 5: Configure hardware SYNC chain ***/
  /*** Step 5.1: PWM0 generate SYNC when TBCNT=0 ***/
  PWM_SetSyncOutEvent (PWM0,SYNCO_TBCNT_EQU_ZERO);

  /*** Step 5.2: PWM1 pass through the SYNC from PWM0 to PWM2 ***/
  PWM_SetSyncOutEvent (PWM1,SYNCO_SYNCI_AND_FRCSYNC);

  /*** Step 6: Configure PWM1 and PWM2 synchronization ***/
  PWM_EnableSync(PWM1);
  PWM_EnableSync(PWM2);

  /*** Step 7: set count direction after sync ***/
  PWM_SetCounterDirAfterSync(PWM1, COUNT_UP);
  PWM_SetCounterDirAfterSync(PWM2, COUNT_UP);

  /*** Step 8: calculate reload value after sync ***/
  u32ActureReLoadVal1 = PWM_CalculateSyncReloadValue
(u32PWMPeriod,
                                              COUNT_UP_DOWN,
                                             COUNT_UP,
                                              0,
                                              0);

  /*** Step 9: set reload value after sync ***/
  PWM_SetSyncReloadValue(PWM1, u32ActureReLoadVal1);
  PWM_SetSyncReloadValue(PWM2, u32ActureReLoadVal1);
  ...
}
```

示例代码 3-7 中，PWM0 将 TBCNT=CMPD 作为同步事件，送给 PWM1，并由后者再传给 PWM2。三个 PWM 均配置为向上计数模式，因此不需要配置同步后的计数方向。由于从 PWM 时钟到计数时钟的分频比大于 1，所以需要调用 PWM_ForceClockSync()来确保分频后的时钟保持同步，而且 PWM1 和 PWM2 的同步复位值保持和 CMPD 一致即可。

示例代码 3-7：PWM0 用 TBCNT=CMPD 来同步 PWM1 和 PWM2，计数分频比=2

```
void main()
{
  uint32_t u32PWMPeriod;
  uint32_t u32ActureReLoadVal1;

  FLASH_WALLOW();
  FLASH_SetTiming(200000000);
  FLASH_WDIS();

  CLOCK_InitWithRCO(CLOCK_HCLK_200MHZ);

  Delay_Init();

  /*
   * Init the UART
   *
   * 1.Set the GPIO44/45 as UART FUNC
   *
   * 2.Enable the UART CLK
   *
   * 3.Set the rest para
   */
  GPIO_SetPinChannel(GPIO_34,GPIO34_UART_TXD);
  GPIO_SetPinChannel(GPIO_35,GPIO35_UART_RXD);
  CLOCK_EnableModule(UART_MODULE);
  UART_Init(UART,38400);

  /*** Step 2: PWM initial***/
  PWM_ComplementaryPairChannelInit(PWM0, 20000, 1000);
  PWM_ComplementaryPairChannelInit(PWM1, 20000, 1000);
  PWM_ComplementaryPairChannelInit(PWM2, 20000, 1000);

  /*** Step 3: set waveform 25% duty ***/
  u32PWMPeriod = ((CLOCK_GetModuleClk(PWM_MODULE))/ 20000)/2;
  PWM_SetCMPA(PWM0, (u32PWMPeriod * 3)/ 4);
  PWM_SetCMPA(PWM1, (u32PWMPeriod * 3)/ 4);
  PWM_SetCMPA(PWM2, (u32PWMPeriod * 3)/ 4);

  /*** Setp 4: config count mode ***/
```

```
PWM_SetCounterMode(PWM0, COUNT_UP);
PWM_SetCounterMode(PWM1, COUNT_UP);
PWM_SetCounterMode(PWM2, COUNT_UP);

/*** setp 5: config waveform action when TBCNT=CMPB ***/
PWM_ActionQualifierCHA(PWM0, ZRO_SET_LOW
                           | PRD_DO_NOTHING
                           | CBU_SET_HIGH
                           | CBD_DO_NOTHING);

PWM_ActionQualifierCHA(PWM1, ZRO_SET_LOW
                           | PRD_DO_NOTHING
                           | CBU_SET_HIGH
                           | CBD_DO_NOTHING);

PWM_ActionQualifierCHA(PWM2, ZRO_SET_LOW
                           | PRD_DO_NOTHING
                           | CBU_SET_HIGH
                           | CBD_DO_NOTHING);

/*** Step 5: Configure and synchronize clock ***/
PWM_SetClockDiv(PWM0, PWM_CLKDIV0_2, PWM_CLKDIV1_1);
PWM_SetClockDiv(PWM1, PWM_CLKDIV0_2, PWM_CLKDIV1_1);
PWM_SetClockDiv(PWM2, PWM_CLKDIV0_2, PWM_CLKDIV1_1);
PWM_ForceClockSync();

/*** Step 6: Start counting ***/
PWM_RunCounter(PWM0);
PWM_RunCounter(PWM1);
PWM_RunCounter(PWM2);

/*** Step 7: Configure hardware SYNC chain ***/
/*** Step 7.1: PWM0 generate SYNC when TBCNT=CMPD ***/
PWM_SetSyncOutEvent(PWM0,SYNCO_TBCNT_EQU_CMPD);

/*** Step 7.2: PWM1 pass through the SYNC from PWM0 to PWM2 ***/
PWM_SetSyncOutEvent(PWM1,SYNCO_SYNCI_AND_FRCSYNC);

/*** Step 8: Configure PWM1 and PWM2 synchronization ***/
PWM_EnableSync(PWM1);
PWM_EnableSync(PWM2);

/*** Step 9: calculate reload value after sync ***/
u32ActureReLoadVal1 = PWM_CalculateSyncReloadValue(u32PWMPeriod,
                                                   COUNT_UP_DOWN,
```

```
                                                COUNT_UP,
                                                0,
                                                0);
  /*** Step 10: set reload value after sync ***/

  PWM_SetSyncReloadValue(PWM1, u32ActureReLoadVal1);
  PWM_SetSyncReloadValue(PWM2, u32ActureReLoadVal1);
  ...
}
```

示例代码 3-8 中，PWM0 将软件强制的信号作为同步事件，送给 PWM1，并由后者再传给 PWM2。这种情况下，只要使能三个 PWM 的同步功能并配置相同的复位值和计数方向即可，与分频比等设置无关。

示例代码 3-8：PWM0 用软件来同步 PWM1 和 PWM2

```
void main()
{
  uint32_t u32PWMPeriod;

  FLASH_WALLOW();
  FLASH_SetTiming(200000000);
  FLASH_WDIS();

  CLOCK_InitWithRCO(CLOCK_HCLK_200MHZ);

  Delay_Init();

  /*
   * Init the UART
   *
   * 1.Set the GPIO44/45 as UART FUNC
   *
   * 2.Enable the UART CLK
   *
   * 3.Set the rest para
   */
  GPIO_SetPinChannel(GPIO_34,GPIO34_UART_TXD);
  GPIO_SetPinChannel(GPIO_35,GPIO35_UART_RXD);
  CLOCK_EnableModule(UART_MODULE);
  UART_Init(UART,38400);

  /*** Step 2: PWM initial***/
  PWM_ComplementaryPairChannelInit(PWM0, 20000, 1000);
  PWM_ComplementaryPairChannelInit(PWM1, 20000, 1000);
  PWM_ComplementaryPairChannelInit(PWM2, 20000, 1000);
```

```
    /*** Step 3: set waveform 25% duty ***/
    u32PWMPeriod = ((CLOCK_GetModuleClk(PWM_MODULE))/ 20000)/2;
    PWM_SetCMPA(PWM0, (u32PWMPeriod * 3)/ 4);
    PWM_SetCMPA(PWM1, (u32PWMPeriod * 3)/ 4);
    PWM_SetCMPA(PWM2, (u32PWMPeriod * 3)/ 4);

    /*** Step 3: Start counting ***/
    PWM_RunCounter(PWM0);
    PWM_RunCounter(PWM1);
    PWM_RunCounter(PWM2);

    /*** Step 4: Configure hardware SYNC chain ***/
    /*** Step 4.1: PWM0 generate SYNC as software forced event ***/
    PWM_SetSyncOutEvent(PWM0,SYNCO_SYNCI_AND_FRCSYNC);

    /*** Step 4.2: PWM1 pass through the SYNC from PWM0 to PWM2 ***/
    PWM_SetSyncOutEvent(PWM1,SYNCO_SYNCI_AND_FRCSYNC);

    /*** Step 5: Configure PWM0, PWM1 and PWM2 synchronization ***/
    PWM_EnableSync(PWM0);
    PWM_EnableSync(PWM1);
    PWM_EnableSync(PWM2);
    PWM_SetSyncReloadValue(PWM0, 100);
    PWM_SetSyncReloadValue(PWM1, 100);
    PWM_SetSyncReloadValue(PWM2, 100);
    PWM_SetPostSyncCounterDir(PWM0, COUNT_DOWN);
    PWM_SetPostSyncCounterDir(PWM1, COUNT_DOWN);
    PWM_SetPostSyncCounterDir(PWM2, COUNT_DOWN);

    /*** Step 6: Assert an software SYNC on PWM0 ***/
    PWM_ForceSync(INC_PWM0);
    ...
}
```

采用全局同步信号可以实现最灵活的 PWM 同步，此时待同步的 PWM 间不需要存在硬件信号通路。如示例代码 3-9 所示，只要使能相关 PWM 的同步功能并配置相同的复位值和计数方向，就可以在全局信号到来时同步这些 PWM。

示例代码 3-9：用全局信号来同步 PWM0、PWM2 和 PWM4

```
void main()
{
  uint32_t u32PWMPeriod;

  FLASH_WALLOW();
  FLASH_SetTiming(200000000);
  FLASH_WDIS();

  CLOCK_InitWithRCO(CLOCK_HCLK_200MHZ);

  Delay_Init();

  /*
   * Init the UART
   *
   * 1.Set the GPIO44/45 as UART FUNC
   *
   * 2.Enable the UART CLK
   *
   * 3.Set the rest para
   */
  GPIO_SetPinChannel(GPIO_34,GPIO34_UART_TXD);
  GPIO_SetPinChannel(GPIO_35,GPIO35_UART_RXD);
  CLOCK_EnableModule(UART_MODULE);
  UART_Init(UART,38400);

  /*** Step 2: PWM initial***/
  PWM_ComplementaryPairChannelInit(PWM0, 20000, 1000);
  PWM_ComplementaryPairChannelInit(PWM2, 20000, 1000);
  PWM_ComplementaryPairChannelInit(PWM4, 20000, 1000);

  /* TBCNT SYNC out signal when TBCNT=0 must be close first */
  PWM_SetSyncOutEvent(PWM0, TBCTL_BIT_SYNCOSEL_DISABLE);
  PWM_SetSyncOutEvent(PWM2, TBCTL_BIT_SYNCOSEL_DISABLE);

  /*** set PWM 25% duty ***/
  u32PWMPeriod = ((CLOCK_GetModuleClk(PWM_MODULE))/ 20000)/2;
  PWM_SetCMPA(PWM0, (u32PWMPeriod * 3)/ 4);
  PWM_SetCMPA(PWM2, (u32PWMPeriod * 3)/ 4);
  PWM_SetCMPA(PWM4, (u32PWMPeriod * 3)/ 4);
```

```
  /*** Step 3: Start counting ***/
  PWM_RunCounter(PWM0);
  PWM_RunCounter(PWM2);
  PWM_RunCounter(PWM4);

  /*** Step 4: Configure PWM0, PWM1 and PWM2 synchronization ***/
  PWM_EnableSync(PWM0);
  PWM_EnableSync(PWM2);
  PWM_EnableSync(PWM4);
  PWM_SetSyncReloadValue(PWM0, 100);
  PWM_SetSyncReloadValue(PWM2, 100);
  PWM_SetSyncReloadValue(PWM4, 100);
  PWM_SetCounterDirAfterSync(PWM0, COUNT_UP);
  PWM_SetCounterDirAfterSync(PWM2, COUNT_UP);
  PWM_SetCounterDirAfterSync(PWM4, COUNT_UP);

  /*** Step 5: Enable synchronization on global event ***/
  /*** Example 5.1: Assert an software SYNC ***/
  PWM_ForceSync(INC_PWM0|INC_PWM2|INC_PWM4);

  /*** Example 5.2: Synchronize when GPIO8 is low ***/
  PWM_EnableSyncFromGPIO(INC_PWM0|INC_PWM2|INC_PWM4);
  PWM_SetSyncFromGPIO (GPIO_8,GPIO_LEVEL_LOW); /*Pin is
automatically*/
                                    /* set as GPIO8 input */

  /* Set timer1 generate SYNC to PWM when count down to 0 */
  TIMER_EnablePWMSync(TIMER0);
  TIMER_EnablePWMSync(TIMER1);
  TIMER_EnablePWMSync(TIMER2);

  /*** Example 5.3: Use SYNC signal from Timer 0 ***/
  PWM_EnableSyncFromTIMER0(INC_PWM0|INC_PWM2|INC_PWM4);

  /*** Example 5.4: Use SYNC signal from Timer 1 ***/
  PWM_EnableSyncFromTIMER1(INC_PWM0|INC_PWM2|INC_PWM4);

  /*** Example 5.5: Use SYNC signal from Timer 2 ***/
  PWM_EnableSyncFromTIMER2(INC_PWM0|INC_PWM2|INC_PWM4);
  ...
}
```

3.6 产生驱动电机的三相 PWM 波形

示例代码 3-10 产生了驱动电机的三相 PWM 波形，宏的使用令代码更具有可读性：

示例代码 3-10：产生驱动电机的三相 PWM 波形

```
#define PWM_U               PWM0
#define PWM_V               PWM1
#define PWM_W               PWM2
#define LINK_PWM_U          SEL_PWM0
#define GPIO_PWM_U_H        GPIO_34
#define GPIO_PWM_V_H        GPIO_24
#define GPIO_PWM_W_H        GPIO_26
#define GPIO_PWM_U_H_SEL    GPIO34_PWM0A
#define GPIO_PWM_V_H_SEL    GPIO24_PWM1A
#define GPIO_PWM_W_H_SEL    GPIO28_PWM2A
#define GPIO_PWM_U_L        GPIO_35
#define GPIO_PWM_V_L        GPIO_25
#define GPIO_PWM_W_L        GPIO_27
#define GPIO_PWM_U_L_SEL    GPIO35_PWM0B
#define GPIO_PWM_V_L_SEL    GPIO25_PWM1B
#define GPIO_PWM_W_L_SEL    GPIO27_PWM2B
void main()
{
  FLASH_WALLOW();
  FLASH_SetTiming(200000000);
  FLASH_WDIS();

  CLOCK_InitWithRCO(CLOCK_HCLK_200MHZ);

  Delay_Init();

  /*
   * Init the UART
   *
   * 1.Set the GPIO44/45 as UART FUNC
   *
   * 2.Enable the UART CLK
   *
   * 3.Set the rest para
   */
  GPIO_SetPinChannel(GPIO_34,GPIO34_UART_TXD);
  GPIO_SetPinChannel(GPIO_35,GPIO35_UART_RXD);
  CLOCK_EnableModule(UART_MODULE);
  UART_Init(UART,38400);
  /* Step 1: Initial basic complementary PWM */
```

```
  PWM_ComplementaryPairChannelInit(PWM_U, 20000, 1000);
  PWM_ComplementaryPairChannelInit(PWM_V, 20000, 1000);
  PWM_ComplementaryPairChannelInit(PWM_W, 20000, 1000);

  /* Step 2: PWM start running */
  PWM_RunCounter(PWM_U);
  PWM_RunCounter(PWM_V);
  PWM_RunCounter(PWM_V);

  /* Step 3: PWM set CMPA */
  PWM_LinkCMPA(PWM_V, LINK_PWM_U);
  PWM_LinkCMPA(PWM_W, LINK_PWM_U);
  PWM_SetCMPA(PWM_U, 2500);  /* Set CMPA=2500 for all the three */

  /* Step 4: Set GPIO pins to send out the PWM waveform */
  GPIO_SetPinChannel(GPIO_PWM_U_H,GPIO_PWM_U_H_SEL);
  GPIO_SetPinChannel(GPIO_PWM_V_H,GPIO_PWM_V_H_SEL);
  GPIO_SetPinChannel(GPIO_PWM_W_H,GPIO_PWM_W_H_SEL);
  GPIO_SetPinChannel(GPIO_PWM_U_L,GPIO_PWM_U_L_SEL);
  GPIO_SetPinChannel(GPIO_PWM_V_L,GPIO_PWM_V_L_SEL);
  GPIO_SetPinChannel(GPIO_PWM_W_L,GPIO_PWM_W_L_SEL);
  ...
}
```

3.7 外部管脚触发 PWM 封锁

若要使用某个外部管脚作为 PWM 的封锁信号 TZ0~TZ4，则先要将该管脚配置为 GPIO 通道，并把管脚方向配置为输入，然后再通过对应的 TZxSRCCTL 寄存器来设置所选的管脚号和触发电平。示例代码 3-11 通过调用 API 实现了对 TZ0~TZ4 的快速配置，并将 TZ0 设定为 PWM0 的一次性封锁信号，将 TZ1 和 TZ2 设置为 PWM0 的周期性封锁信号，最后设置当封锁发生时（同时包括一次性封锁和周期性封锁）停止输出。

用户在调用 PWM_SetCHAOutputWhenTrip 和 PWM_SetCHBOutputWhenTrip 时，需要注意**在参数中显性指明所有六种场景下的 PWM 输出**（哪怕选择 DO_NOTHING）。

示例代码 3-11：配置 TZ0~TZ4 用于封锁 PWM0

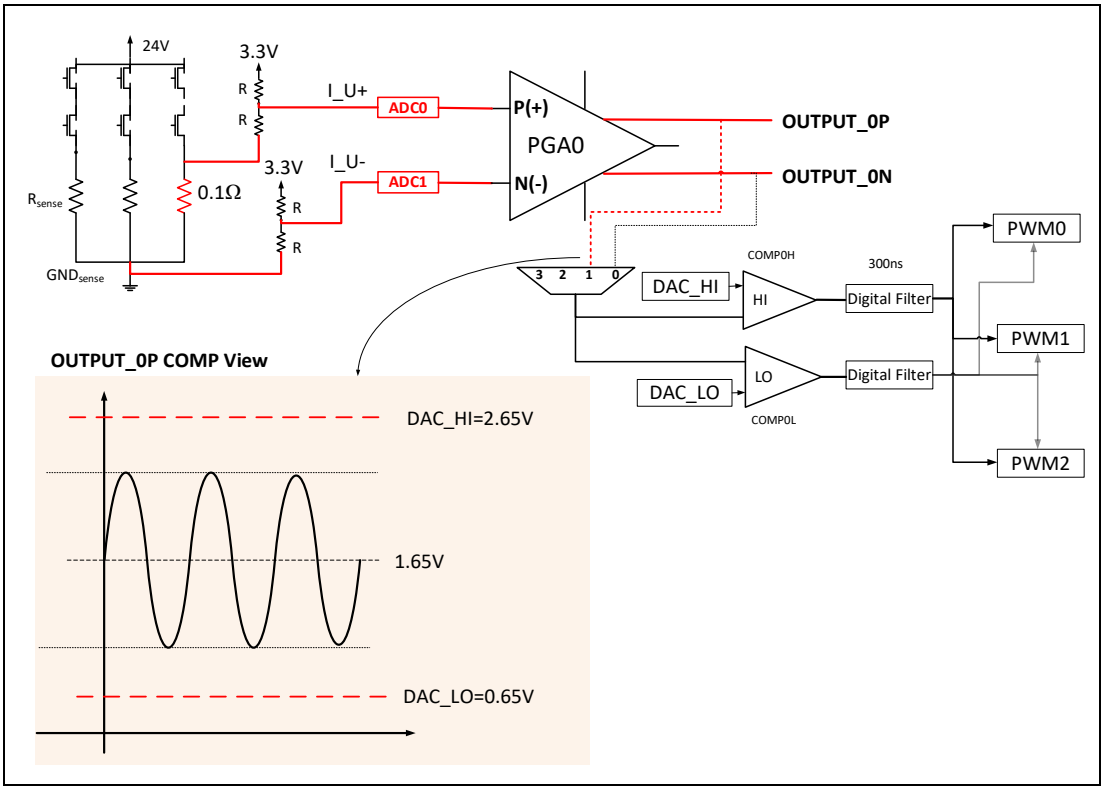
```
/*  Trigger TZ0 event when GPIO3 is high */
PWM_SetTZ0FromGPIO(GPIO_3,GPIO_LEVEL_HIGH);/* Pin GPIO3 is
automatically*/
                                    /* configured as GPIO input */
/*  Trigger TZ1 event when GPIO4 is high */
PWM_SetTZ1FromGPIO(GPIO_4,GPIO_LEVEL_HIGH);/* Pin GPIO4 is
automatically*/
                                    /* configured as GPIO input */
/*  Trigger TZ2 event when GPIO5 is high */
PWM_SetTZ2FromGPIO(GPIO_5,GPIO_LEVEL_HIGH);/* Pin GPIO5 is
automatically*/
                                    /* configured as GPIO input */
/*  Trigger TZ3 event when GPIO6 is low */
PWM_SetTZ3FromGPIO(GPIO_6,GPIO_LEVEL_LOW); /* Pin GPIO6 is
automatically*/
                                    /* configured as GPIO input */
/*  Trigger TZ4 event when GPIO7 is low */
PWM_SetTZ4FromGPIO(GPIO_7,GPIO_LEVEL_LOW); /* Pin GPIO7 is
automatically*/
                                    /* configured as GPIO input */
/* Set TZ0 as one-shot trip event */
PWM_SetOneShotTripEvent(PWM0, TRIP_TZ0, TZEVT_ASYNC_OR_LATCH);
/* Set TZ1 and TZ2 as CBC trip event */
PWM_SetCBCTripEvent(PWM0,TRIP_TZ1|TRIP_TZ2,TZEVT_ASYNC_OR_LATCH);
/* Set PWM0 output as tristate upon one-shot and CBC trip event */
/* Need to explicitly specify actions for all 6 trip scenarios */
PWM_SetCHAOutputWhenTrip(PWM0, TZU_TRIP_AS_TRI_STATE |
                               TZD_TRIP_AS_TRI_STATE |
                               DCEVT0U_TRIP_DO_NOTHING |
                               DCEVT0D_TRIP_DO_NOTHING |
                               DCEVT1U_TRIP_DO_NOTHING |
                               DCEVT1D_TRIP_DO_NOTHING);
PWM_SetCHBOutputWhenTrip(PWM0, TZU_TRIP_AS_TRI_STATE |
```

```
                                TZD_TRIP_AS_TRI_STATE |
                                DCEVT0U_TRIP_DO_NOTHING |
                                DCEVT0D_TRIP_DO_NOTHING |
                                DCEVT1U_TRIP_DO_NOTHING |
                                DCEVT1D_TRIP_DO_NOTHING);
```

3.8 电流防护触发 PWM 封锁

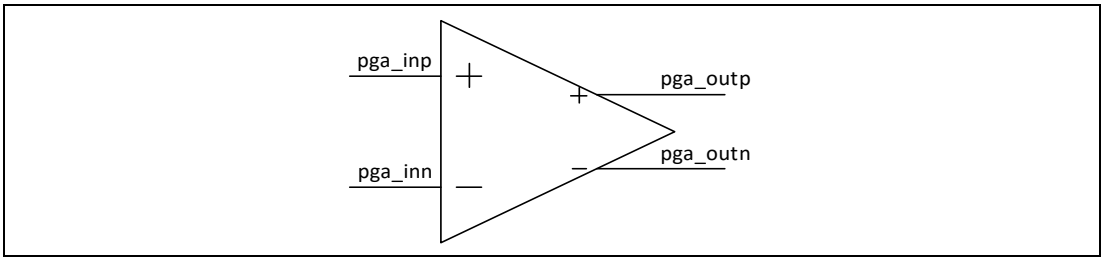
SPC11X8/SPD11X8 的比较器（COMP）专为相电流防护做特殊设计：当 PGA 作为三相电流采集的时候，比较器可以提供每一相独立的电流防护，比一般的总电流防护更加安全。而且任意一个 COMP 可以设置同时触发停止任意多个 PWM 的输出波形，这对于相电流防护来说是很重要的一个功能。

图 3-6: 基于比较器的相电流防护



三电阻采样的情况下，会使用到 PGA0、PGA1 和 PGA2，此时运放的输出是相对共模点上下对称的信号，一般选取它们的 P 端作为比较器的输入信号。

图 3-7: PGA 输入和输出



选择 INN 做为输出共模电压，输出电压和输入电压关系如下所示：

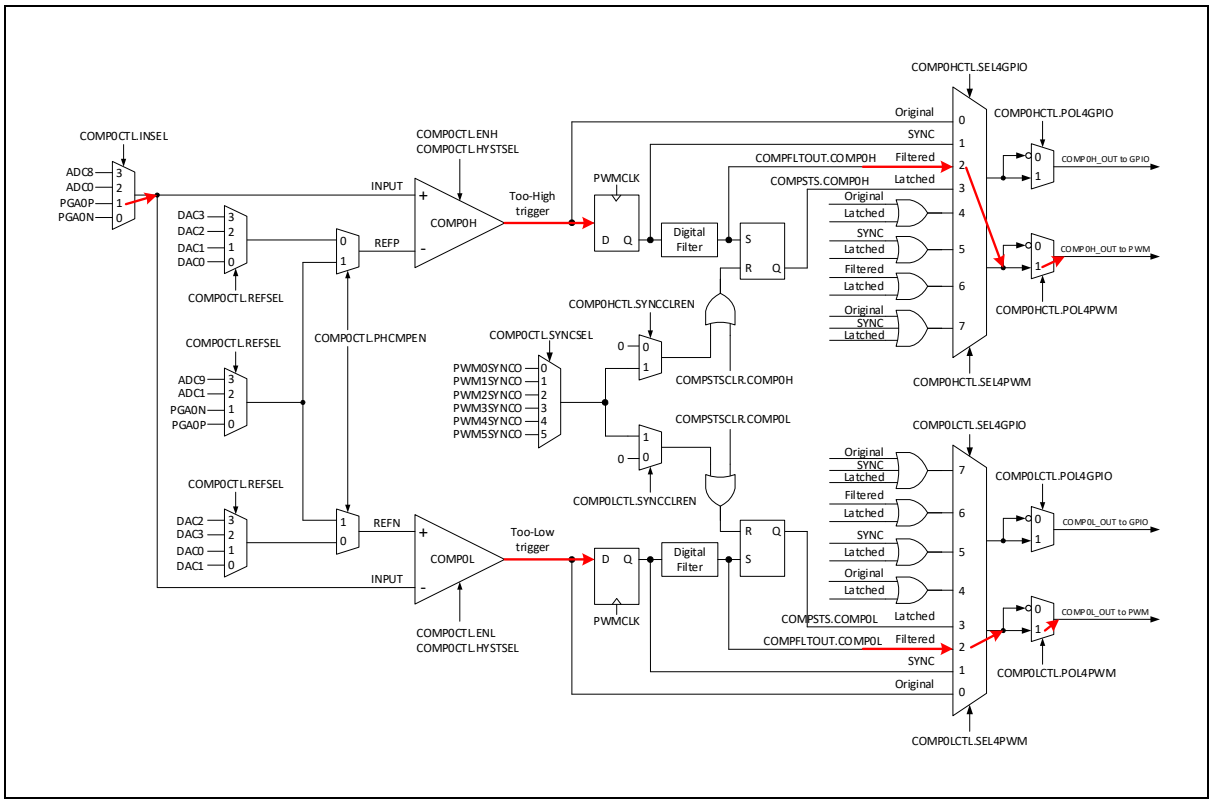
$$\mathrm{OUTP} = \mathrm{INN} + (INP - INN) \times \frac{Gain}{2}$$

$$\mathrm{OUTN} = \mathrm{INN} - (INP - INN) \times \frac{Gain}{2}$$

当 P 端信号（OUTP）高于/低于比较值的时候，可以分别让 COMPxH，COMPxL 动作，产生相应的 COMPxH 和 COMPxL 信号，这两个信号（如果是三电阻采样则是 6 个信号）后续会输入到数字比较（DC）模块中，做信号的逻辑或操作。

一般情况，会选择经过滤波器后的同步信号、不反向、作为比较器的输出。

图 3-8: 比较器输出配置



这里要注意的两个小点是：

- 比较器的输出信号可以分别输出到 GPIO 或者 PWM，且它们的极性是可以通过 POL4GPIO 和 POL4PWM 来分别配置；
- 比较器的原始信号应该是比较杂乱，其中带有很多干扰，一般情况下需要经过比较器自带的 Digital Filter 才能得到一个较为稳定的信号。

1） 数字比较（DC）子模块的配置

因为比较器的输出信号不能直接连接到 TZ 模块，但是比较器的信号可以先连接到 DC 模块，然后再由 DC 模块连接到 TZ 模块，所以需要对 DC 模块进行相应的配置。

其中，TZ 模块：

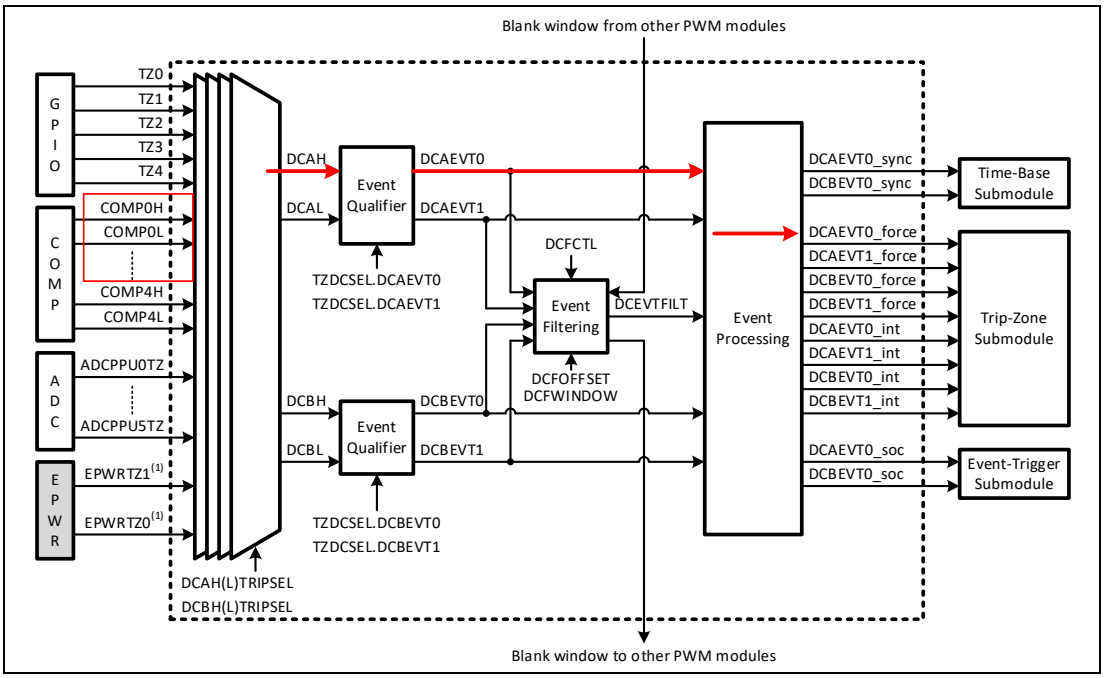
- CBC 子模块可以连接到 DCAEVT1 和 DCBEVT1，OST 子模块可以连接到 DCAEVT0 和 DCBEVT0，所以需要根据具体使用 CBC 或是 OST 连选择不同的 DC 子模块。
- DCAH、DCAL、DCBH、DCBL，因为它们从相同的输入源中选取信号，所以它们四个对于这个应用是等价的，可以任意选择一个来连接所有的比较器信号。比如选择 DCAH/DCAL，那么后续就要用到 DCAEVTx 的信号连接到 TZ，如果选择 DCBH/DCBL，那么就要用到 DCBEVTx 的信号连接到 TZ。

因为 CBC/OST 的配置是基本类似的，所以下面的主要说 OST 情况下的配置方法。简单起见，我们选择 DCAH 来做所有比较器信号的逻辑或操作：

（1） 设定 TZDCSEL.DCAEVT0=4 来选择 DCAH=1 触发原始 DCAEVT0 事件；

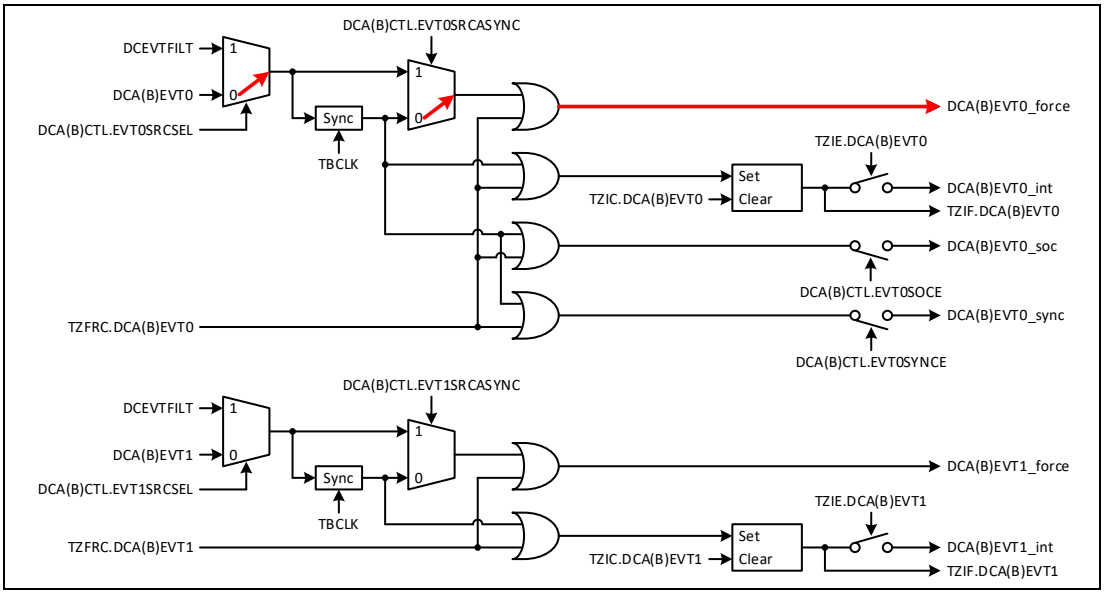
（2） 在 Event Processing 子模块中，最后选择未滤波的 DCAEVT0 的同步信号作为最终的 DCAEVT0_force 输出，这个信号最终会连入 TZ 子模块。

图 3-9: DCAEVT0 信号选择



（1） 在 SPC11X8/SPD11X8 中，EPWR 模块处于未连接状态，EPWRTZ0 和 EPWRTZ1 始终为 0。

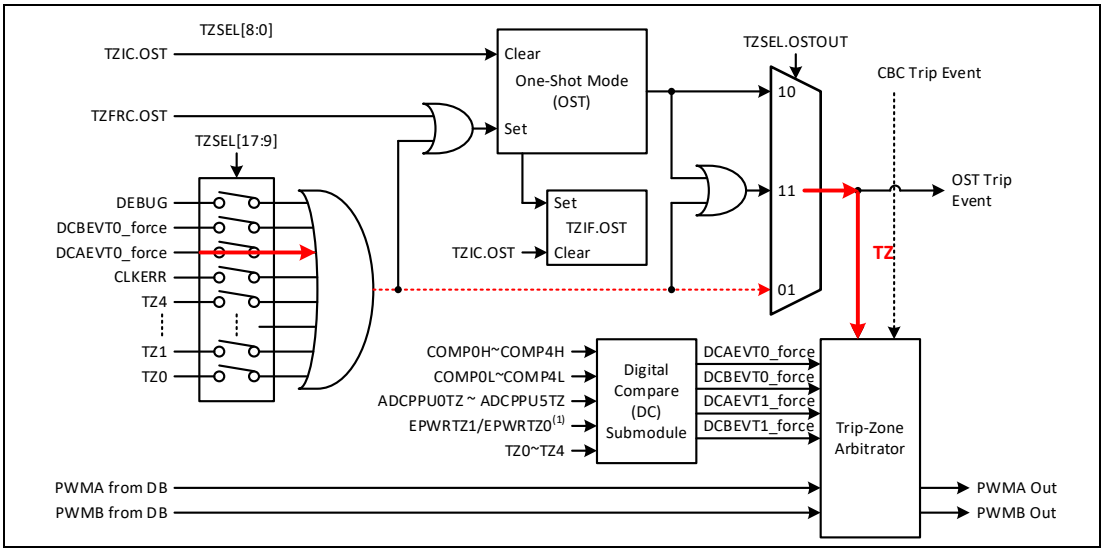
图 3-10: DCAEVT0_force 信号选择



2） 信号封锁（TZ）子模块的配置

对于 OST 关闭的配置，在其输入信号中，打开之前配置的 DCAEVT0_force 的通路，然后再设置 TZSEL.OSTOUT=3，选取 latch 信号作为最终的 TZ（OST）信号的输出。

图 3-11: TZ OST 设置



配置 TZACTL.TZAU = 1，TZACTL.TZAD = 1，TZBCTL.TZBU = 1，TZBCTL.TZBD = 1，这样当 TZ 信号有效时，PWMA out 和 PWMB out 都会被强制拉低到 0。

这里需要注意：

- TZA 和 TZB 实际是同一个信号，就是如上图所标识的 TZ，并不是两个不同的信号。TZ 信号可以是 OSTOUT，也可以是 CBCOUT。
- 当使用 TZ 信号进行过流保护信号封锁的时候，需要把 TZACTL/TZBCTL 的其他 bit-field 设置为 0x7，即这些信号不会对最后的 PWM 输出产生影响。

- 当 TZSEL.OSTOUT=1 时，这时候 OST 输出的信号实际是不带锁存功能的，所以看到的封锁行为会表现为当过流比较信号一旦消失，PWM 又会恢复，所以需要注意，实际情况中一定要选取 TZSEL.OSTOUT=2 或 3。

示例代码 3-12 假设用户使用 PGA0 到 PGA2 来放大三相电流采样电阻上的电压，设置 COMP0 到 COMP2 监控 PGA 的输出是否在 0.65V~2.65V 的范围内。比较的结果经过 300ns 除颤滤波后生成相应的 COMPxL/COMPxH 事件。这六个事件全被选中用于产生 DCAH 信号。当任意一个 COMPxL/COMPxH 事件发生时，DCAH 信号变高，触发原始 DCAEVT0 事件，然后通过盲区对可能的尖峰信号加以消隐，得到最终的 DCAEVT0 封锁信号。PWM 被设置为当出现 DCAEVT0 封锁时停止输出，从而实现相电流防护。

示例代码 3-12：设置比较器触发封锁实现三相电流防护

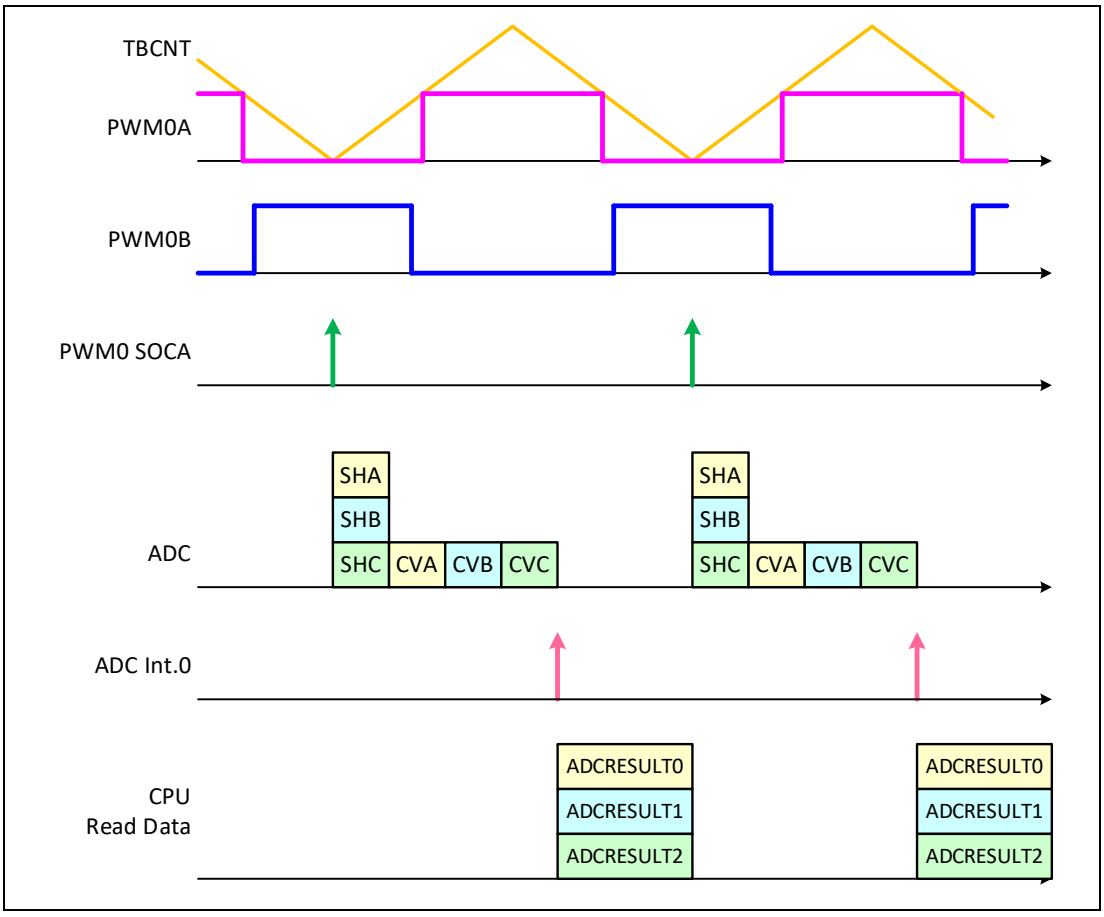
```
#define PWM_U         PWM0
#define PWM_V         PWM1
#define PWM_W         PWM2
#define INC_PWM_U     INC_PWM0
#define INC_PWM_V     INC_PWM1
#define INC_PWM_W     INC_PWM2
void MotorPWM_InitProtection(void)
{
  /* Note: Please initial PGA first */
  /* Initialize comparator with 300ns deglitch filtering window */
  COMP_Init(COMP_0_HI, COMP0_FROM_PGA0P_OUT, 2650, 300);
  COMP_Init(COMP_0_LO, COMP0_FROM_PGA0P_OUT,  650, 300);
  COMP_Init(COMP_1_HI, COMP1_FROM_PGA1P_OUT, 2650, 300);
  COMP_Init(COMP_1_LO, COMP1_FROM_PGA1P_OUT,  650, 300);
  COMP_Init(COMP_2_HI, COMP2_FROM_PGA2P_OUT, 2650, 300);
  COMP_Init(COMP_2_LO, COMP2_FROM_PGA2P_OUT,  650, 300);
  /* Affected by results monitored from all three phases */
  PWM_EnableDCAHTripEvent (PWM_U, DCTRIP_COMP0H |
                                  DCTRIP_COMP1H |
                                  DCTRIP_COMP2H |
                                  DCTRIP_COMP0L |
                                  DCTRIP_COMP1L |
                                  DCTRIP_COMP2L);
  /* Define raw DCAEVT0 as whenever COMP0~COMP2 event is asserted */
  PWM_SetRawDCAEvent0(PWM_U, DCAH_HIGH_DCAL_X);
  /* Configure DC event filter */
  /* Input is DCAEVT0, apply original polarity without filter*/
  PWM_U->DCACTL.all |= DCACTL_ALL_EVT0SRCSEL_DCAEVT0
                     | DCACTL_ALL_EVT0SRCASYNC_DCAEVT0_ASYNC;
  /* configure Tripzone input as DCAEVT0 */
  PWM_U->TZSEL.all = TZSEL_ALL_DCAEVT0_OST_ENABLE |
                     TZSEL_ALL_OSTOUT_ASYNC_OR_LATCH;
  /* Set output to set low upon Trip trip event */
```

```
  PWM_SetCHAOutputWhenTrip(PWM_U, TZU_TRIP_SET_LOW |
                                  TZD_TRIP_SET_LOW |
                                  DCEVT0U_TRIP_DO_NOTHING |
                                  DCEVT0D_TRIP_DO_NOTHING |
                                  DCEVT1U_TRIP_DO_NOTHING |
                                  DCEVT1D_TRIP_DO_NOTHING);
  PWM_SetCHBOutputWhenTrip(PWM_U, TZU_TRIP_SET_LOW |
                                  TZD_TRIP_SET_LOW |
                                  DCEVT0U_TRIP_DO_NOTHING |
                                  DCEVT0D_TRIP_DO_NOTHING |
                                  DCEVT1U_TRIP_DO_NOTHING |
                                  DCEVT1D_TRIP_DO_NOTHING);
  /* Add similar codes for PWM_V and PWM_W */
  ...
}
```

3.9 PWM 触发 ADC 功能

SPC11X8/SPD11X8 支持 ADC 同时对三相信号进行采样和依次转换，从而可以支持在算法中更好地重建反馈回来的电机相电流波形。图 3-12 给出了实际使用的方案示例。假设采用 PWM0~PWM2 对应于电机的三相控制，三者均输出双通道带死区的互补波形。当 PWM0 的计数过零时，触发 ADC SOC0，ADC 的三个采样保持模块（SHA~SHC）可以配置为对三相信号同时采样，然后依次转换。在完成所有三个信号的模数转换后，ADC 向 CPU 发送 ADCINT0 中断请求。用户可以在中断服务程序中通过 ADCRESULT0~ADCRESULT2 寄存器读取三路对应的电压值。

图 3-12: PWM 触发 ADC 三通道同时采样



示例代码 3-13 中，PWM0SOCA 信号触发 ADC 的 SOC0 事件，并配置该事件下同时使能 ADC 的三个采样保持模块。此时，SOC1 和 SOC2 的各项配置信息中，仅有通道选择有效。系统不会触发 SOC1 事件和 SOC2 事件，而是在 SOC0 发生时，根据 SOC0~SOC2 的配置，选择需要三个 GPIO 管脚上的信号进行采样和转换。对应的采样时间和转换时间在 SOC0 中配置。在完成转换后，发送 ADC0_IRQn 给 CPU。

示例代码 3-13：PWM 触发 ADC 同时采样三相信号并依次转换

```
void MotorPWM_TrigADCSOC()
{
  uint32_t u32PWMPeriod;

  FLASH_WALLOW();
  FLASH_SetTiming(200000000);
  FLASH_WDIS();

  CLOCK_InitWithRCO(CLOCK_HCLK_200MHZ);

  Delay_Init();

  /*
   * Init the UART
   *
   * 1.Set the GPIO44/45 as UART FUNC
   *
   * 2.Enable the UART CLK
   *
   * 3.Set the rest para
   */
  GPIO_SetPinChannel(GPIO_34,GPIO34_UART_TXD);
  GPIO_SetPinChannel(GPIO_35,GPIO35_UART_RXD);
  CLOCK_EnableModule(UART_MODULE);
  UART_Init(UART,38400);

  /* Step 1: Initial Basic Complementary PWM */
  PWM_ComplementaryPairChannelInit(PWM0, 20000, 1000);
  PWM_ComplementaryPairChannelInit(PWM1, 20000, 1000);
  PWM_ComplementaryPairChannelInit(PWM2, 20000, 1000);

  /* Drive PWM0A output 25% Duty */
  u32PWMPeriod = PWMPeriod(PWM_FREQ);
  PWM_SetCMPA(PWM0, (u32PWMPeriod * 3)/ 4);
  PWM_SetCMPA(PWM1, (u32PWMPeriod * 3)/ 4);
  PWM_SetCMPA(PWM2, (u32PWMPeriod * 3)/ 4);

  /* Step 2: PWM Trig Select */
```

```
  PWM_SetSOCATiming(PWM0, EQU_ZERO);
  PWM_SetSOCAPeriod(PWM0, ON_1ST_EVENT);
  PWM_EnableSOCA (PWM0);

  /* Step 3: ADC initial */
  ADC_EasyInit1(ADC_SOC_0, GPIO_3, ADCTRIG_PWM0SOCA);
  ADC_EasyInit1(ADC_SOC_1, GPIO_5, ADCTRIG_Software);
  ADC_EasyInit1(ADC_SOC_2, GPIO_7, ADCTRIG_Software);

  /* Step 4: Set simultaneous sampling */
  ADC_SetSOCSH(ADC_SOC_0, SHA_AND_SHB_SHC);

  /* Step 5. Interrupt service routine configuration */
  NVIC_SetPriority(ADC0_IRQn,1);
  NVIC_EnableIRQ(ADC0_IRQn);
}
```

4 修订记录

表 4-1: 文档修订记录

日期	版本	修改内容
2019-07-25	1	初始版本
2021-11-13	2	1. 更新图 2-11。 2. 更新图 2-13。 3. 更新章节 3.8。