SPC1169 PWM 使用指南

版本 A/0 – 2023 年 5 月

概述

PWM 在电力电子系统中有着至关重要的作用，广泛的应用在电机控制、开关电源及 UPS 中。SPC1169 中带有 4 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。

目录

图片列表

表格列表

版本历史

版本	日期	作者	状态	变更
A/0	2023 年 5 月 11 日	Hang Su	Released	首次发布。

术语或缩写

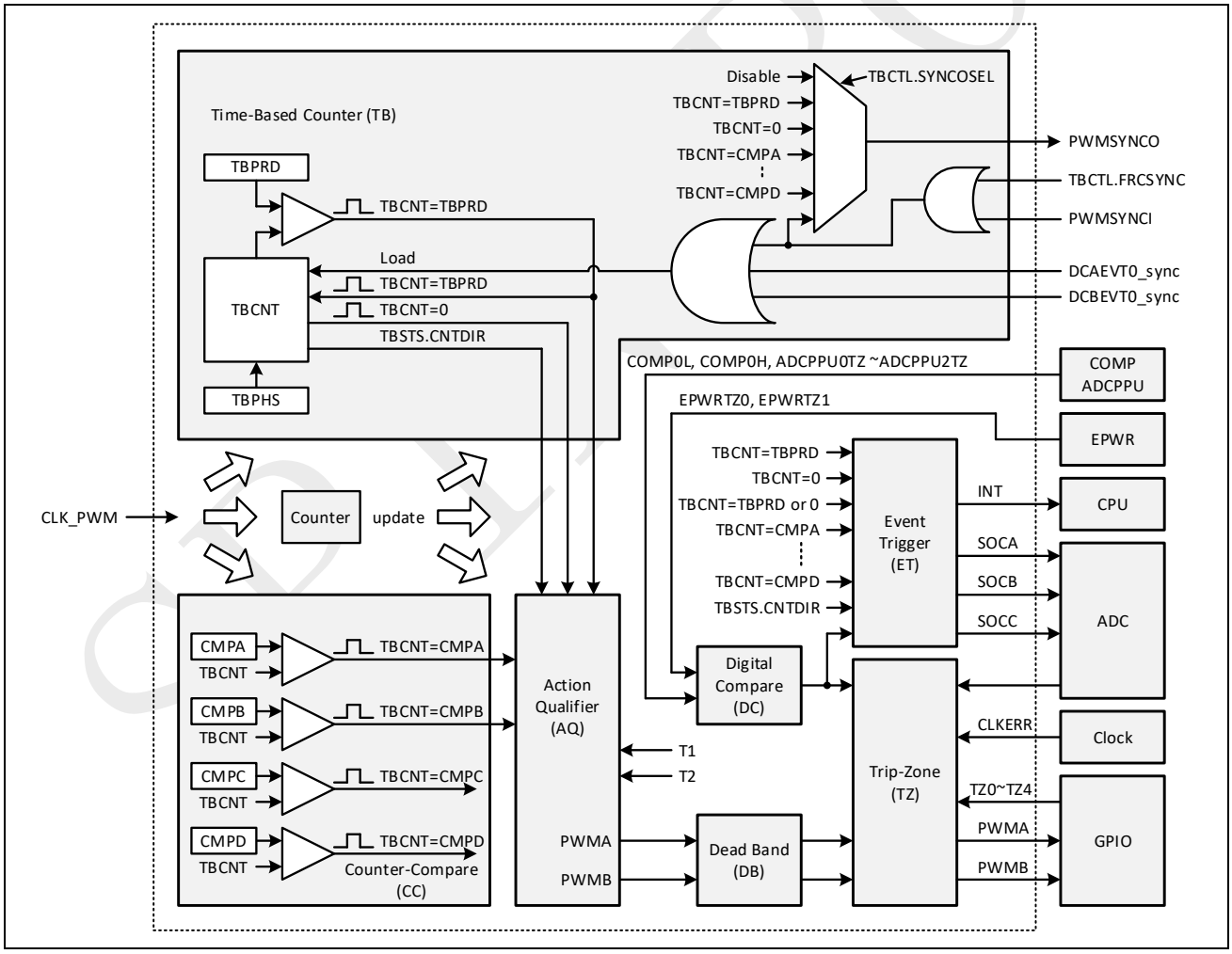
术语或缩写	描述

1 PWM 特性

SPC1169 的 PWM 单元有以下特点：

- 4 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 8 组 PWM 输出管脚；
- 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；
- 内部比较器 Comparator 可关断任意 PWM，只需简单配置。

图 1-1：PWM 单元功能框图

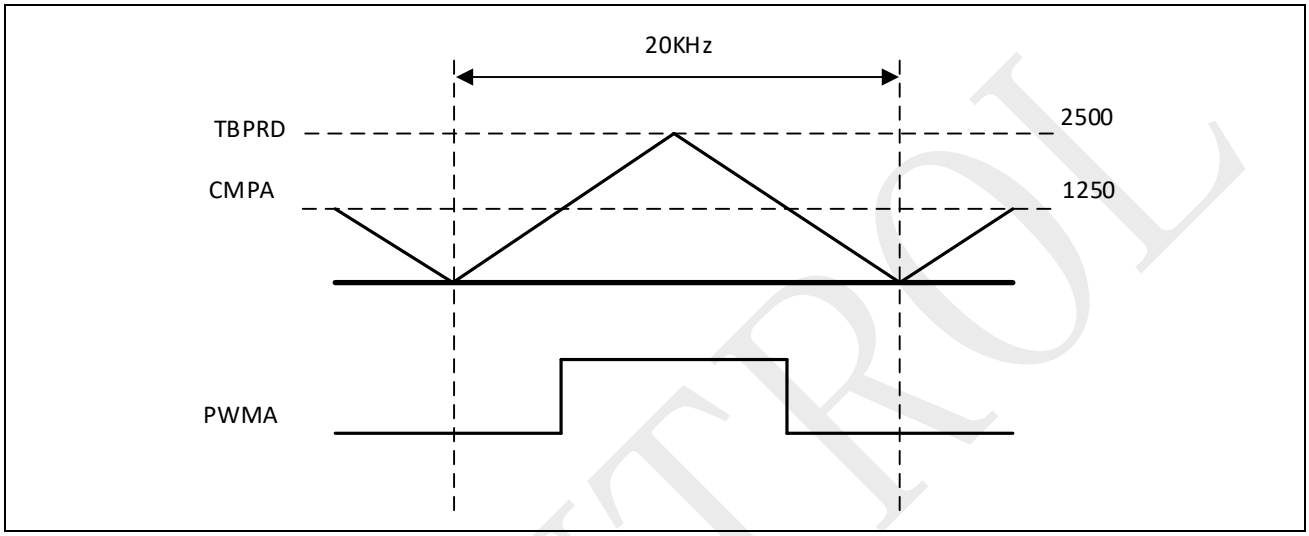


2 PWM 实例

2.1 设定单通道独立 PWM

如果有产生任意占空比单通道 PWM 的需求，如图 2-1 所示。

图 2-1：单通道独立 PWM



PWM_InitSingleChannel ()配置重点如下：

- PWM 波形预设为中央对称型，中央对称型 PWM counter 之周期计算如下：

  Period = PWM_Clock_Freq / PWM_Freq/2

  以上述为例，PWM 的 Clock 与 HCLK 同频，因此为 100MHz，设定的 PWM 载波率为 20KHz，因此周期为 50us。故此时 Period = 2500；
- 预设当 counter 在往上数遇到 CMP 时，将波形拉高，往下数遇到 CMP 时，将波形拉低；
- 务必先执行 CLOCK_InitWithRCO()或是 CLOCK_Init()，否则 PWM clock 可能会有错误；
- 本函数不配置 PWM 波形的 dead time，用户若有控制需求，可参考死区配置示例。

Example Code

```
int main(void)
{
    CLOCK_InitWithRCO(CLOCK_CPU_100MHZ);

    Delay_Init();

    /*
     * Init the UART
     */
    PIN_SetChannel(PIN_GPIO10, PIN_GPIO10_UART0_TXD);
    PIN_SetChannel(PIN_GPIO11, PIN_GPIO11_UART0_RXD);
    UART_Init(UART0, 38400);

    /* Init PWM1 and output 20kHz waveform on channel A */
    PWM_InitSingleChannel(PWM1, PWM_CHA, PWM_Frequency);
```

```
    /* Select PIN_GPIO12 as the channel A output of PWM1 */
    PIN_SetChannel(PIN_GPIO12, PIN_GPIO12_PWM1A);

    /* Start PWM1 */
    PWM_RunCounter(PWM1);

    while (1)
    {

    }
}
```

2.2 产生两路互补、带死区 PWM 波

在电力电子、开关电源以及电机控制中，往往要应对半桥电路、全桥电路；这种电路的一个基本控制方法，就是用互补的 PWM 波去分别驱动半桥、全桥的上下桥臂开关管，同时互补的 PWM 波还需要带有一定的死区时间。

PWM 单元非常灵活，可以覆盖到半桥、全桥中的各种应用，这里先讲解一个简单的半桥带有死区的 PWM 波。下图是 PWM 波单元中的死区控制部分，可以看出它本身是具有复杂灵活的死区配置方式。对于半桥控制中的死区应用，一般是通过将一个 PWM 波增加死区时间后，取反得到带死区的互补 PWM 波。

如图 2-2：死区控制单元框图和图 2-3：生成带死区互补 PWM 示意图，展示了如何从一路 PWM 波，最终称为一对带死区互补 PWM 波的过程。

图 2-2：死区控制单元框图

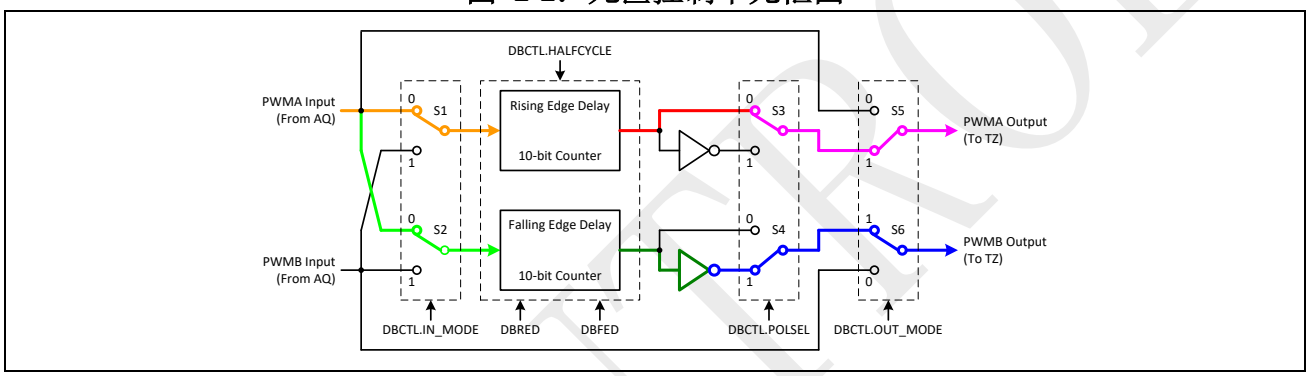
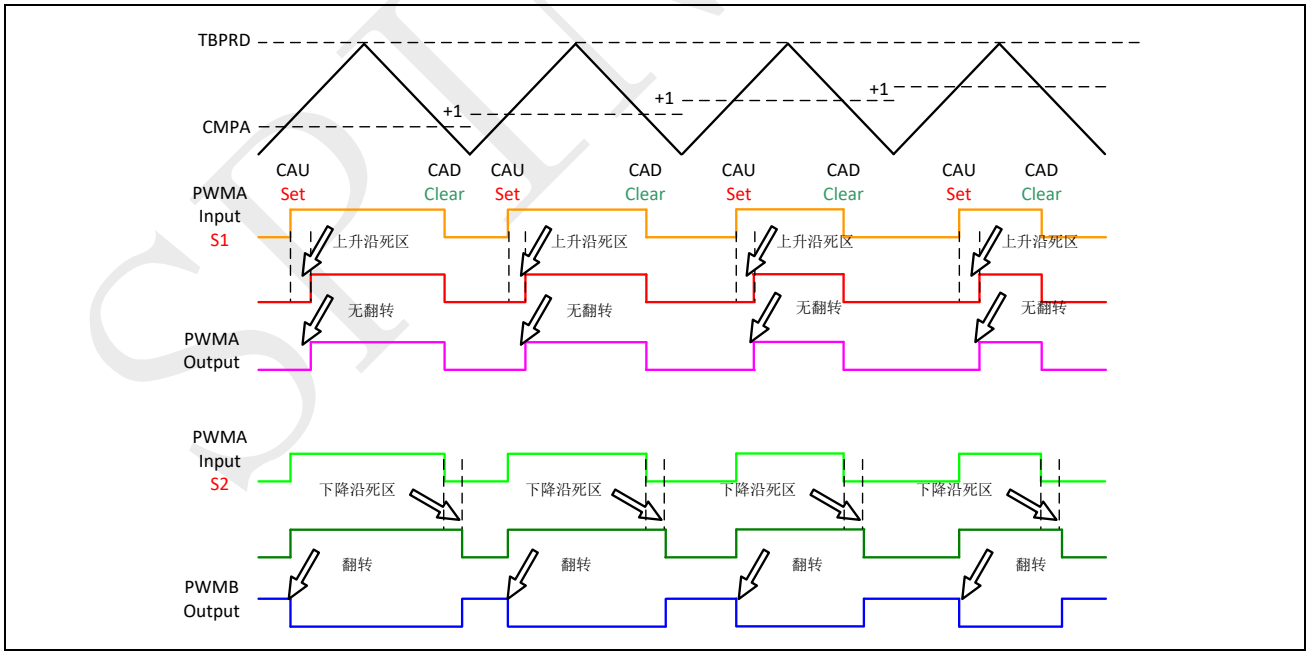


图 2-3：生成带死区互补 PWM 示意图



PWM_ComplementaryPairChannelInit()配置重点如下：

- PWM 波形预设为中央对称型，中央对称型 PWM counter 之周期计算如下：

  Period = PWM_Clock_Freq / PWM_Freq/2

以上述为例，PWM 的 Clock 与 HCLK 同频，因此为 100MHz，设定的 PWM 载波频率为 20KHz，因此周期为 50us。故此时 Period = 2500；

- 预设当 counter 在往上数遇到 CMPA 时，将波形拉高，往下数遇到 CMPA 时，将波形拉低。**注意：当以此种波形配置时，CMPA 越高，代表最终输出的 PWM 波形 Duty 越低，反之亦然；**
- CMPB 的配置在本配置中不影响波形输出；
- 务必先执行 CLOCK_InitWithRCO()或是 CLOCK_Init()，否则 PWM clock 可能会有错误。

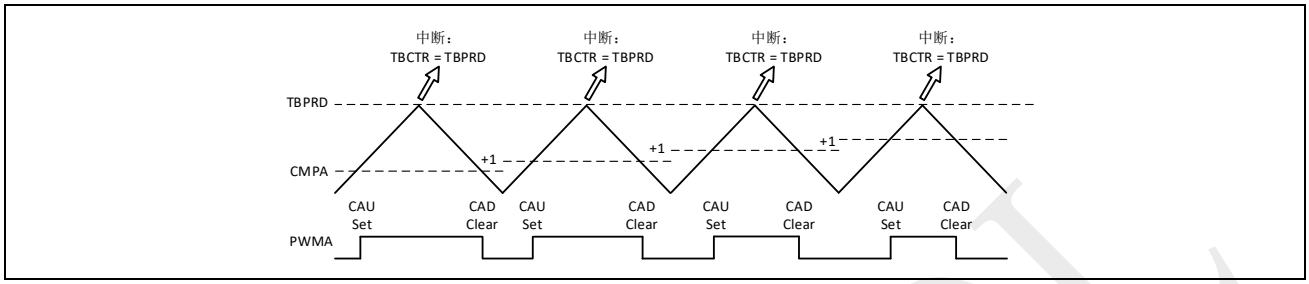
Example Code

```
int main(void)
{
    CLOCK_InitWithRCO(CLOCK_CPU_100MHZ);

    Delay_Init();

    /*
     * Init the UART
     */
    PIN_SetChannel(PIN_GPIO10, PIN_GPIO10_UART0_TXD);
    PIN_SetChannel(PIN_GPIO11, PIN_GPIO11_UART0_RXD);
    UART_Init(UART0, 38400);

    /*
    * 1. Init PWM1 and output 20kHz waveform on channel A
    *
    * 2. Add 1ns dead band at falling and rising edge
    *
    * 3. Channel B output the channel A waveform which added DB
    */
    PWM_InitComplementaryPairChannel(PWM1, PWM_FREQ, PWM_DB_NS);

    /* Set PWM0A output 75% duty waveform */
    u32PWMPeriod = PWMPeriod(PWM_FREQ);
    PWM_SetCMPA(PWM1, u32PWMPeriod / 4);

    /* Select PIN_GPIO12/PIN_GPIO13 as the channel A/B output of PWM1
respectively */
    PIN_SetChannel(PIN_GPIO12, PIN_GPIO12_PWM1A);
    PIN_SetChannel(PIN_GPIO13, PIN_GPIO13_PWM1B);

    /* Start PWM1 */
    PWM_RunCounter(PWM1);

    while (1)
    {

    }
}
```

2.3 设定周期性变化的 PWM 波

如果有产生任意占空比单通道 PWM 的需求，如图 2-4 所示。

图 2-4：周期性变化 PWM



PWM_InitSingleChannel ()配置重点如下：

- 配置 CMPA 到 CMPAA 的时机；
- 配置 PWM 中断；
- 务必先执行 CLOCK_InitWithRCO()或是 CLOCK_Init()，否则 PWM clock 可能会有错误；
- 本函数不配置 PWM 波形的 dead time，用户若有控制需求，可参考死区配置示例。

Example Code

```
int main(void)
{
    CLOCK_InitWithRCO(CLOCK_CPU_100MHZ);

    Delay_Init();

    /*
     * Init the UART
     */
    PIN_SetChannel(PIN_GPIO10, PIN_GPIO10_UART0_TXD);
    PIN_SetChannel(PIN_GPIO11, PIN_GPIO11_UART0_RXD);
    UART_Init(UART0, 38400);

    /* Init PWM1 and output 20kHz waveform on channel A */
    PWM_InitSingleChannel(PWM1, PWM_CHA, PWM_Frequency);

    /* Set PWM1A output 50% duty waveform */
    u32PWMPeriod = PWMPeriod(PWM_Frequency);
    PWM_SetCMPA(PWM1, u32PWMPeriod / 2);

    /* Select PIN_GPIO12 as the channel A output of PWM1 */
    PIN_SetChannel(PIN_GPIO12, PIN_GPIO12_PWM1A);

    /* Set CMPA load time */
    PWM_SetCMPALoadTiming(PWM1, CMPCTL_LOAD_ON_ZERO);

    /* Enable PWM1 TBCTR = TBPRD event */
    PWM_SetTimeEventIntTiming(PWM1, EQU_PERIOD);
    PWM_SetTimeEventIntPeriod(PWM1, ON_1ST_EVENT);
    PWM_EnableTimeEventInt(PWM1);

    /* Enable PWM1 Interrupt in CPU side */
```

```
    NVIC_EnableIRQ(PWM1_IRQn);

    /* Start PWM1 */
    PWM_RunCounter(PWM1);

    while (1)
    {

    }
}

void PWM1_IRQHandler(void)
{
    uint16_t u16CMPAVal = 0;

    u16CMPAVal = PWM1->CMPA;

    /* Change PWM duty */
    if((u16CMPAVal +1) >= PWM1->TBPRD)
    {
        PWM_SetCMPA(PWM1,0);
    }
    else
    {
        PWM_SetCMPA(PWM1, u16CMPAVal + 1);
    }

    /* Clear interrupt flag */
    PWM_ClearTimeEventInt(PWM1);
}
```

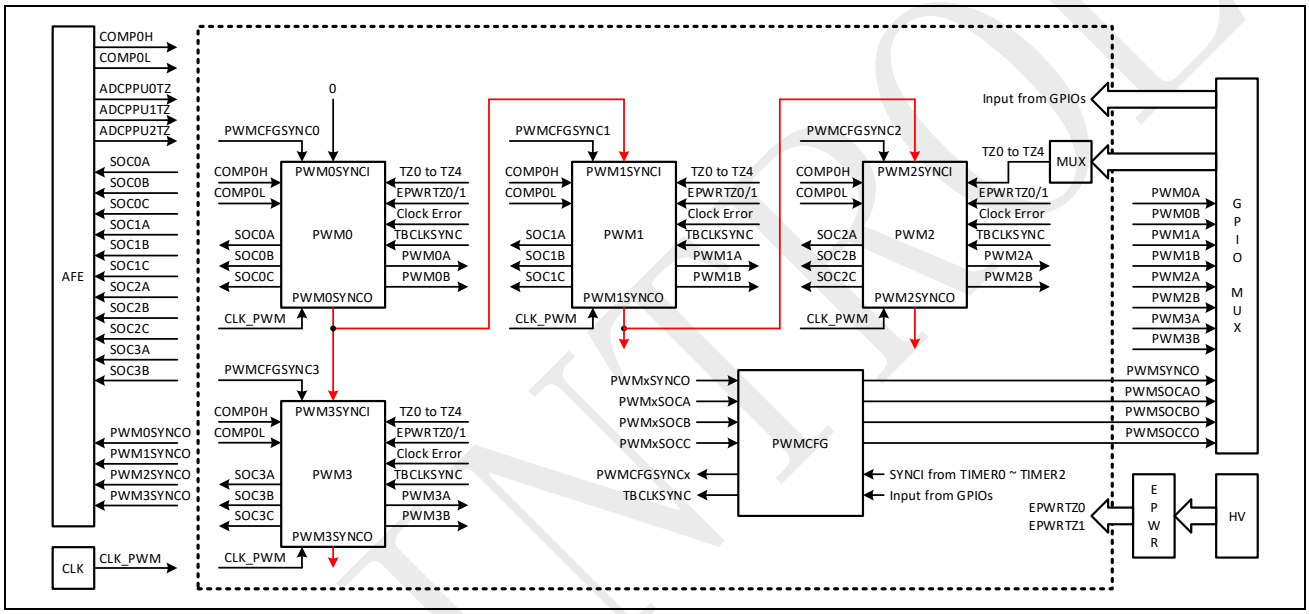
2.4 PWM 同步

在使用 PWM 过程中，有时需要在不同 PWM 间产生一固定相位差，这时可以用 PWM 级联同步或非级联同步实现。

2.4.1 级联方式

级联同步指的是同步信号在不同 PWM 模块中依次传导，SPC1169 可以建立级联同步关系的链路为 PWM0->PWM1->PWM2 或 PWM0->PWM1，如图 2-5 所示。

图 2-5：级联关系



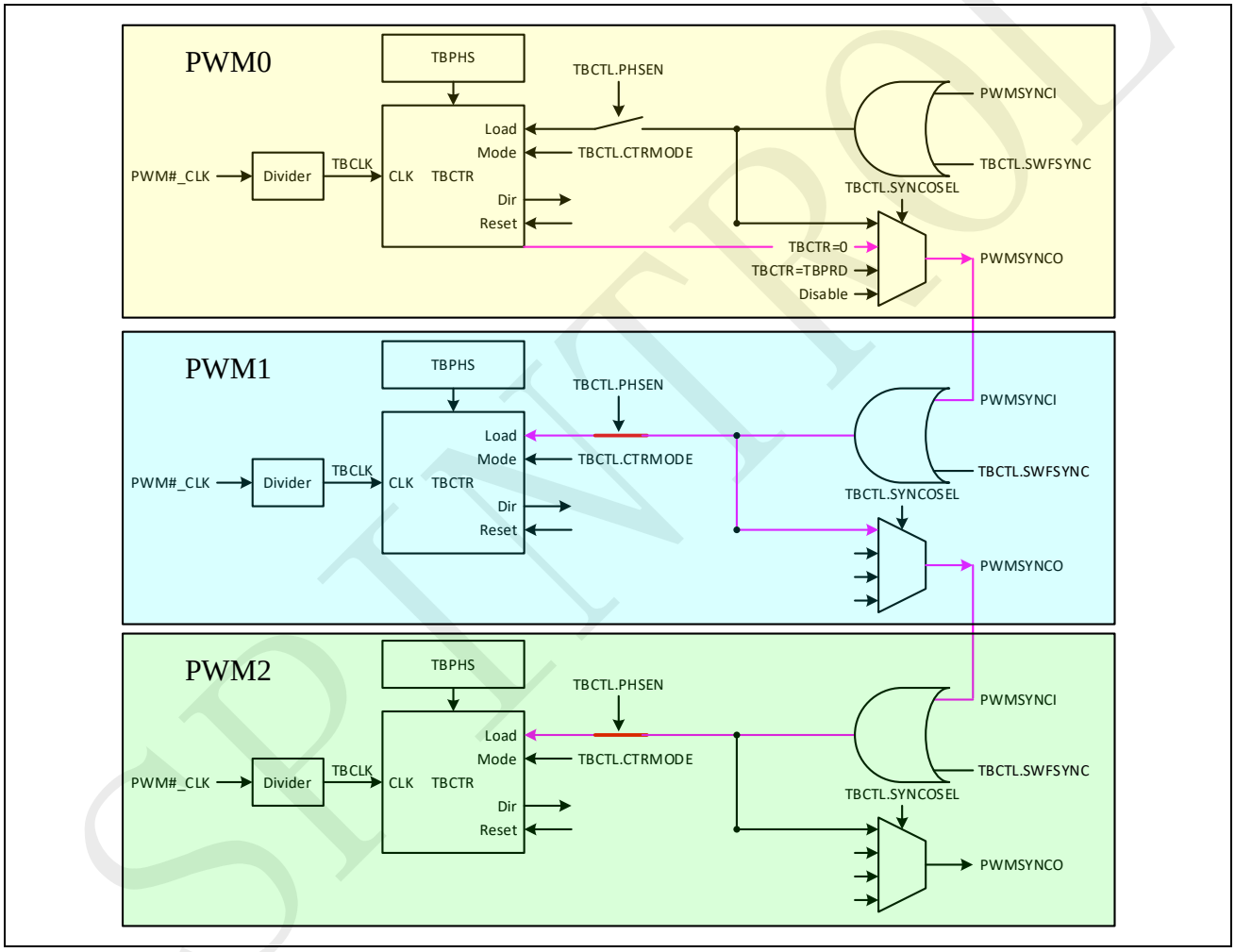
注意：PWMSYNCI 除了来自 PWMSYNCO，还可以来自 Timer 或 GPIO。

示例代码演示了级联同步，如图 2-6 所示：

- PWM0 使能 TBCTR=0 信号通过 PWMSYNCO 的输出；
- PWM0 PWMSYNCO 输出通过 PWM1 PWMSYNCI 改变 PWM1 相位；
- 由于 PWM1 使能 PWMSYNCI 信号通过 PWMSYNCO 的输出，PWM1 PWMSYNCO 输出通过 PWM2 PWMSYNCI 改变 PWM2 相位；

经过如上配置，将会在每个 PWM0 TBCTR=0 的时刻，同步 PWM0, PWM1, PWM2 之间的相位差。

图 2-6：级联式同步



代码实现如下所示：

Example Code

```
int main(void)
{
    CLOCK_InitWithRCO(CLOCK_CPU_100MHZ);

    Delay_Init();

    /*
     * Init the UART
     */
    PIN_SetChannel(PIN_GPIO10, PIN_GPIO10_UART0_TXD);
    PIN_SetChannel(PIN_GPIO11, PIN_GPIO11_UART0_RXD);
    UART_Init(UART0, 38400);

    #if defined(SPC1169)
    /* HV init */
    eErrorState = HV_Init(&u16PREDRIID);
    if (eErrorState == ERROR)
    {
        printf("Init HV mode FAIL\n");
        return 0;
    }
    else
    {
        printf("Init HV mode SUCCESS[ID:%d]\n", u16PREDRIID);
    }

    /* HV parameter write enable */
    eErrorState = EPWR_WriteRegister(HV_REG_CTLKEY, KEY_USER_REG);
    if (eErrorState == ERROR)
    {
        printf("Write CTLKEY register FAIL\n");
        return 0;
    }

    /* Power up PRE-DRIVER mode*/
    eErrorState = EPWR_WriteRegisterField(HV_REG_PDRVCTL0, PDRVCTL0_EN_Msk,
PDRVCTL0_EN_ENABLE);
    if (eErrorState == ERROR)
    {
        printf("Power up PRE-DRIVER FAIL\n");
        return 0;
    }

    /* Wait for PRE-DRIVER mode become enable */
    while ((u16PREDRIDATA & PMUSTS_CPRDY_READY) == 0)
    {
        eErrorState = EPWR_ReadRegister(HV_REG_PMUSTS, &u16PREDRIDATA);
        if (eErrorState == ERROR)
        {
            printf("PRE-DRIVER enable FAIL[%02x]\n", u16PREDRIDATA);
            return 0;
        }
    }
    #else
    /* Cofig PWM0/1/2 GPIO Function */
    PIN_SetChannel(PIN_GPIO19, PIN_GPIO19_PWM2B);
    PIN_SetChannel(PIN_GPIO20, PIN_GPIO20_PWM2A);
    PIN_SetChannel(PIN_GPIO21, PIN_GPIO21_PWM1B);
```

```
    PIN_SetChannel(PIN_GPIO22, PIN_GPIO22_PWM1A);
    PIN_SetChannel(PIN_GPIO23, PIN_GPIO23_PWM0B);
    PIN_SetChannel(PIN_GPIO24, PIN_GPIO24_PWM0A);
    #endif

    /* Init PWM0/PWM1/PWM2 and output 20kHz waveform on both channel A and
channel B */
    PWM_InitComplementaryPairChannel(PWM0, PWM_FREQ, PWM_DB_NS);
    PWM_InitComplementaryPairChannel(PWM1, PWM_FREQ, PWM_DB_NS);
    PWM_InitComplementaryPairChannel(PWM2, PWM_FREQ, PWM_DB_NS);
    PWM2->AQCTLA = AQCTLA_CAU_SET_LOW | AQCTLA_CAD_SET_HIGH;

    /* set the GPIO to observe the SYNC signal*/
    #if defined(SPC1169)
    PIN_SetChannel(PIN_GPIO18, PIN_GPIO18_PWMSYNCO);
    #else
    PIN_SetChannel(PIN_GPIO31, PIN_GPIO31_PWMSYNCO);
    #endif
    PWMCFG->SYNCOCTL = (PWMCFG->SYNCOCTL & (~SYNCOCTL_SYNCO0EN_Msk)) |
SYNCOCTL_SYNCO0EN_ENABLE;
    PWMCFG->SYNCOCTL = (PWMCFG->SYNCOCTL & (~SYNCOCTL_DURATION_Msk)) |
SYNCOCTL_DURATION_32_PWM_CLK;

    /* Set PWM0A output duty */
    u32PWMPeriod = PWMPeriod(PWM_FREQ);
    PWM_SetCMPA(PWM0, (u32PWMPeriod * 2) / 3);
    PWM_SetCMPA(PWM1, (u32PWMPeriod * 2) / 3);
    PWM_SetCMPA(PWM2, (u32PWMPeriod * 1) / 3);

    /* Select PWMSYNCO signal */
    PWM_SetSyncOutEvent(PWM0, SYNCO_DISABLE);
    PWM_SetSyncOutEvent(PWM1, SYNCO_SYNCI_AND_FRCSYNC);
    PWM_SetSyncOutEvent(PWM2, SYNCO_SYNCI_AND_FRCSYNC);

    /* Change PWMSYNCO signal of PWM0 */
    PWM_SetSyncOutEvent(PWM0, SYNCO_TBCNT_EQU_ZERO);

    /* Start counting */
    PWM_RunCounter(PWM0);
    PWM_RunCounter(PWM1);
    PWM_RunCounter(PWM2);

    /* Set PWMx counting down after SYNC */
    PWM_SetCounterDirAfterSync(PWM0, COUNT_UP);
    PWM_SetCounterDirAfterSync(PWM1, COUNT_UP);
    PWM_SetCounterDirAfterSync(PWM2, COUNT_UP);

    /*
    * There is a delay before SYNC signal reach PWM1 and PWM2,
    * so we must re-calculate the value for PWM1 and PWM2.
    */
    u32ActualReLoadVal1 = PWM_CalculateSyncReloadValue(u32PWMPeriod,
                        COUNT_UP_DOWN,
                        COUNT_UP,
                        PWM1->CMPA,
                        1);


    u32ActualReLoadVal2 = PWM_CalculateSyncReloadValue(u32PWMPeriod,
                        COUNT_UP_DOWN,
                        COUNT_UP,
                        (PWM2->CMPA),
                        1);
```

```
    /* Set the re-load value when SYNC signal happened */
    PWM_SetSyncReloadValue(PWM1, u32ActualReLoadVal1);
    PWM_SetSyncReloadValue(PWM2, u32ActualReLoadVal2);

    while (1)
    {

    }
}
```

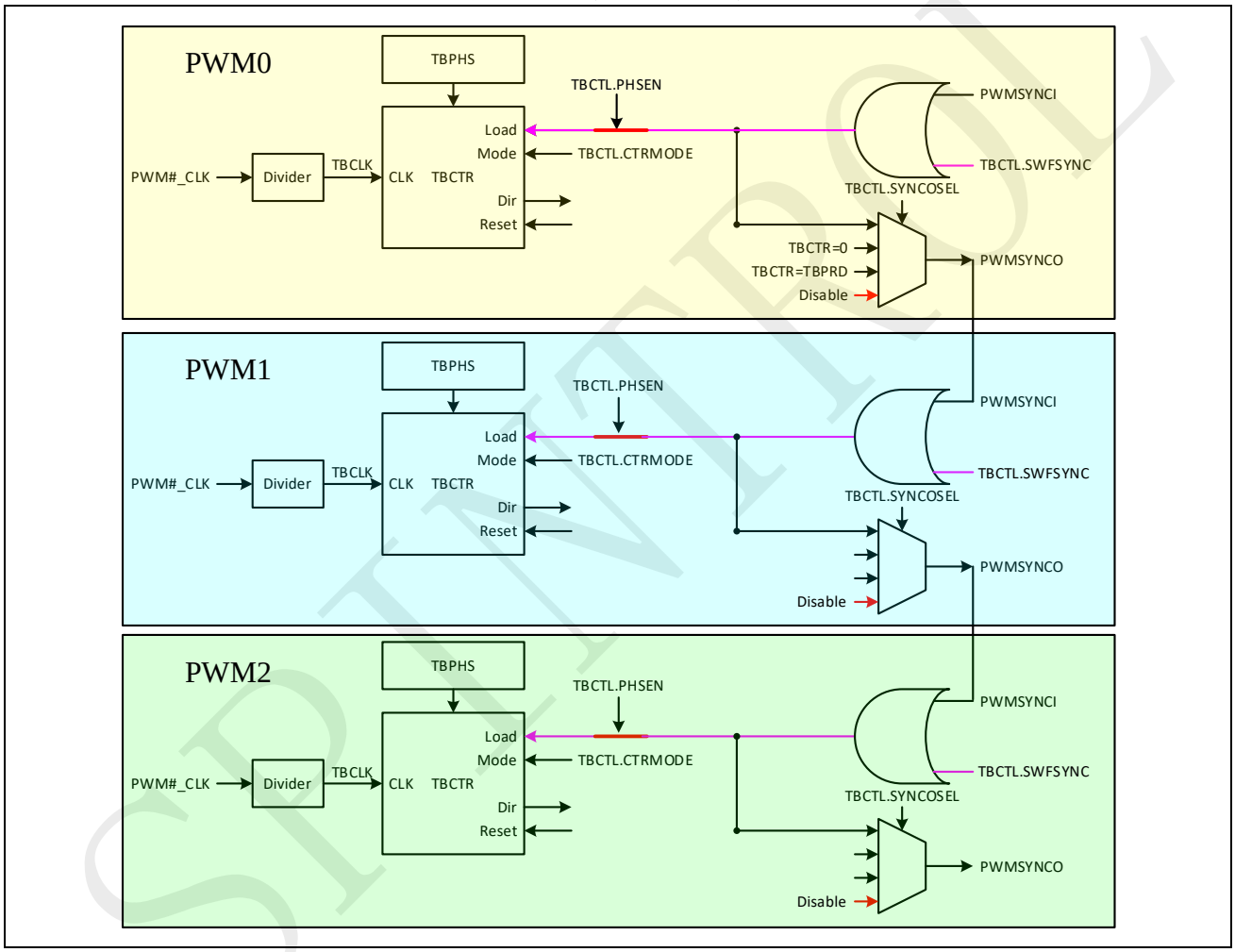
2.4.2 非级联方式

非级联同步指的是同步信号同时加在不同 PWM 模块上，其信号链路如图 2-7：非级联式同步中红色信号线路所示。

示例代码配置动作可描述为：

- 关闭各 PWM 模块的 PWMSYNCO 功能；
- 采用软件同时改变 PWM0, PWM1, PWM2 的相位。

图 2-7：非级联式同步



代码实现如下所示：

Example Code

```
int main(void)
{
    CLOCK_InitWithRCO(CLOCK_CPU_100MHZ);

    Delay_Init();

    /*
     * Init the UART
     */
```

```
    PIN_SetChannel(PIN_GPIO10, PIN_GPIO10_UART0_TXD);
    PIN_SetChannel(PIN_GPIO11, PIN_GPIO11_UART0_RXD);
    UART_Init(UART0, 38400);

    #if defined(SPC1169)
    /* HV init */
    eErrorState = HV_Init(&u16PREDRIID);
    if (eErrorState == ERROR)
    {
        printf("Init HV mode FAIL\n");
        return 0;
    }
    else
    {
        printf("Init HV mode SUCCESS[ID:%d]\n", u16PREDRIID);
    }

    /* HV parameter write enable */
    eErrorState = EPWR_WriteRegister(HV_REG_CTLKEY, KEY_USER_REG);
    if (eErrorState == ERROR)
    {
        printf("Write CTLKEY register FAIL\n");
        return 0;
    }

    /* Power up PRE-DRIVER mode*/
    eErrorState = EPWR_WriteRegisterField(HV_REG_PDRVCTL0, PDRVCTL0_EN_Msk,
PDRVCTL0_EN_ENABLE);
    if (eErrorState == ERROR)
    {
        printf("Power up PRE-DRIVER FAIL\n");
        return 0;
    }

    /* Wait for PRE-DRIVER mode become enable */
    while ((u16PREDRIDATA & PMUSTS_CPRDY_READY) == 0)
    {
        eErrorState = EPWR_ReadRegister(HV_REG_PMUSTS, &u16PREDRIDATA);
        if (eErrorState == ERROR)
        {
            printf("PRE-DRIVER enable FAIL[%02x]\n", u16PREDRIDATA);
            return 0;
        }
    }
    #else
    /* Cofig PWM0/1/2 GPIO Function */
    PIN_SetChannel(PIN_GPIO19, PIN_GPIO19_PWM2B);
    PIN_SetChannel(PIN_GPIO20, PIN_GPIO20_PWM2A);
    PIN_SetChannel(PIN_GPIO21, PIN_GPIO21_PWM1B);
    PIN_SetChannel(PIN_GPIO22, PIN_GPIO22_PWM1A);
    PIN_SetChannel(PIN_GPIO23, PIN_GPIO23_PWM0B);
    PIN_SetChannel(PIN_GPIO24, PIN_GPIO24_PWM0A);
    #endif

    /* Init PWM0/PWM1/PWM2 and output 20kHz waveform on both channel A and
channel B */
    PWM_InitComplementaryPairChannel(PWM0, PWM_FREQ, PWM_DB_NS);
    PWM_InitComplementaryPairChannel(PWM1, PWM_FREQ, PWM_DB_NS);
    PWM_InitComplementaryPairChannel(PWM2, PWM_FREQ, PWM_DB_NS);
    PWM2->AQCTLA = AQCTLA_CAU_SET_LOW | AQCTLA_CAD_SET_HIGH;

    /* Can only receive force signal of PWM0, PWM1, PWM2 */
    PWM_SetSyncOutEvent(PWM0, SYNCO_DISABLE);
```

```
    PWM_SetSyncOutEvent(PWM1, SYNCO_DISABLE);
    PWM_SetSyncOutEvent(PWM2, SYNCO_DISABLE);

    /* Set PWM0A output duty */
    u32PWMPeriod = PWMPeriod(PWM_FREQ);
    PWM_SetCMPA(PWM0, (u32PWMPeriod * 2) / 3);
    PWM_SetCMPA(PWM1, (u32PWMPeriod * 2) / 3);
    PWM_SetCMPA(PWM2, (u32PWMPeriod * 1) / 3);

    /* Start counting */
    PWM_RunCounter(PWM0);
    PWM_RunCounter(PWM1);
    PWM_RunCounter(PWM2);

    /* Set PWMx counting up after SYNC */
    PWM_SetCounterDirAfterSync(PWM0, COUNT_UP);
    PWM_SetCounterDirAfterSync(PWM1, COUNT_UP);
    PWM_SetCounterDirAfterSync(PWM2, COUNT_UP);

    /*
     * Set the value will be loaded to TBCNT after SYNC signal.
     */
    PWM_SetSyncReloadValue(PWM0, (PWM0->CMPA));
    PWM_SetSyncReloadValue(PWM1, 0);
    PWM_SetSyncReloadValue(PWM2, (PWM2->CMPA));


    /* Force a software SYNC signal in PWM0 PWM1 PWM2 */
    PWM_ForceSync(INC_PWM0 | INC_PWM1 | INC_PWM2);

    while (1)
    {

    }
}
```

2.4.3 观测同步输出信号

可以将 PWM 同步输出信号通过具有 PWMSOCO 功能的引脚送出，输出给示波器观测。

Example Code

```
    /* set the GPIO to observe the SYNC signal*/
    #if defined(SPC1169)
    PIN_SetChannel(PIN_GPIO18, PIN_GPIO18_PWMSYNCO);
    #else
    PIN_SetChannel(PIN_GPIO31, PIN_GPIO31_PWMSYNCO);
    #endif
    PWMCFG->SYNCOCTL = (PWMCFG->SYNCOCTL & (~SYNCOCTL_SYNCO0EN_Msk)) |
SYNCOCTL_SYNCO0EN_ENABLE;
    PWMCFG->SYNCOCTL = (PWMCFG->SYNCOCTL & (~SYNCOCTL_DURATION_Msk)) |
SYNCOCTL_DURATION_32_PWM_CLK;
```

2.5 PWM Trip Zone

在很多应用场景里头，存在类似紧急停车信号，当这个信号来到时候，需要立即或者安全的停止一切动作。如果把 PWM 的输出看成这种动作或者动作的驱动信号，同样有着立即或者安全停止的需求。

- 按周期(Cycle-by-Cycle, CBC)封锁 PWM 输出，会在下一个 PWM 周期重新启动（适用于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）；
- 一次性封锁(One Shot, OST)PWM 输出，One-shot 则是会直接停止 PWM 输出，直到使用者介入，清除 one-shot flag 之后才重新输出波形。

图 2-8：周期封锁

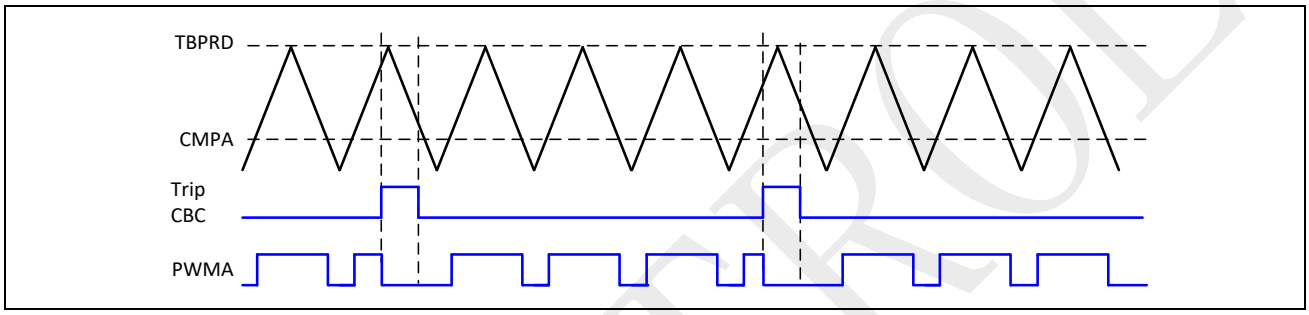
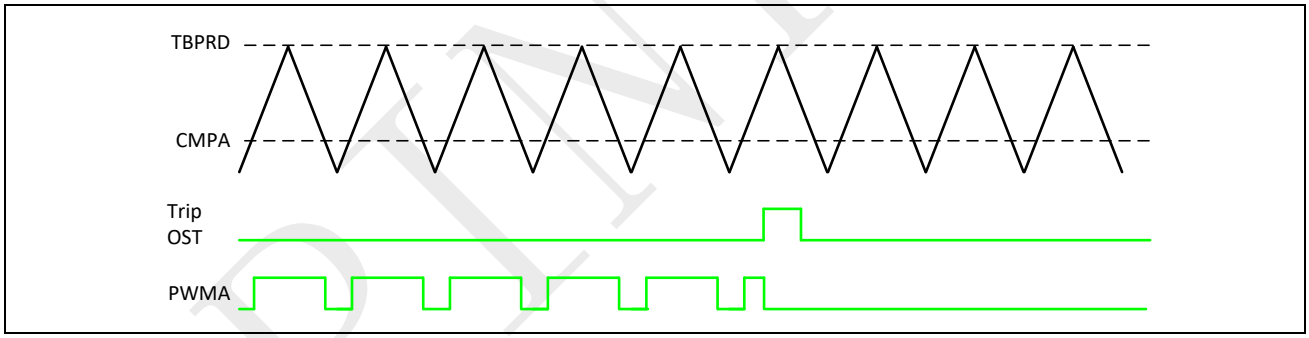


图 2-9：一次性封锁



PWM Trip Zone 输入信号源有：COMP，ADCPPU，GPIO(TZ0~TZ4)。

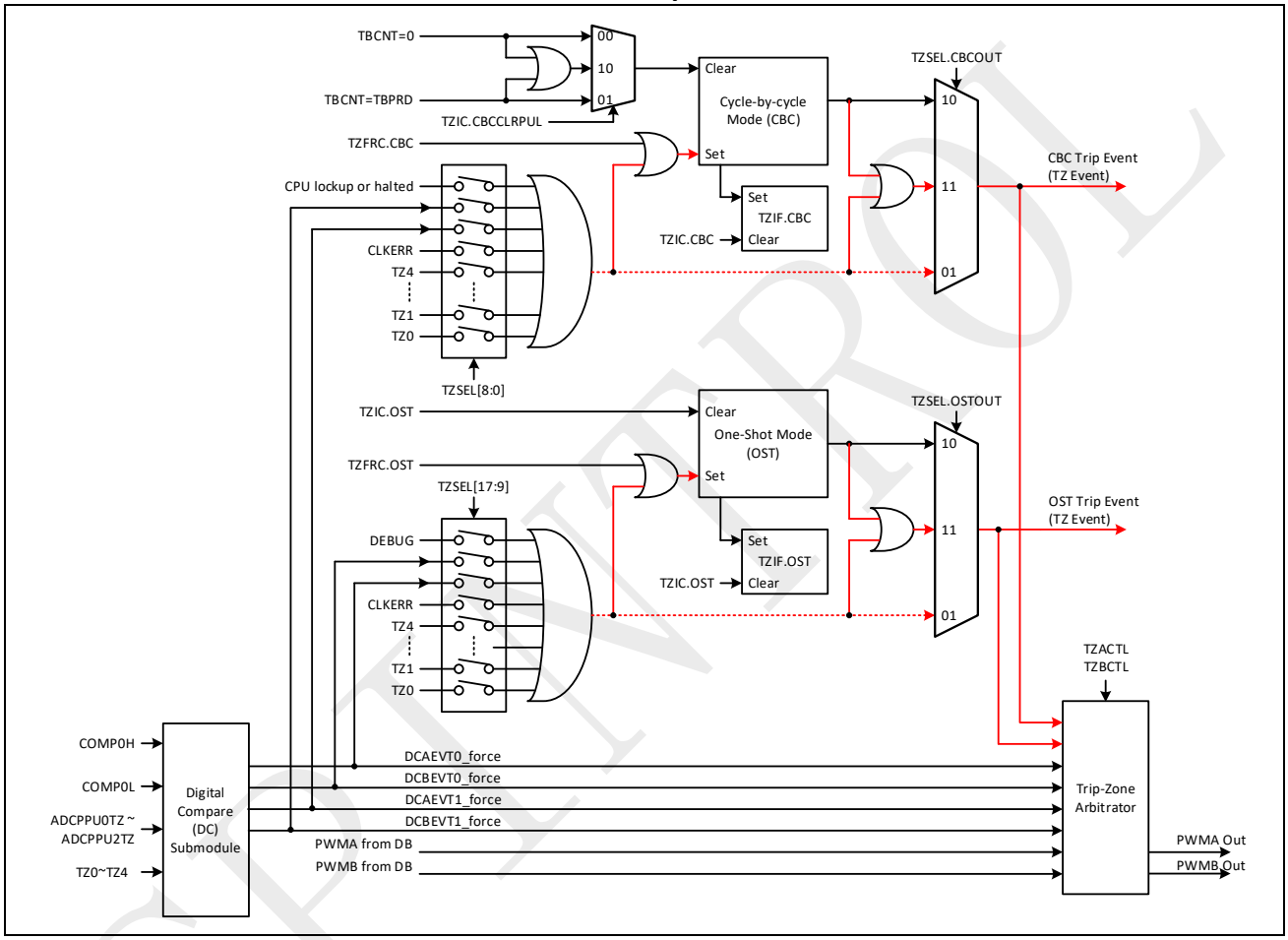
Trip Zone 的按周期和一次性可以同时配置，如果两个信号同时产生，则一次性封锁的信号将覆盖按周期的信号，即一次性封锁优先级高于按周期封锁。

2.5.1 GPIO 触发

TZ0~TZ4 就是来自 GPIO 的触发信号，其既可以通过 CBC 或 OST 直接产生 TZ 事件，又可以通过 DC 模块先转换为 DCAEVT0_force，DCBEVT0_force，DCAEVT1_force，DCBEVT1_force，再通过 CBC 或 OST 产生 TZ 事件。

通常使用第一种，因为其路径较短，如图 2-10 所示。

图 2-10：Trip Zone 功能框图



代码实现如下所示：

Example Code

```
int main(void)
{
    CLOCK_InitWithRCO(CLOCK_CPU_100MHZ);

    Delay_Init();

    /*
     * Init the UART
     */
    PIN_SetChannel(PIN_GPIO10, PIN_GPIO10_UART0_TXD);
    PIN_SetChannel(PIN_GPIO11, PIN_GPIO11_UART0_RXD);
    UART_Init(UART0, 38400);
    printf("Enter the test\n");
```

```
    /* Select PIN_GPIO13 as the channel B output of PWM1 */
    PIN_SetChannel(PIN_GPIO13, PIN_GPIO13_PWM1B);

    /* Init PWM1 and output 20kHz waveform on channel B */
    PWM_InitSingleChannel(PWM1, PWM_CHB, PWM_FREQ);

    /* Configure PWM counter as up counting mode */
    PWM_SetCounterMode(PWM1, COUNT_DOWN);

    /*
    * The PWM counting mode is set to be up-down by default, as a result, we
need
    * to re-config the AQCTLA register, which is used for controlling the PWM
output
    * action when ZRO/CAU/PRD/CAD (for more information, please read the PWM
application
    * note) event had happened.
    *
    * Key Point: the PWM freq will be double because of the ZRO/CAU/PRD/CAD
event.
    */
    PWM_ActionQualifierCHB(PWM1, AQCTLB_ZRO_SET_HIGH
                         | AQCTLB_CAU_DO_NOTHING
                         | AQCTLB_PRD_DO_NOTHING
                         | AQCTLB_CAD_SET_LOW);

    /* Set PWM1B output 50% duty waveform */
    u32PWMPeriod = PWMPeriod(PWM_FREQ);
    PWM_SetCMPA(PWM1, u32PWMPeriod / 2);

    /* Start PWM1 */
    PWM_RunCounter(PWM1);

    /*
    * Trigger TZ0 event when PIN_GPIO6 is low
    */
    PWM_SetTZ0FromGPIO(PIN_GPIO6, GPIO_LEVEL_LOW);

    /*
    * De-glitch settings, if IO signal state can keep 128 PCLK cycles,
    * then it will will be assumed valid
    */
    PIN_SetDeglitchWindow(DGCLKCTL_PDIV_128);
    PIN_EnableDeglitch(PIN_GPIO6);

    /* Trigger TZ1 event when PIN_GPIO8 is low */
    PWM_SetTZ1FromGPIO(PIN_GPIO8, GPIO_LEVEL_LOW);

    /* Trigger TZ2 event when PIN_GPIO9 is low */
    PWM_SetTZ2FromGPIO(PIN_GPIO9, GPIO_LEVEL_LOW);

    /*
    * Set TZ0 as one-shot trip event. The one-shot mode means once the
    * corresponding event has happened, the PWM waveform will stop, and
    * will not start until the flag is cleared.
    *
    * The key point there is 'TZEVT_ASYNC_ONLY_FOR_DEBUG' can not be
    * set in practical engineering, it can only be used in DEBUG mode.
    */
    PWM_SetOneShotTripEvent(PWM1, TRIP_EVENT_TZ0, TRIP_OUTPUT_ASYNC_OR_LATCH);

    /*
```

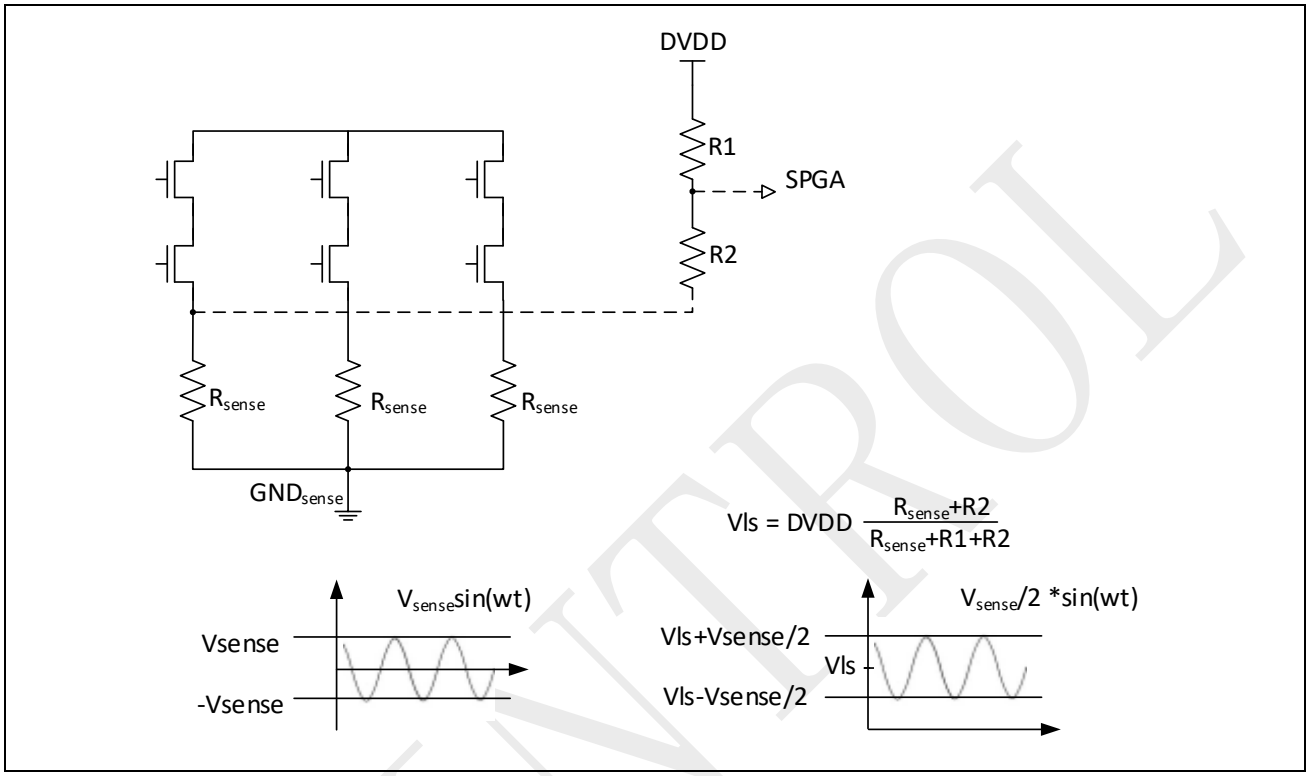
```
    * Set TZ1 and TZ2 as CBC trip event. The symbol 'CBC' means once the
    * corresponding event has happened, the PWM waveform will stop, but
    * start again at the next PWM CLK period with do nothing manually.
    *
    * The key point there is the same as 'PWM_SetOneShotTripEvent()'
    */
    PWM_SetCBCTripEvent(PWM1, TRIP_EVENT_TZ1 | TRIP_EVENT_TZ2,
TRIP_OUTPUT_ASYNC_OR_LATCH);

    /*
    * Set PWM1 output as tristate upon one-shot and CBC trip event,
    * need to explicitly specify actions for all 6 trip scenarios
    */
    PWM_SetCHAOutputWhenTrip(PWM1, 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(PWM1, 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_EnableTripInt(PWM1, TRIP_INT_OST);
    PWM_EnableTripInt(PWM1, TRIP_INT_CBC);
    NVIC_EnableIRQ(PWM1TZ_IRQn);

    while (1)
    {
        Delay_Ms(500);

        /* Restore the output of wave in oneshot mode*/
        PWM_ClearTripInt(PWM1, TRIP_INT_OST);
    }
}

void PWM1TZ_IRQHandler(void)
{
    if (PWM_GetOneShotTripEventFlag(PWM1, TRIP_EVENT_TZ0))
    {
        printf("TZ0 one-shot trip event occurred\n");

        PWM_ClearOneShotTripEventFlag(PWM1, TRIP_EVENT_TZ0);
    }

    if (PWM_GetCBCTripEventFlag(PWM1, TRIP_EVENT_TZ1))
    {
        printf("TZ1 cycle-by-cycle trip event occurred\n");

        PWM_ClearCBCTripEventFlag(PWM1, TRIP_EVENT_TZ1);
    }

    if (PWM_GetCBCTripEventFlag(PWM1, TRIP_EVENT_TZ2))
    {
        printf("TZ2 cycle-by-cycle trip event occurred\n");

        PWM_ClearCBCTripEventFlag(PWM1, TRIP_EVENT_TZ2);
    }
```

```
    PWM_ClearTripInt(PWM1, TRIP_INT_CBC);

    PWM_ClearTripInt(PWM1, TRIP_INT_GLOBAL);
}
```

2.5.2 COMP 触发

COMP 可作为电流防护使用，以下代码以 PWM1 电流为例，当 SPGA 输出超过 2500mV 时，COMP0H 触动 PWM 输出停止，当 SPGA 的电流小于 1500mV，由 COMP0L 触发 PWM 停止。

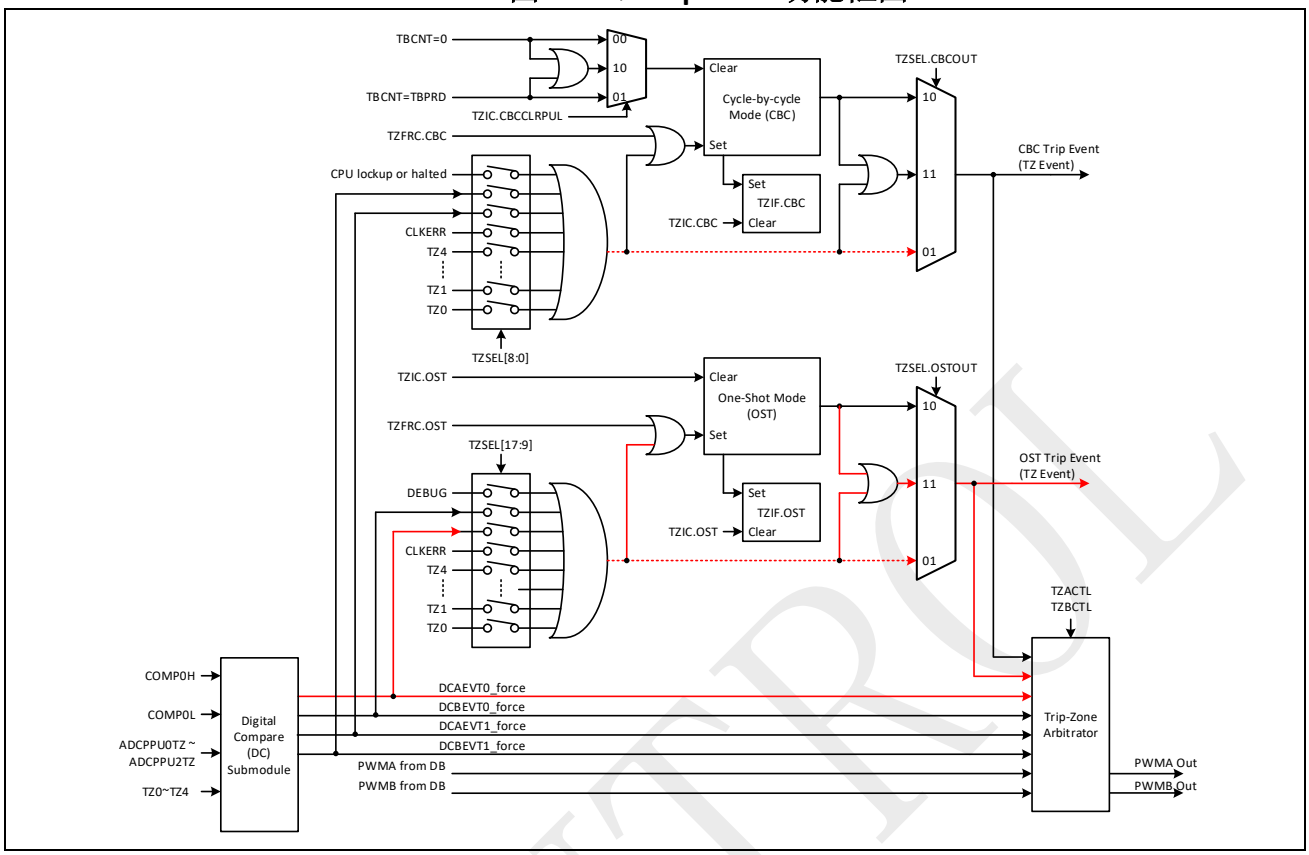
图 2-11：单端模式下采样电流示意图



COMP0H 或 COMP0L 可以通过 DC 模块转换为 DCAEVT0_force，DCBEVT0_force，DCAEVT1_force，DCBEVT1_force，再通过 CBC 或 OST 产生 TZ 事件，如图 2-12 所示。

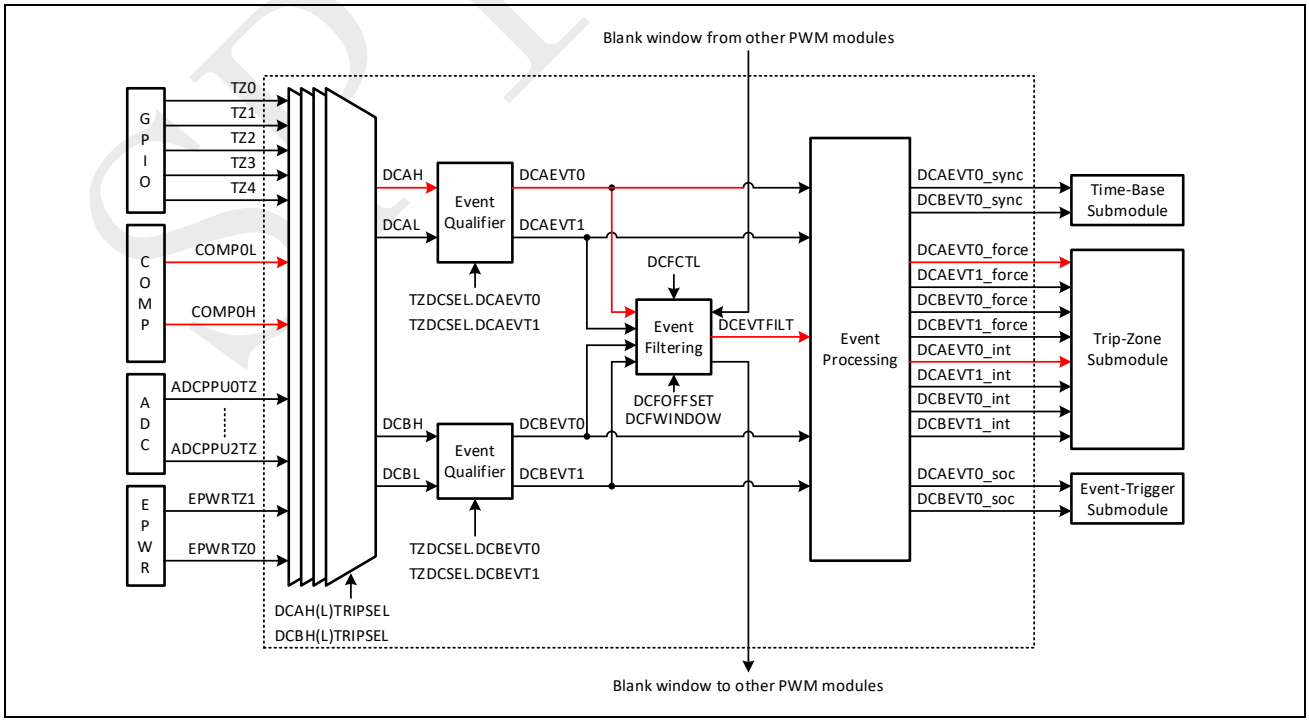
在使用该路信号的时候，要将 TZACTL, TZBCTL 中，和 DCAEVT0_force，DCBEVT0_force，DCAEVT1_force，DCBEVT1_force 相关控制位设置为 DO_NOTHING，否者 DCAEVT0_force，DCBEVT0_force，DCAEVT1_force，DCBEVT1_force 会不经过 CBC 或 OST，直接到达 Trip-Zone Arbitrator 模块。

图 2-12：Trip Zone 功能框图



其中在 DC 模块中的信号流如图 2-13 所示，示例代码通过 Event Filtering 模块对 DCAEVT0 信号进行过滤，避免了 Trip Zone 误触发。

图 2-13：DC 功能框图



代码实现如下所示：

Example Code

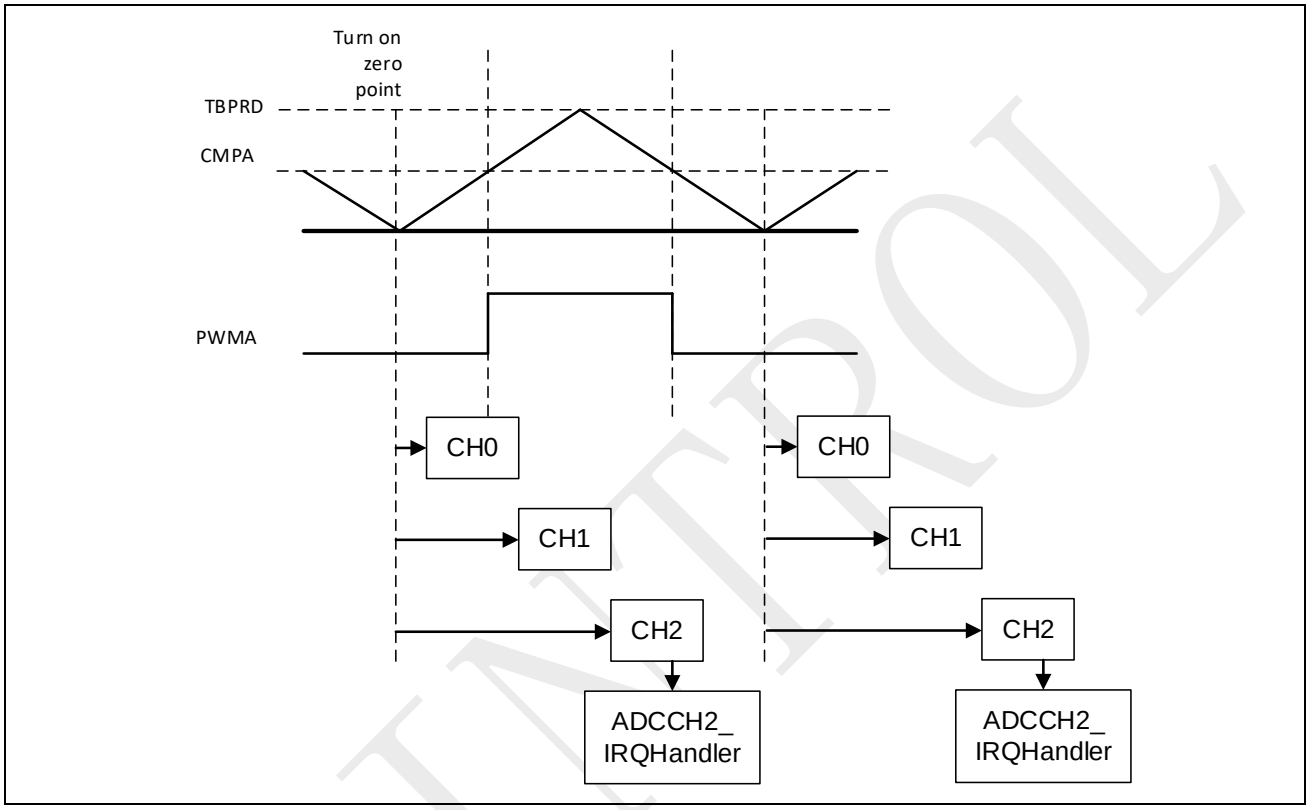
```
void PWMx_Set_TZ_Event(PWM_REGS *PWMx)
{
    /* Affected by results monitored from COMP0 */
    PWM_EnableDCAHTripEvent(PWMx, DC_TRIP_COMP0H);
    PWM_EnableDCAHTripEvent(PWMx, DC_TRIP_COMP0L);

    /*DCAEVT0 event comes from DCAL=don't care, DCAH=high*/
    PWM_SetRawDCAEVT0(PWMx, DCH_HIGH_DCL_X);

    /* Set the filter, Filter trigger conditions is TBCNT=0*/
    PWM_SetDCFilter(PWMx, DCF_FROM_RAW_DCAEVT0, DCF_ALIGN_ON_ZERO);

    /* Enable PWM DC filter blank function */
    PWM_EnableDCFilterBlank(PWMx);

    /* Set blank window size as 100 TBCLK and offset as 50 TBCLK */
    PWM_SetDCFilterBlankWindow(PWMx, PWM_BlankWIN_Size, PWM_Blank_Offset);

    /* USe the filtered to deal the data from DCAEVT0 */
    PWM_SetDCAEVT0(PWMx, DCEVT_FILTERED);

    /* Set the DCAEVT0 as OneShot mode */
    PWM_SetOneShotTripEvent(PWMx, TRIP_EVENT_DCAEVT, TRIP_OUTPUT_LATCH);

    /* Set output to tri-state upon OST trip event */
    PWM_SetCHAOutputWhenTrip(PWMx, TZU_TRIP_AS_LOW |
                           TZD_TRIP_AS_LOW |
                           DCEVT0U_TRIP_DO_NOTHING |
                           DCEVT0D_TRIP_DO_NOTHING |
                           DCEVT1U_TRIP_DO_NOTHING |
                           DCEVT1D_TRIP_DO_NOTHING);

    PWM_SetCHBOutputWhenTrip(PWMx, TZU_TRIP_AS_LOW |
                           TZD_TRIP_AS_LOW |
                           DCEVT0U_TRIP_DO_NOTHING |
                           DCEVT0D_TRIP_DO_NOTHING |
                           DCEVT1U_TRIP_DO_NOTHING |
                           DCEVT1D_TRIP_DO_NOTHING);

    PWM_EnableTripInt(PWMx, TRIP_INT_OST);
}


int main(void)
{
    CLOCK_InitWithRCO(CLOCK_CPU_100MHZ);

    Delay_Init();

    /*
     * Init the UART
     */
    PIN_SetChannel(PIN_GPIO10, PIN_GPIO10_UART0_TXD);
    PIN_SetChannel(PIN_GPIO11, PIN_GPIO11_UART0_RXD);
    UART_Init(UART0, 38400);

    printf("Enter the test\n");
    /*
```

```
    * set the GPIO as ADC.
    */
    PIN_SetChannel(PIN_GPIO0, PIN_GPIO0_ANA_IN0);

    /*
    * Set PGA in  single-ended mode.
    *
    */
    PGA_InitSPGA(SPGA_FROM_ANA_IN0, SPGA_GAIN_1X);

    /*
     * 1. Initialize comparator with 300ns deglitch filtering window.
     * 2. Set DAC voltage, COMP_HI's negative port is 2500mV, COMP_LO's positive
port is 1500mV
     */
    COMP_Init(COMP_H, COMP_FROM_SPGA_OUT, SampleRgisterNVol + VolOffsetmV,
COMP_FilterWIN);
    COMP_Init(COMP_L, COMP_FROM_SPGA_OUT, SampleRgisterNVol - VolOffsetmV,
COMP_FilterWIN);

    /* Select the channel A/B output of PWM1 respectively */
    PIN_SetChannel(PIN_GPIO12, PIN_GPIO12_PWM1A);
    PIN_SetChannel(PIN_GPIO13, PIN_GPIO13_PWM1B);

    PWM_InitComplementaryPairChannel(PWM1, PWM_FREQ, PWM_DB_NS);

    /* Set PWM1A output 25% duty waveform */
    u32PWMPeriod = PWMPeriod(PWM_FREQ);
    PWM_SetCMPA(PWM1, (u32PWMPeriod * 3) / 4);

    /* Start counting */
    PWM_RunCounter(PWM1);

    /* Set the DC sub-module for PWM1 */
    PWMx_Set_TZ_Event(PWM1);

    NVIC_EnableIRQ(PWM1TZ_IRQn);

    while (1)
    {
        Delay_Ms(1000);

        /* Restore the output of wave in oneshot mode*/
        PWM_ClearTripInt(PWM1, TRIP_INT_OST);
    }
}

void PWM1TZ_IRQHandler(void)
{
    if (PWM_GetOneShotTripEventFlag(PWM1, TRIP_EVENT_DCAEVT))
    {
        printf("one-shot trip event occurred\n");

        PWM_ClearOneShotTripEventFlag(PWM1, TRIP_EVENT_DCAEVT);
    }
    PWM_ClearTripInt(PWM1, TRIP_INT_GLOBAL);
}
```

2.6 PWM 触发 ADC 采样

在 PWM 触发三相电流采样的场景里，设定 PWM0 之 TBCNT 过零时触发 ADC，一次同时触发 CH0～CH2 分别负责转换三相电流，当 CH2 转换完之后，进入中断服务程序(ADCCH2_IRQHandler)，获取采样值，如图 2-14 所示。

图 2-14：PWM 触发三相电流采样



代码实现如下所示：

Example Code

```
int main(void)
{
    CLOCK_InitWithRCO(CLOCK_CPU_100MHZ);

    Delay_Init();

    /*
     * Init the UART
     */
    PIN_SetChannel(PIN_GPIO10, PIN_GPIO10_UART0_TXD);
    PIN_SetChannel(PIN_GPIO11, PIN_GPIO11_UART0_RXD);
    UART_Init(UART0, 38400);

    printf("Enter the test\n");

    /* Select the channel A/B output of PWM0 respectively */
    PIN_SetChannel(PIN_GPIO19, PIN_GPIO19_PWM0A);
    PIN_SetChannel(PIN_GPIO22, PIN_GPIO22_PWM0B);

    /* set the GPIO to observe the SYNC signal*/
```

```
    PIN_SetChannel(PIN_GPIO18, PIN_GPIO18_PWMSOCO);

    /* Initial Basic Complementary PWM */
    PWM_InitComplementaryPairChannel(PWM0, PWM_FREQ, PWM_DB_NS);

    /* Enable PWM0SOCA output to the pin */
    PWMCFG->SOCAOCTL = (PWMCFG->SOCAOCTL & (~SOCAOCTL_SOCA0EN_Msk)) |
SOCAOCTL_SOCA0EN_ENABLE;

    /* Set pulse duration of PWMSOCA output to the pin */
    PWMCFG->SOCAOCTL = (PWMCFG->SOCAOCTL & (~SOCAOCTL_DURATION_Msk)) |
SOCAOCTL_DURATION_32_PWM_CLK;


    /* Set PWM0A output 25% duty waveform */
    u32PWMPeriod = PWMPeriod(PWM_FREQ);
    PWM_SetCMPA(PWM0, (u32PWMPeriod * 3) / 4);

    /* Start counting */
    PWM_RunCounter(PWM0);

    /* PWM trigger timing selecting */
    PWM_SetSOCATiming(PWM0, EQU_ZERO);

    /* PWM trigger period selecting */
    PWM_SetSOCAPeriod(PWM0, ON_1ST_EVENT);

    /* Enable PWM SOCA trigger */
    PWM_EnableSOCA(PWM0);

    /* ADC initial */
    ADC_EasyInit1(ADC, ADC_CH0, PIN_GPIO0, ADC_SOC_TRIGGER_FROM_PWM0SOCA);
    ADC_EasyInit1(ADC, ADC_CH1, PIN_GPIO2, ADC_SOC_TRIGGER_FROM_PWM0SOCA);
    ADC_EasyInit1(ADC, ADC_CH2, PIN_GPIO4, ADC_SOC_TRIGGER_FROM_PWM0SOCA);

    /* SOC0 averaging 4 times */
    ADC_SetChannelResultAverageCount(ADC, ADC_CH0, ADC_AVERAGE_COUNT_4);
    ADC_SetChannelResultAverageCount(ADC, ADC_CH1, ADC_AVERAGE_COUNT_4);
    ADC_SetChannelResultAverageCount(ADC, ADC_CH2, ADC_AVERAGE_COUNT_4);

    /* INT service routine configuration */
    NVIC_SetPriority(ADCCH2_IRQn, 1);
    NVIC_EnableIRQ(ADCCH2_IRQn);

    while (1)
    {

    }
}



void ADCCH2_IRQHandler(void)
{
    /* The result */
    if (u32Count >= 15)
    {
        printf("CH0 %d\n", ADC_GetChannelResult(ADC, ADC_CH0));
        printf("CH1 %d\n", ADC_GetChannelResult(ADC, ADC_CH1));
        printf("CH2 %d\n", ADC_GetChannelResult(ADC, ADC_CH2));
        u32Count = 0;
    }
```

```
    u32Count++;

    /* Clear ADC SOC0 INT flag */
    ADC_ClearChannelInt(ADC, ADC_CH2);
}
```

2.6.1 观测 PWM 触发 ADC 采样信号

可以将 PWM 触发 ADC 采样信号通过具有 PWMSOCO 功能的引脚送出，输出给示波器观测。

Example Code

```
    /* set the GPIO to observe the SYNC signal*/
    PIN_SetChannel(PIN_GPIO18, PIN_GPIO18_PWMSOCO);

    /* Enable PWM0SOCA output to the pin */
    PWMCFG->SOCAOCTL = (PWMCFG->SOCAOCTL & (~SOCAOCTL_SOCA0EN_Msk)) |
SOCAOCTL_SOCA0EN_ENABLE;

    /* Set pulse duration of PWMSOCA output to the pin */
    PWMCFG->SOCAOCTL = (PWMCFG->SOCAOCTL & (~SOCAOCTL_DURATION_Msk)) |
SOCAOCTL_DURATION_32_PWM_CLK;
```