PWM Application Note

Overview

PWM plays a crucial role in power electronics systems and is widely used in motor control, switching power supplies, and UPS applications.

Applicable Range	
SPC1125 Series	SPC1125, SPC1128
SPC1168 Series	SPC1155, SPC1156, SPC1158, SPC1168, SPD1148, SPD1178, SPD1188, SPD1163, SPM1173
SPC2168 Series	SPC2168, SPC2165, SPC2166, SPC1198
SPC1169 Series	SPC1169, SPD1179, SPD1176, SPD1179B
SPC2188 Series	SPC1185, SPC2188

目录

图片列表

表格列表

Revision History

Version	Date	Author	State	Changes
C/0	2025-03-10	LemengZhou	Released	1. Initial release.

Terms or Abbreviations

Terms or Abbreviation	Description
PWM	Pulse Width Modulation

1 Feature

The PWM unit has the following features:

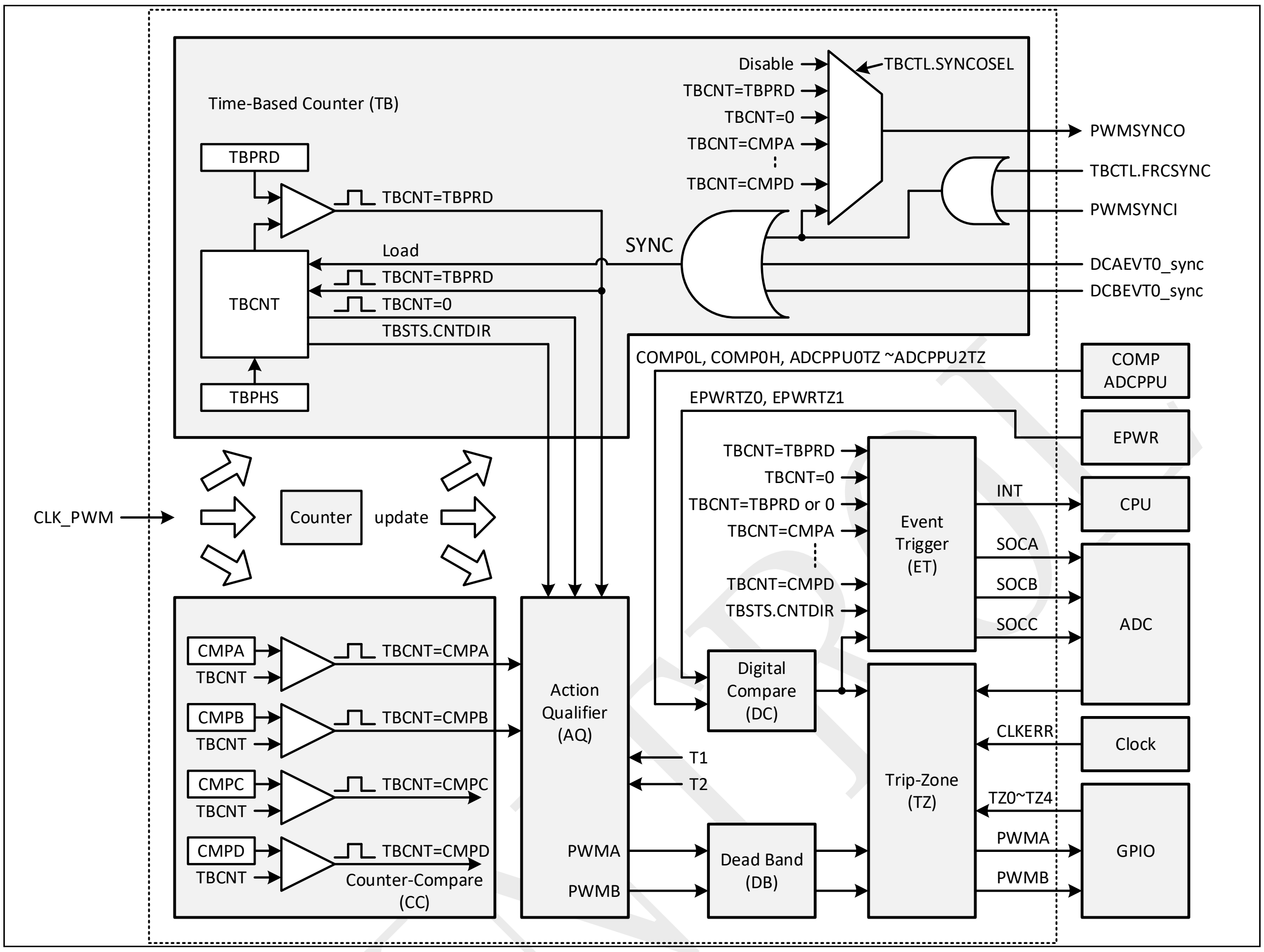
- A 16-bit time-base counter with configurable periods
- Software can directly force changes to the PWM output
- Independently configurable phase control
- Periodic phase synchronization
- Dead-time control with independently configurable rising-edge delay and falling-edge delay
- Configurable periodic or one-time blocking in the event of an anomaly
- Configurable PWM output to high, low, or high-impedance during blocking
- Digital comparison or blocking signal input can generate raw blocking events or filtered blocking events
- All events can be used to generate CPU interrupts or ADC REQ (start sampling) signals.

2 Function Description

The PWM module is composed of the following submodules:

- Time Base (TB) submodule
- Counter Compare (CC) submodule
- Action Qualifier (AQ) submodule
- Dead-Band (DB) submodule
- Trip Zone (TZ) submodule
- Digital Compare (DC) submodule
- Event Trigger (ET) submodule

Figure 2-1: PWM Unit Function Diagram



3 Function Example

3.1 PWM basic function

3.1.1 Example 1: Single-channel fixed-frequency PWM

3.1.1.1 Functional requirements

To generate a single-channel PWM waveform with the following requirements:

- Duty cycle of 50%;
- Frequency of 20 kHz;

3.1.1.2 Function implementation

The following steps are required to achieve the function:

1. Determine the PWM_Clock_Freq: This example is based on a PWM_Clock_Freq of 100 MHz for subsequent configuration value calculations.
2. Calculate TBPRD:

$$\mathrm{TBPRD} = \mathrm{PWM\_Clock\_Freq} / \mathrm{PWM\_Freq}/2$$

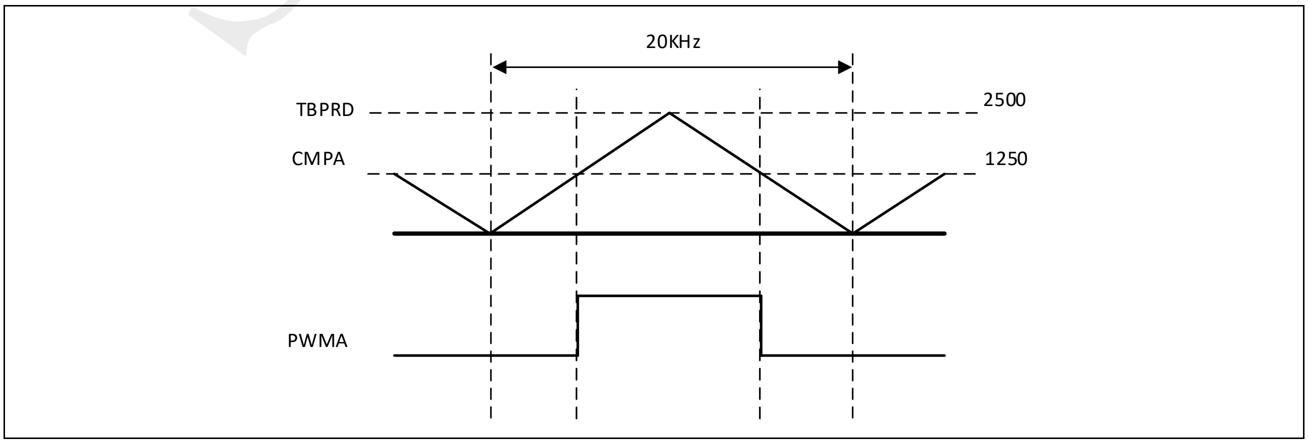
   The PWM_Freq is 20KHz and the calculated TBPRD is 2500.
3. Configure the PWM counting mode as up-down counting mode. In this counting mode, when the counter counts up and reaches the CMP value, the waveform is pulled high; when the counter counts down and reaches the CMP value, the waveform is pulled low.
4. Calculate CMP:

$$\mathrm{Duty\_Ratio} = \mathrm{CMP}/\mathrm{TBPRD}$$

   The Duty_Ratio is 50% and the calculated CMP is 1250.
5. Configure GPIO function as PWM output.
6. Run the PWM module.

Through the above implementation steps, the following Figure 3-1 example waveform can be generated:

Figure 3-1: Single-channel Independent PWM



The example code for the above implementation steps can refer to the Demo provided in the SDK, as shown in Table 3-1:

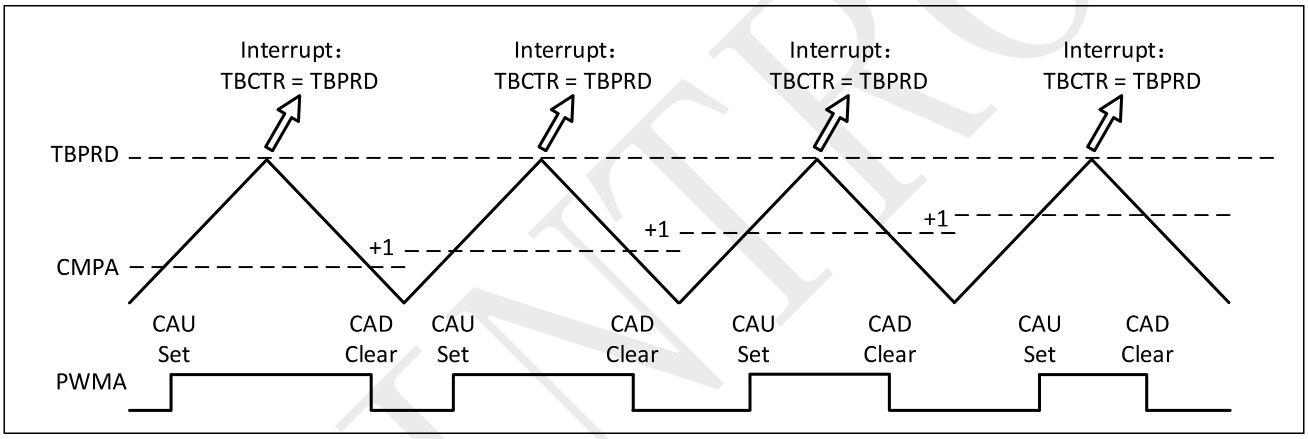
Table 3-1: Example 1 Code Path

MCU PRODUCT MODEL	Code Path
All	SDK DIRECTORY\0_Examples\ PWM_Single_Output_With_Up_Down_Counting_Mode

3.1.2 Example 2: Waveform Duty Cycle Adjustment

PWM can generate an interrupt event when the time-base counter value TBCTR equals TBPRD. In the interrupt service routine, the duty cycle can be adjusted by modifying the value of the CMP shadow register, as shown in Figure 3-2:

Figure 3-2: Periodically Varying PWM



3.1.2.1 Functional requirements

To generate a single-channel PWM waveform with the following requirements:

- Periodically adjust the PWM duty cycle from 0% to 100%.
- Frequency of 20 kHz;

3.1.2.2 Function implementation

The following steps are required to achieve the function:

1. Determine the PWM_Clock_Freq: This example is based on a PWM_Clock_Freq of 100 MHz for subsequent configuration value calculations.
2. Calculate TBPRD:

$$\text{TBPRD} = \text{PWM\_Clock\_Freq} / \text{PWM\_Freq}/2$$

The PWM_Freq is 20KHz and the calculated TBPRD is 2500.

3. Configure the PWM counting mode as up-down counting mode. In this counting mode, when the counter counts up and reaches the CMP value, the waveform is pulled high; when the counter counts down and reaches the CMP value, the waveform is pulled low.

4. Calculate CMP:

$$\text{Duty\_Ratio} = \text{CMP}/\text{TBPRD}$$

   The Duty_Ratio is 50% and the calculated CMP is 1250.

5. Configure the GPIO function as PWM output.
6. Set the loading timing of the CMP shadow register to the CMPA register to occur when the counter value is zero.
7. Enable the PWM to generate an interrupt event when TBCTR equals TBPRD.
8. Enable the CPU to handle PWM interrupt events.
9. Write an interrupt service routine (ISR) to modify the CMP value in order to adjust the duty cycle.
10. Run the PWM module.

The example code for the above implementation steps is as follows:

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);
```

Example Code

```
    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);
}
```

[1] The example code is applicable to the SPC1169 Series products. Example code for other series products will be supplemented based on actual requirements.

3.1.3 Example 3: Two complementary PWM waveforms with dead time.

In power electronics, switching power supplies, and motor control, it is often necessary to handle half-bridge and full-bridge circuits. A fundamental control method for such circuits is to use complementary PWM waveforms to drive the upper and lower bridge arm switching transistors of the half-bridge or full-bridge, respectively. Additionally, these complementary PWM waveforms need to include a certain dead time.

For dead time application in half-bridge control, it is generally achieved by adding dead time to one PWM waveform and then inverting it to obtain the complementary PWM waveform with dead time. The PWM module provides flexible dead time configuration, as shown in Figure 3-3.

Figure 3-3 and Figure 3-4 illustrate the process of generating a pair of complementary PWM waveforms with dead time from a single PWM waveform.

Figure 3-3: Dead Time Control Unit Block Diagram

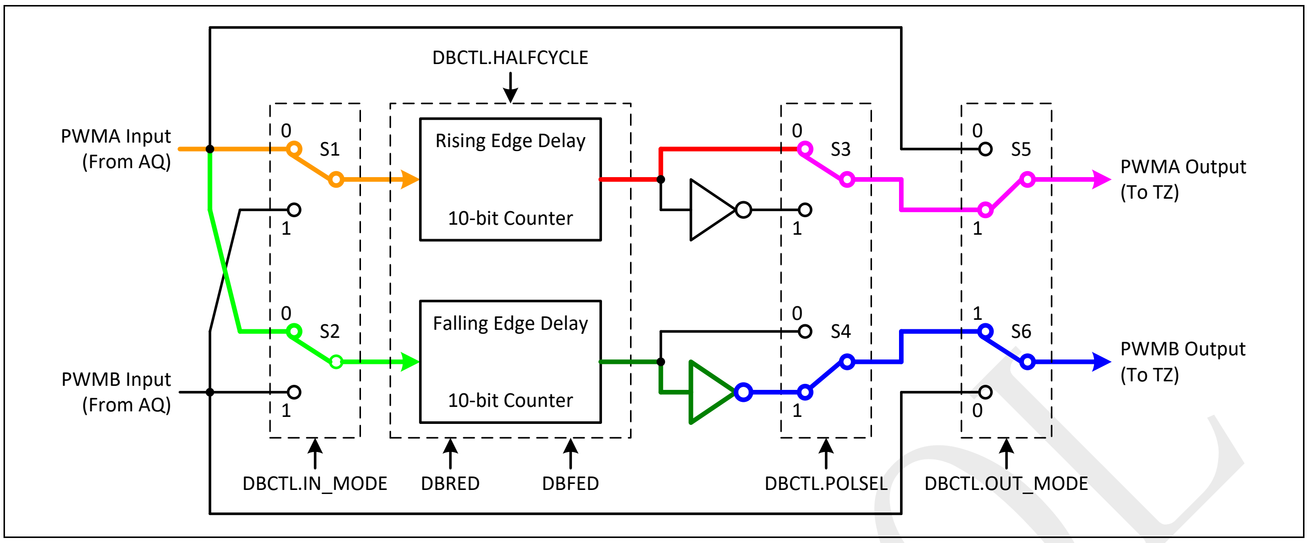
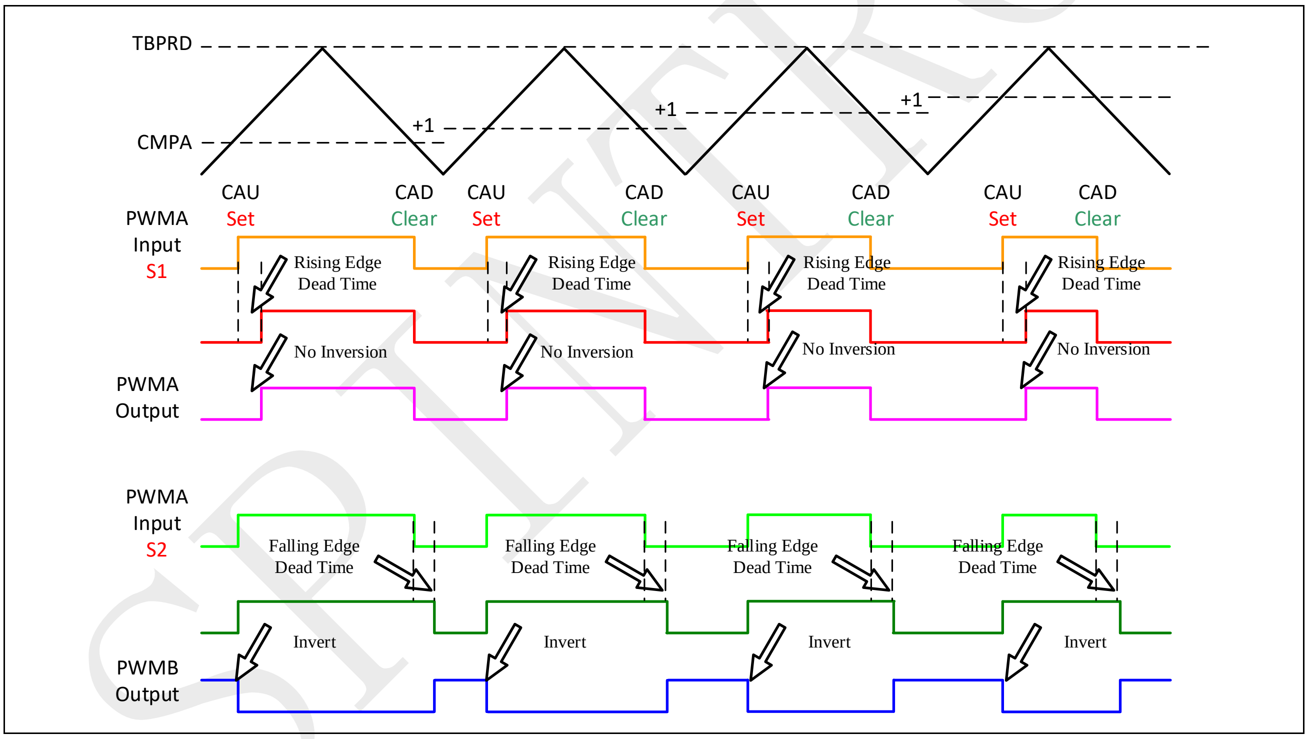


Figure 3-4: Schematic Diagram of Generating Complementary PWM with Dead Time



3.1.3.1 Functional requirements

To generate a half-bridge PWM waveform with dead time:

- PWM duty cycle of 75%;
- Frequency of 20 kHz;
- Dead time of 1000 ns.

3.1.3.2 Function implementation

1. Determine the PWM_Clock_Freq: This example is based on a PWM_Clock_Freq of 100 MHz for subsequent configuration value calculations.
2. Calculate TBPRD:

$$\mathrm{TBPRD} = \mathrm{PWM\_Clock\_Freq} / \mathrm{PWM\_Freq}/2$$

   The PWM_Freq is 20KHz and the calculated TBPRD is 2500.

3. Configure the PWM counting mode as up-down counting mode. In this counting mode, when the counter counts up and reaches the CMP value, the waveform is pulled high; when the counter counts down and reaches the CMP value, the waveform is pulled low.
4. Calculate CMP:

$$\mathrm{Duty\_Ratio} = \mathrm{CMP}/\mathrm{TBPRD}$$

   The Duty_Ratio is 75% and the calculated CMP is 725.

5. Configure the PWM for dual-channel complementary output.
6. Configure the dead time as 1000ns.
7. Configure the GPIO function as PWM output.
8. Run the PWM module.

The example code for the above implementation steps can refer to the Demo provided in the SDK, as shown in Table 3-2.

Table 3-2: Example 3 Code Path

MCU PRODUCT MODEL	Code Path
All	SDK DIRECTORY\0_Examples\PWM_Complementary_Pair_Channel

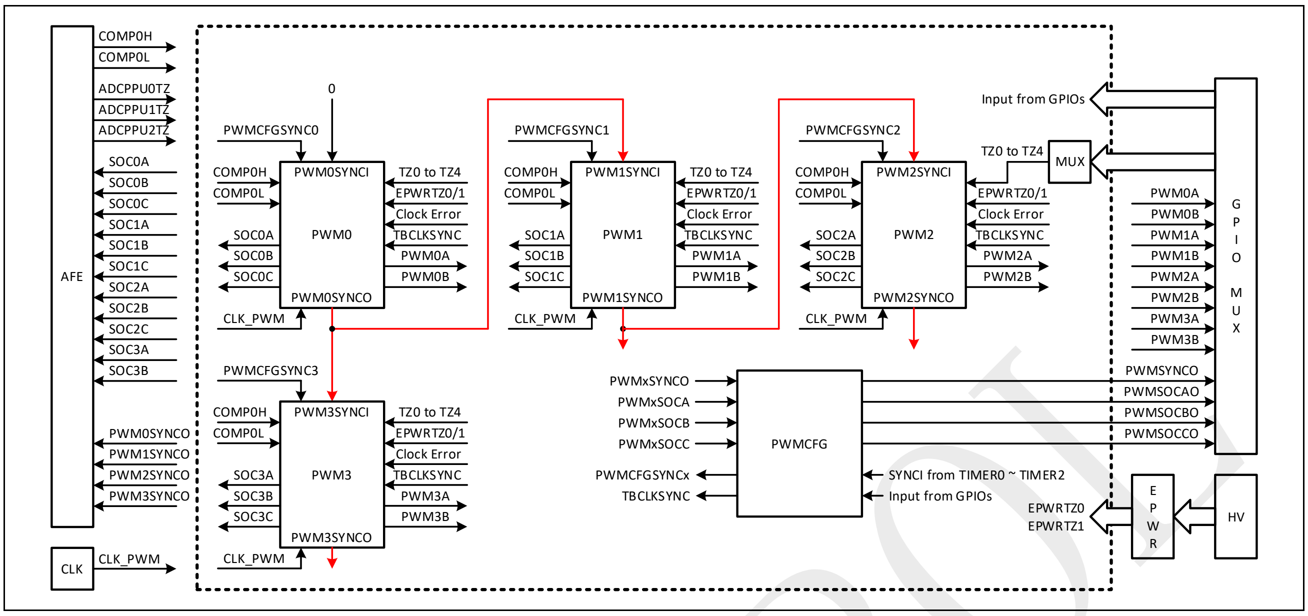
3.2 PWM Synchronization

During the use of PWM, it is sometimes necessary to generate a fixed phase difference between different PWM signals. This can be achieved using PWM cascade synchronization or non-cascade synchronization.

3.2.1 Example 4: Cascade Method for PWM Synchronization

Cascade synchronization refers to the sequential transmission of synchronization signals across different PWM modules, as shown in Figure 3-5.

Figure 3-5: Cascade Relationship



[1] Taking SPC1169 as an example, the cascade synchronization relationship can be established through the following chains: PWM0 -> PWM1 -> PWM2 or PWM0 -> PWM3, as shown in the figure above.

Note:
1. PWMSYNCI can come not only from PWMSYNCO but also from Timer or GPIO.
2. The number of PWMx modules and their cascade relationships vary across different product models. Please refer to the "PWM System Overview" diagram in the TRM of the corresponding product model for confirmation.

3.2.1.1 Functional requirements

Using the cascade function, synchronize PWM0, PWM1, and PWM2 to output PWM waveforms.

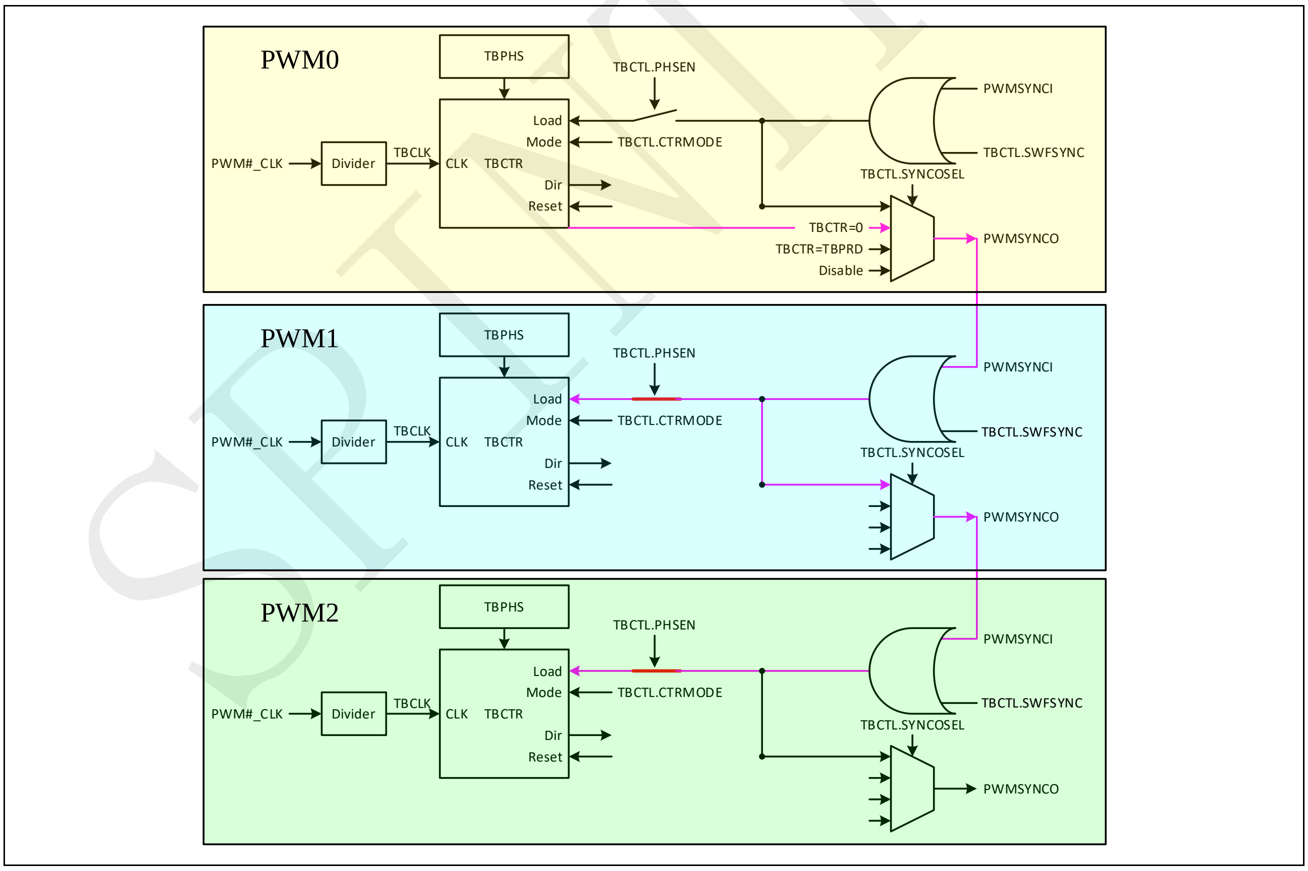
- PWM0, PWM1, and PWM2 are all in up-down counting mode.
- The frequency of PWM0, PWM1, and PWM2 is 20 kHz.
- The duty cycle of PWM0, PWM1, and PWM2 is 33.33%.
- When the PWM0 counter reaches 0, cascade synchronization is applied to PWM1 and PWM2, resulting in a 120° phase difference between PWM0, PWM1, and PWM2.

3.2.1.2 Function implementation

The following steps are required to achieve the function:

1. Determine the cascade synchronization relationship:
   a) Enable PWM0 to output the TBCTR=0 signal through PWMSYNCO to PWM1;
   b) Enable PWM1 to output the PWMSYNCI signal through PWMSYNCO to PWM2;
   c) The block diagram of the cascade synchronization relationship is shown in Figure 3-6:

Figure 3-6: Cascade Synchronization



2. Determine the phase difference generation method:
   a) The synchronized time-base phase value should be the value of the counter comparator.
   b) Both PWM1 and PWM2 are configured in up-down counting mode.

c) PWM1 is configured to transition to a high level when the TBCNT=CMPA event occurs and TBCNT is counting up.
d) PWM1 is configured to transition to a low level when the TBCNT=CMPA event occurs and TBCNT is counting down.
e) PWM2 is configured to transition to a low level when the TBCNT=CMPA event occurs and TBCNT is counting up.
f) PWM2 is configured to transition to a high level when the TBCNT=CMPA event occurs and TBCNT is counting down.

For other basic requirements, please refer to Section 3.1。

3. The example code for the above implementation steps can refer to the Demo provided in the SDK, such as:Table 3-2:

Table 3-3: Example 4 Code Path

Product Model	Code Path
SPC1125 Series, SPC1169 Series, SPC2188 Series	SDK DIRECTORY\0_Examples\PWM_Software_Group_SYNC
SPC1168 Series, SPC2168 Series	SDK DIRECTORY \0_Examples\ PWM_TBCNT_0_SYNC

3.2.2 Example 5: Non-Cascade Method for PWM Synchronization

Non-cascade synchronization refers to the synchronization signal being simultaneously applied to different PWM modules.

3.2.2.1 Functional requirements

Using the software-forced synchronization function, synchronize PWM0, PWM1, and PWM2 to output PWM waveforms.

- PWM0, PWM1, and PWM2 are all in up-down counting mode.
- The frequency of PWM0, PWM1, and PWM2 is 20 kHz.
- The duty cycle of PWM0, PWM1, and PWM2 is 33.33%.
- When software-forced synchronization is applied:

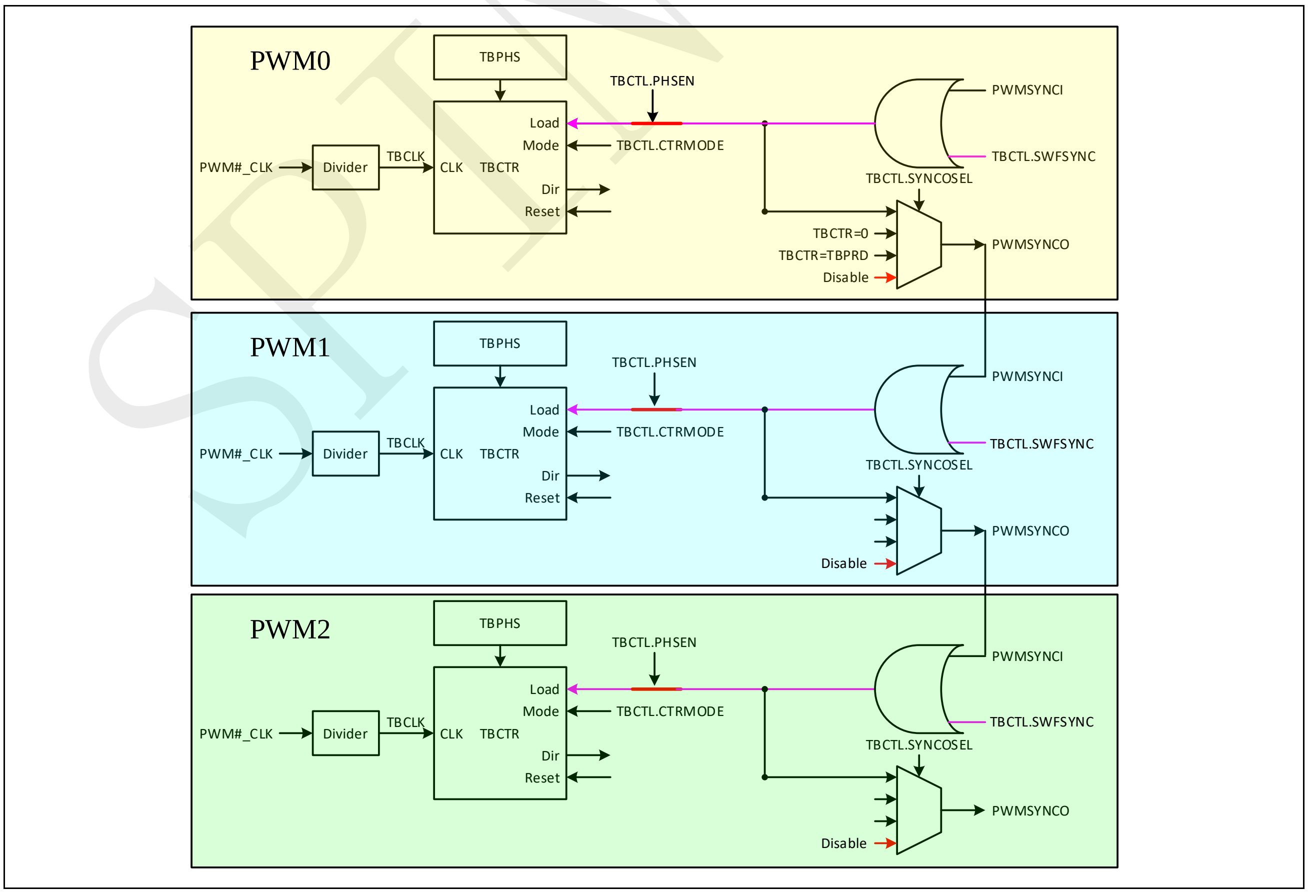
PWM0, PWM1, and PWM2 generate a 120° phase difference.

3.2.2.2 Function implementation

The following steps are required to achieve the function:

1. Confirm the synchronization relationship:
   a) Use the software-forced synchronization signal TBCTL.SWFSYNC to simultaneously synchronize PWM0, PWM1, and PWM2.
   b) The synchronization relationship is shown in the following figure:

Figure 3-7: Non-Cascade Synchronization



2. Determine the level transition state when the time-base counter TBCNT equals the counter comparator CMPA.
   a) PWM0, PWM1, and PWM2 are all configured in up-down counting mode.
   b) PWM0 is configured to transition to a high level when the TBCNT=CMPA event occurs and TBCNT is counting up.
   c) PWM0 is configured to transition to a low level when the TBCNT=CMPA event occurs and TBCNT is counting down.
   d) PWM2 is configured to transition to a low level when the TBCNT=CMPA event occurs and TBCNT is counting up.
   e) PWM2 is configured to transition to a high level when the TBCNT=CMPA event occurs and TBCNT is counting down.
   f) There are no special requirements for PWM1, so it is configured to be consistent with PWM0.
3. Disable the PWMSYNCO function for each PWM module.
4. Use software to simultaneously change the phase of PWM0, PWM1, and PWM2.
5. For other basic requirements, please refer to Section 3.1.

The example code for the above implementation steps can refer to the Demo provided in the SDK, as shown in Table 3-2:

Table 3-4: Example 5 Code Path

MCU PRODUCT MODEL	Code Path
SPC1125 Series, SPC1169 Series, SPC2188 Series	SDK DIRECTORY\0_Examples\PWM_Software_Global_SYNC
SPC1168 Series, SPC2168 Series	SDK DIRECTORY\0_Examples\PWM_Global_Software_Force_SYNC

3.2.3 Observe Synchronized Output Signals

The PWM synchronized output signals can be sent through pins with the PWMSOCO function and output to an oscilloscope for observation.

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;
```

[1] The example code is applicable to the SPC1169 series. Example code for other series products will be supplemented based on actual requirements.

3.3 PWM Trip Zone

In many application scenarios, there are signals similar to emergency stops. When such a signal is received, it is necessary to immediately or safely halt all actions. If the PWM output is regarded as such an action or the driving signal for the action, there is also a need to immediately perform a safe stop. This can be achieved in the following ways:

- Cycle-by-Cycle (CBC) blocking of PWM output, which restarts in the next PWM cycle (suitable for constant current control in power supplies or constant current microstepping control in stepper motors).
- One-Shot (OST) blocking of PWM output, where the PWM output is directly stopped and will not resume until the user intervenes and clears the One-shot flag.
- PWM Trip Zone input signal sources include: COMP, ADCPPU, and GPIO (TZ0~TZ4).

Figure 3-8: CBC Blocking

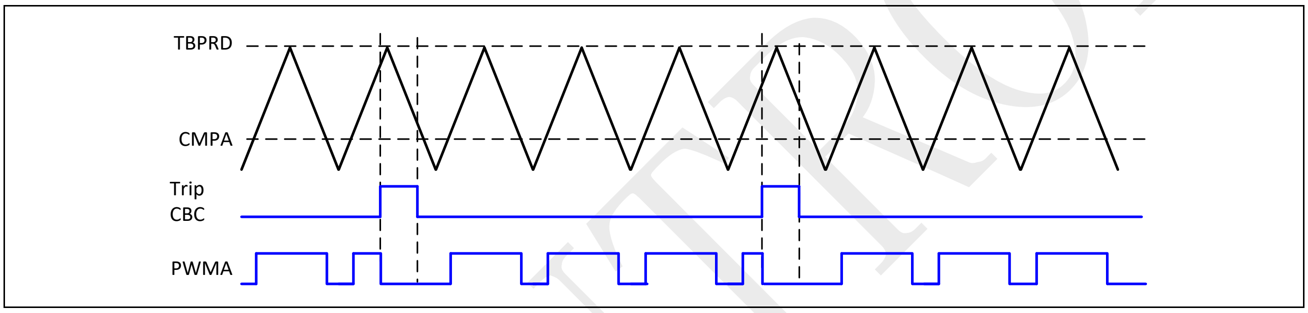
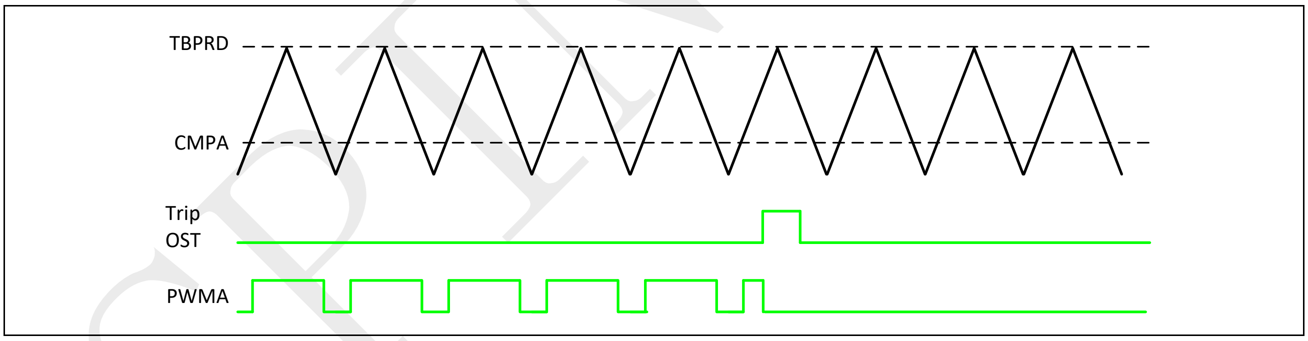


Figure 3-9: One-shot Blocking



Note: Trip Zone can be configured for both Cycle-by-Cycle and One-Shot blocking simultaneously. If both signals are generated at the same time, the One-Shot blocking signal will override the Cycle-by-Cycle signal, meaning One-Shot blocking has a higher priority than Cycle-by-Cycle blocking.

3.3.1 Example 6: GPIO Trigger Blocking

3.3.1.1 Functional requirements

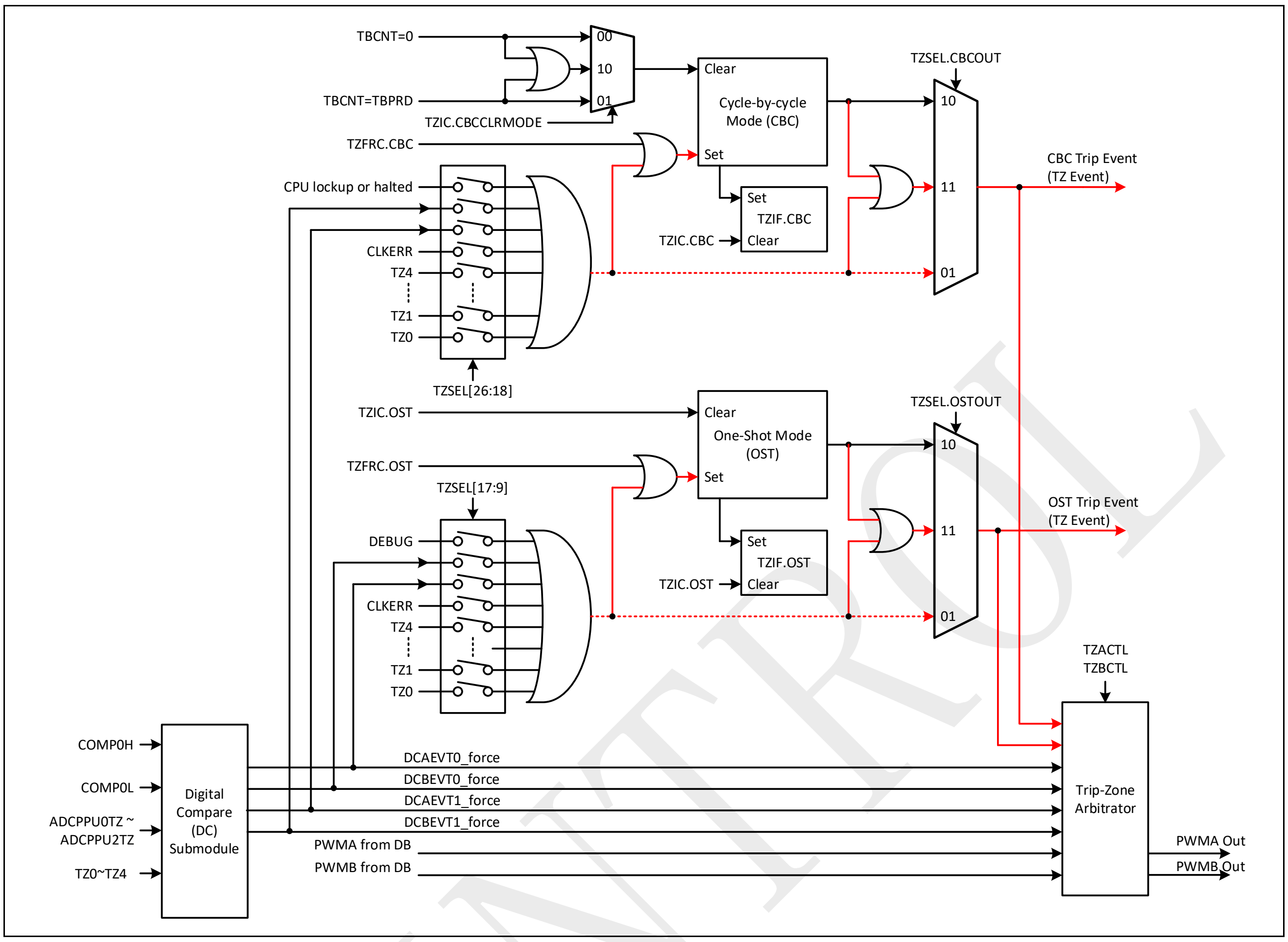
Use a GPIO trigger signal to generate a TZ event, resulting in a one-time blocking of the output PWM waveform.

Use a GPIO trigger signal to generate a TZ event, resulting in cycle-by-cycle blocking of the output PWM waveform.

3.3.1.2 Function implementation

1. Determine the trigger signal path:
   a) The GPIO trigger signals are named TZ0~TZ4, which can directly generate TZ events through CBC or OST.
   b) Alternatively, they can first be converted into DCAEVT0_force, DCBEVT0_force, DCAEVT1_force, or DCBEVT1_force through the DC module, and then generate TZ events through CBC or OST.
   c) Analysis shows that the first method is typically used because its path is shorter, as shown in Figure 3-10.
2. Assign TZ0, TZ1, and TZ2 signals to different GPIOs.
3. Configure TZ0 as a One-Shot blocking event, and TZ1, TZ2 as Cycle-by-Cycle blocking events.
4. Configure the Action Qualifier Output Control Register (AQCTLx) so that the PWM output pins transition to a high-impedance state when a blocking event occurs.

Figure 3-10: Trip Zone Function Diagram



The example code for the above implementation steps can refer to the Demo provided in the SDK, such as:Table 3-2:

Table 3-5: Example 6 Code Path

MCU PRODUCT MODEL	Code Path
SPC1125 Series, SPC1169 Series, SPC2188 Series	SDK DIRECTORY\0_Examples\ PWM_GPIO_Trigger_TZ
SPC1168 Series, SPC2168 Series	SDK DIRECTORY\0_Examples\PWM_GPIO_Trigger_Trip_Zone

3.3.2 Example 7: COMP Trigger Blocking

Example Applicable Range
SPC1125 Series, SPC1169 Series, SPC2188 Series

3.3.2.1 Functional requirements

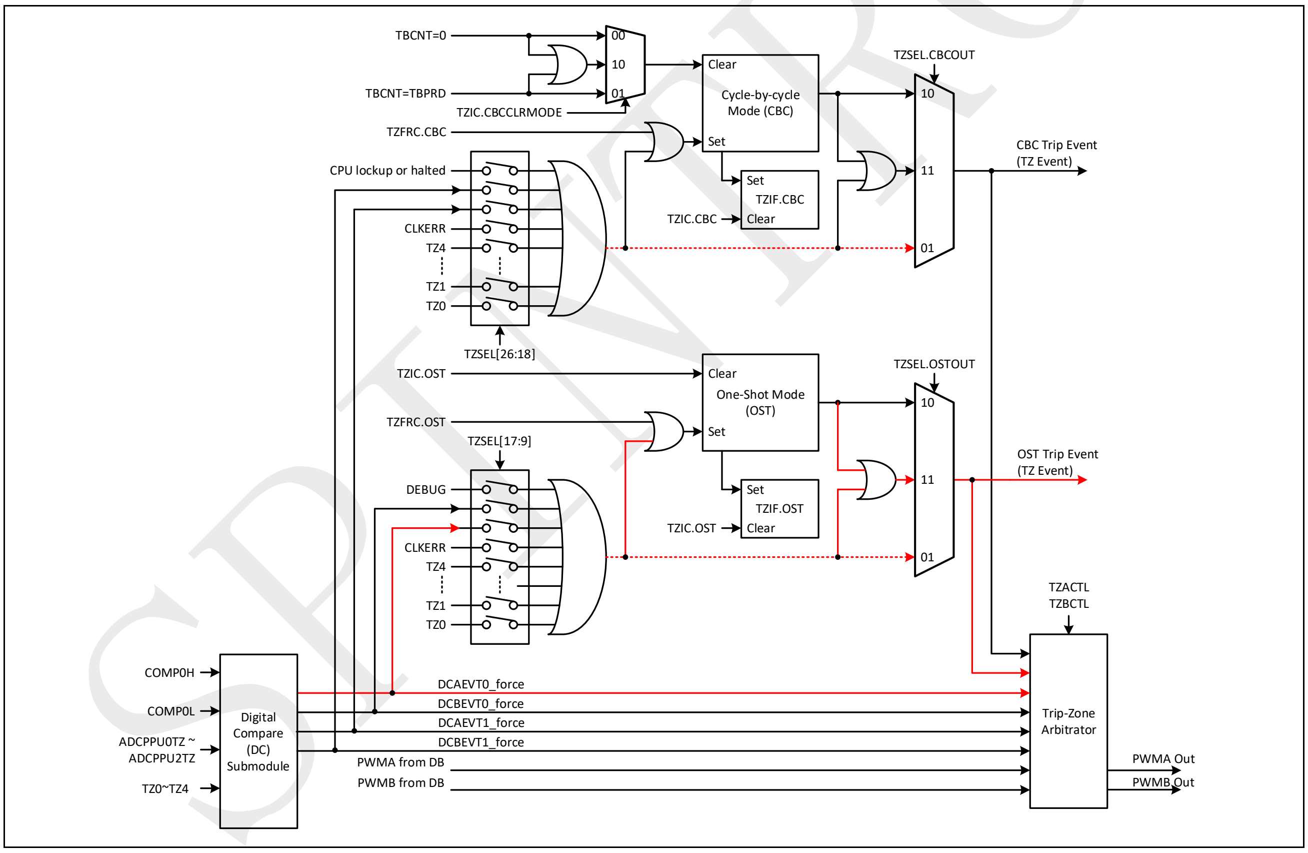
Use a COMP trigger signal to generate a TZ event, resulting in a one-time blocking of the output PWM waveform.

- Use COMP for current protection functionality, taking PWM1 output as an example. When the SPGA output exceeds 2500 mV, COMP0H triggers the PWM output to stop. When the SPGA current is less than 1500 mV, COMP0L triggers the PWM to stop.

3.3.2.2 Function implementation

1. Determine the trigger signal path:
   a) COMP0H or COMP0L can be converted into DCAEVT0_force, DCBEVT0_force, DCAEVT1_force, or DCBEVT1_force through the DC module, and then generate TZ events through CBC or OST, as shown in Figure 3-11.
   b) When using this signal path, set the control bits related to DCAEVT0_force, DCBEVT0_force, DCAEVT1_force, and DCBEVT1_force in TZACTL and TZBCTL to DO_NOTHING. Otherwise, DCAEVT0_force, DCBEVT0_force, DCAEVT1_force, and DCBEVT1_force will bypass CBC or OST and directly reach the Trip-Zone Arbitrator module.

Figure 3-11: Trip Zone Function Diagram



2. Detailed description of signal filtering functionality:
   a) Signal filtering is performed through the DC module, as shown in Figure 3-12.
   b) The blue dashed box represents an event selector. By configuring the TRIPSEL register, a logical OR selection can be performed on the events on the left. The selected event can trigger any of the events on the right, such as DCAH(L) or DCBH(L).
   c) The purple dashed box contains two Event Qualifier units. One Event Qualifier unit is used to logically filter DCAH(L) to generate RAWDCxEVT0 or RAWDCxEVT11. Taking the

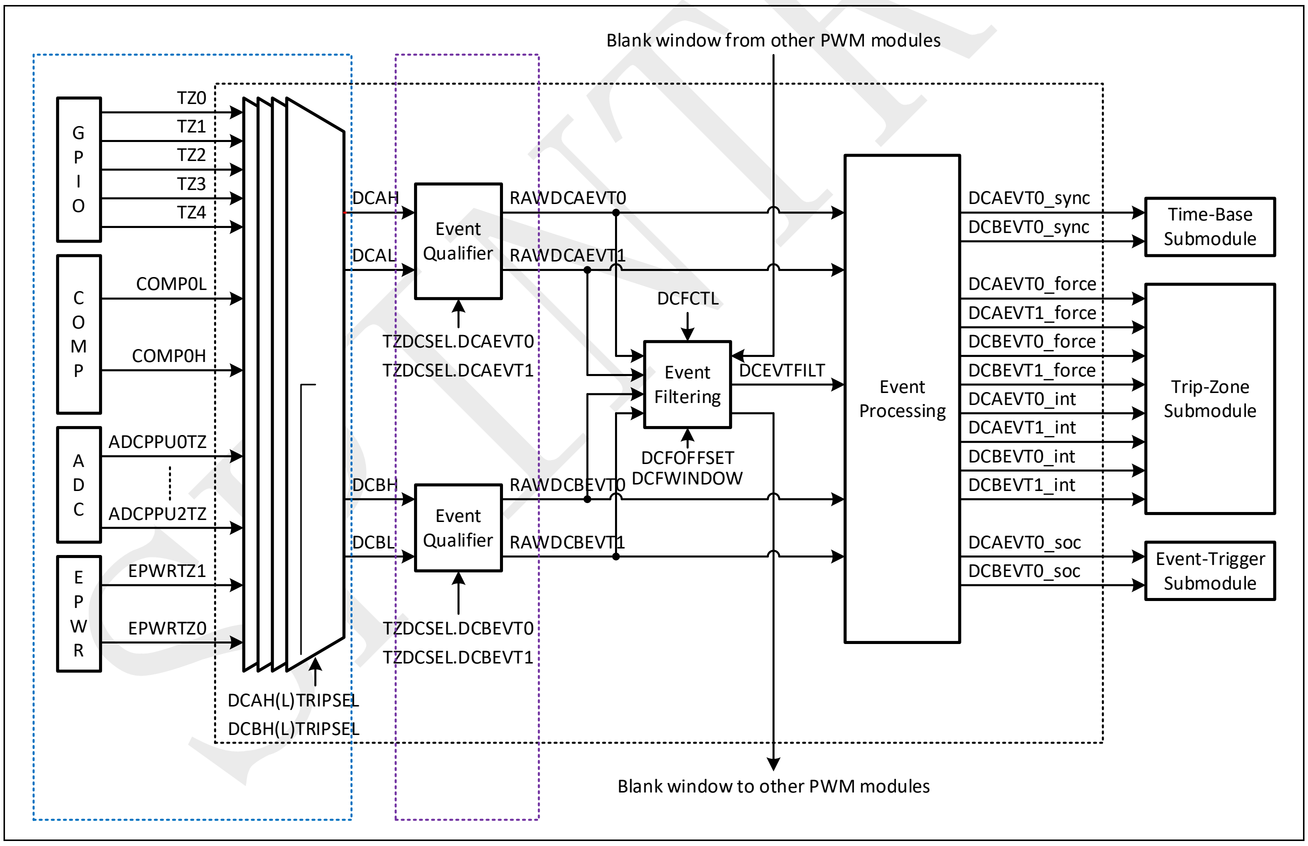
generation of the RAWDCAEVT0 event as an example, the logical filtering functionality of the Event Qualifier unit is described in Table 3-6:

Table 3-6: Event Qualifier Function List

Logical Filtering Configuration	Input Event	Trigger Event
0	/	Disable event
1	DCAL is low	RAWDCAEVT0
2	DCAL is high	
3	DCAH is low	
4	DCAH is high	
5	DCAL is low and DCAH is high	
6	DCAL is high and DCAH is low	
7	DCAL is high and DCAH is high	

Finally, the Event Filtering module performs window filtering on the RAWDCxEVT0 or RAWDCxEVT1 signals, and the Event Processing module generates the final output signal.

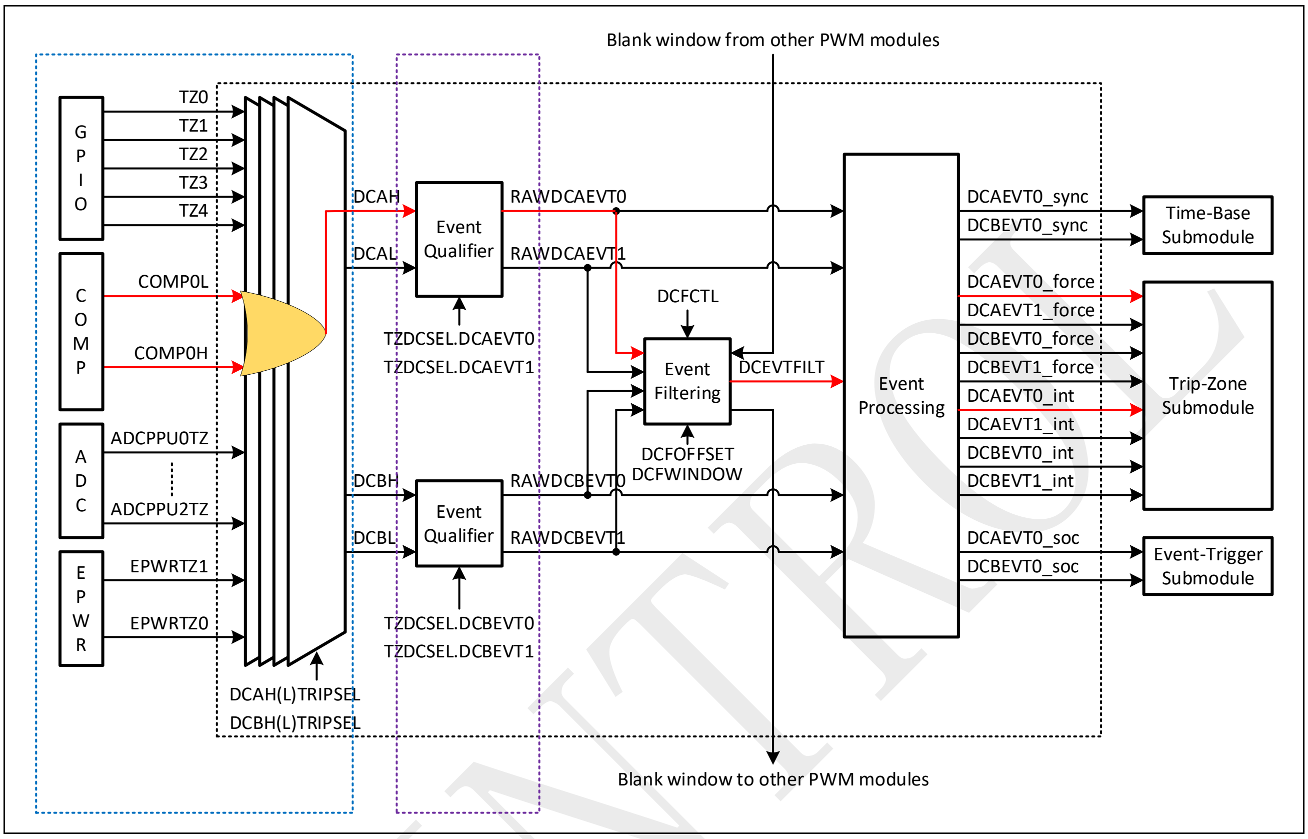
Figure 3-12: DC Function Diagram



3. Signal filtering in this example:
   a) The signal path is shown by the red line in Figure 3-13.
   b) By configuring the DCAHTRIPSEL register, any event triggered by COMP0L or COMP0H generates a DCAH event.
   c) By configuring the TZSCSEL.DCAEVT0 field, a RAWDCAEVT0 event is generated when DCAH is high.
   d) The Event Filtering module performs window filtering on the event signal.

e) The Event Processing module ensures that any event triggered by COMP0L or COMP0H generates the final output signals DCAEVT0_force and DCAEVT0_int.

Figure 3-13: DC Function Diagram



The example code for the above implementation steps can refer to the Demo provided in the SDK, such as:Table 3-2:

Table 3-7: Example 7 Code Path

MCU PRODUCT MODEL	Code Path
SPC1125 Series, SPC1169 Series, SPC2188 Series	SDK DIRECTORY \0_Examples\PWM_Current_Protect_Trigger_TZ

3.3.3 Example 8: COMP Trigger Blocking

Example Applicable Range
SPC1168 Series, SPC2168 Series

3.3.3.1 Functional requirements

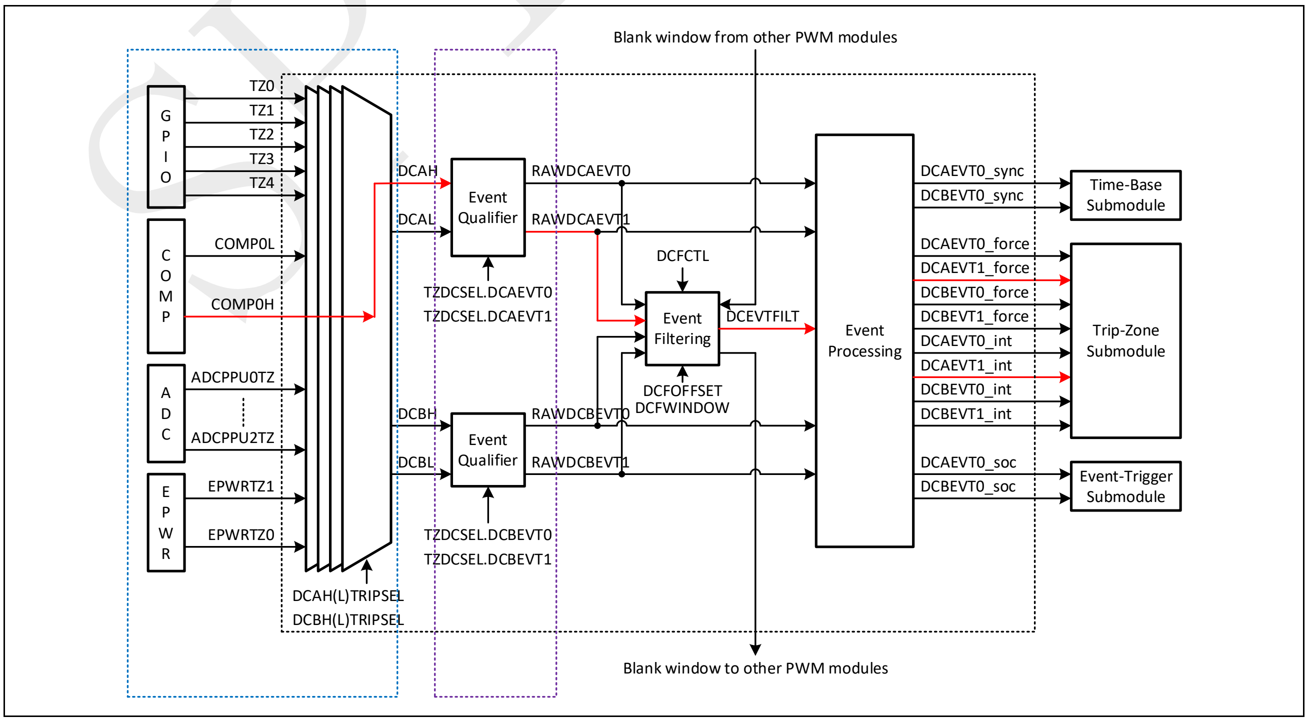
Use a COMP trigger signal to generate a TZ event, resulting in a one-time blocking of the output PWM waveform.

- Use COMP as an ADC sampling comparison function, taking PWM0 and PWM1 outputs as examples. When the sampled ADC value exceeds 3000 mV, a one-time blocking is applied to the currently output PWM0 and PWM1. The PWM stop is triggered by COMP0H.

3.3.3.2 Function implementation

1. Determine the trigger signal path:
   a) COMP0H can be converted into DCAEVT0_force, DCBEVT0_force, DCAEVT1_force, or DCBEVT1_force through the DC module, and then generate TZ events through CBC or OST, as shown in Figure 3-11.
   b) When using this signal path, set the control bits related to DCAEVT0_force, DCBEVT0_force, DCAEVT1_force, and DCBEVT1_force in TZACTL and TZBCTL to DO_NOTHING. Otherwise, DCAEVT0_force, DCBEVT0_force, DCAEVT1_force, and DCBEVT1_force will bypass CBC or OST and directly reach the Trip-Zone Arbitrator module.
2. Detailed description of signal filtering functionality:
   a) Signal filtering is performed through the DC module, as shown in Figure 3-12.
   b) The blue dashed box represents an event selector. By configuring the TRIPSEL register, a logical OR selection can be performed on the events on the left. The selected event can trigger any of the events on the right, such as DCAH(L) or DCBH(L).
   c) The purple dashed box contains two Event Qualifier units. One Event Qualifier unit is used to logically filter DCAH(L) to generate RAWDCxEVT0 or RAWDCxEVT11. Taking the generation of the RAWDCAEVT0 event as an example, the logical filtering functionality of the Event Qualifier unit is described in Table 3-6.
   d) Finally, the Event Filtering module performs window filtering on the RAWDCxEVT0 or RAWDCxEVT1 signals, and the Event Processing module generates the final output signal.
3. Signal filtering in this example:
   a) The signal path is shown by the red line in Figure 3-14.
   b) By configuring the DCAHTRIPSEL register, the COMP0H event triggers a DCAH event.
   c) By configuring the TZSCSEL.DCAEVT0 field, a RAWDCAEVT1 event is generated when DCAH is high.
   d) The Event Filtering module performs window filtering on the event signal.
   e) The Event Processing module ensures that the COMP0 event triggers the final output signals DCAEVT1_force and DCAEVT11_int.

Figure 3-14: DC Function Diagram



The example code for the above implementation steps can refer to the Demo provided in the SDK, such as:Table 3-8:

Table 3-8: Example 8 Code Path

Product Model	Code Path
SPC1168 Series, SPC2168 Series	SDK DIRECTORY\0_Examples\PWM_Independent_Trigger_TZ

3.4 PWM-Triggered ADC Sampling

3.4.1 Example 9: PWM-Triggered ADC Sampling

3.4.1.1 Functional requirements

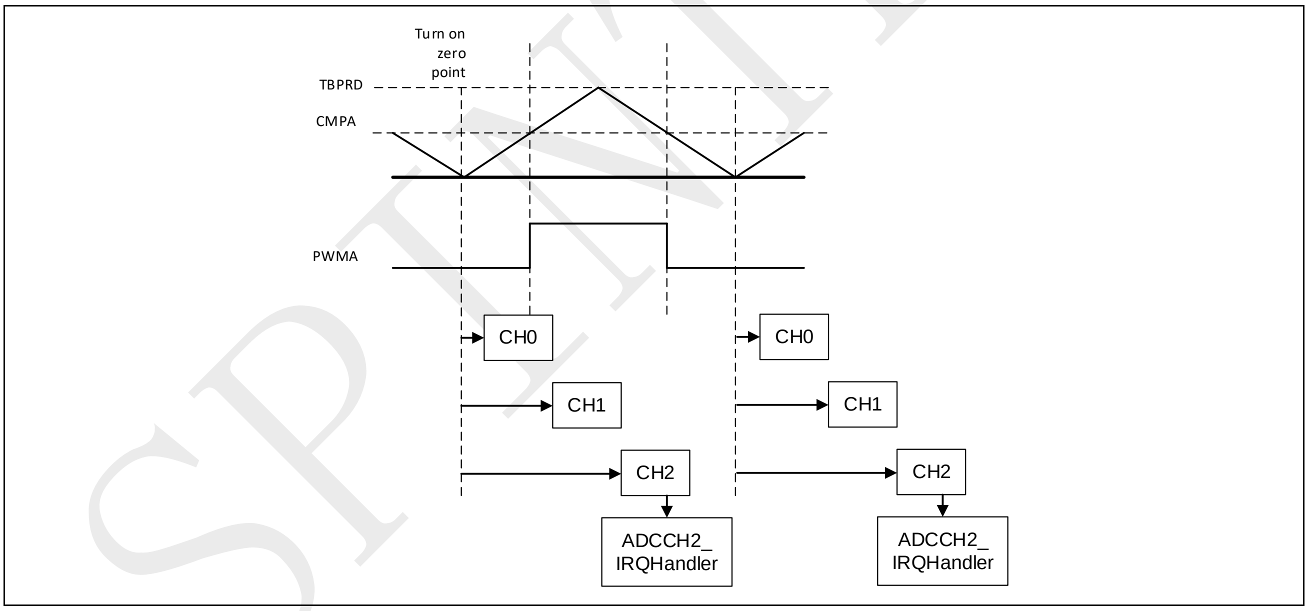
Implement PWM-triggered three-phase current sampling:

- PWM operates in up-down counting mode with a duty cycle of 25%.
- The three-phase current conversion is triggered when the PWM time-base counter reaches 0

3.4.1.2 Function implementation

1. Use PWM0 to generate the PWM waveform. Please refer to Section 3.1.
2. Configure PWM0 to trigger the ADC when TBCNT crosses zero, simultaneously triggering CH0~CH2 to convert the three-phase currents.
3. After CH2 completes the conversion, enter the interrupt service routine (ADCCH2_IRQHandler) to obtain the sampled values, as shown in Figure 3-15.

Figure 3-15: PWM-Triggered Three-Phase Current Sampling



The example code for the above implementation steps can refer to the Demo provided in the SDK, such as:Table 3-9:

Table 3-9: Example 9 Code Path

MCU PRODUCT MODEL	Code path
SPC1125 Series, SPC1169 Series, SPC2188 Series	SDK DIRECTORY \0_Examples\ PWM_Trigger_ADC_Sample_and_Signal_Observe
SPC1168 Series, SPC2168 Series	SDK DIRECTORY \0_Examples\ PWM_Trigger_ADC_Sample