# FA95/VA95 Non-OS PWM Library Reference Guide

V1.0

***Publication Release Date: Oct 2011***

**Support Chips:**
W55FA Series

**Support Platforms:**
Non-OS

# Table of Contents


**1. PWM Library ..... 4**

1.1. Programming Guide ..... 4

System Overview ..... 4
Block Diagram ..... 5
PWM Timer Control ..... 5
PWM Library Constant Definition ..... 8
PWM Library Property Definition ..... 9

1.2. PWM API ..... 10

PWM_Open ..... 10
PWM_Close ..... 10
PWM_SetClockSetting ..... 11
PWM_GetEngineClock ..... 11
PWM_SetTimerClk ..... 12
PWM_SetTimerIO ..... 13
PWM_Enable ..... 13
PWM_IsTimerEnabled ..... 14
PWM_SetTimerCounter ..... 14
PWM_GetTimerCounter ..... 15
PWM_EnableDeadZone ..... 16
PWM_EnableInt ..... 16
PWM_DisableInt ..... 17
PWM_InstallCallBack ..... 18
PWM_ClearInt ..... 18
PWM_GetIntFlag ..... 19
PWM_GetCaptureIntStatus ..... 19
PWM_ClearCaptureIntStatus ..... 20
PWM_GetRisingCounter ..... 20
PWM_GetFallingCounter ..... 21

**2. Revision History ..... 22**

# 1. PWM Library

This library is designed to make user application to set FA95/VA95 PWM more easily.
The PWM library has the following features:

- PWM signal frequency and duty setting
- PWM Capture function

## 1.1. Programming Guide

### *System Overview*

The W55FA95 have 4 channels pwm-timers. The 4 channels pwm-timers has 2 prescaler, 2 clock divider, 4 clock selectors, 4 16-bit counters, 4 16-bit comparators, 2 Dead-Zone generator. They are all driven by system clock. Each channel can be used as a timer and issue interrupt independently.
Each two channels pwm-timers share the same prescaler(channel0-1 share prescalar0 and channel2-3 share prescalar1). Clock divider provides each channel with 5 clock sources (1, 1/2, 1/4, 1/8, 1/16). Each channel receives its own clock signal from clock divider which receives clock from 8-bit prescaler. The 16-bit counter in each channel receive clock signal from clock selector and can be used to handle one pwm period. The 16-bit comparator compares number in counter with threshold number in register loaded previously to generate pwm duty cycle.

The W55FA95 have 4 channels pwm-timers and each pwm-timer includes a capture channel. The Capture 0 and PWM 0 share a timer that included in PWM 0; and the Capture 1 and PWM 1 share another timer, and etc. Therefore user must setup the PWM-timer before turn on Capture feature. After enabling capture feature, the capture always latched PWM-counter to CRLR when input channel has a rising transition and latched PWM-counter to CFLR when input channel has a falling transition. Capture channel 0 interrupt is programmable by setting CCR0[1] (Rising latch Interrupt enable) and CCR0[2] (Falling latch Interrupt enable) to decide the condition of interrupt occur. Capture channel 1 has the same feature by setting CCR0[17] and CCR0[18]. And capture channel 2 & 3 has the same feature by setting CCR1[1],CCR1[2] and CCR1[17], CCR1[18] respectively. Whenever Capture issues Interrupt 0/1/2/3, the PWM counter 0/1/2/3 will be reload at this moment.

There are only four interrupts from PWM to advanced interrupt controller (AIC). PWM 0 and Capture 0 share the same interrupt channel, PWM1 and Capture 1 share the same interrupt and so on. Therefore, PWM function and Capture function in the same channel cannot be used at the same time.

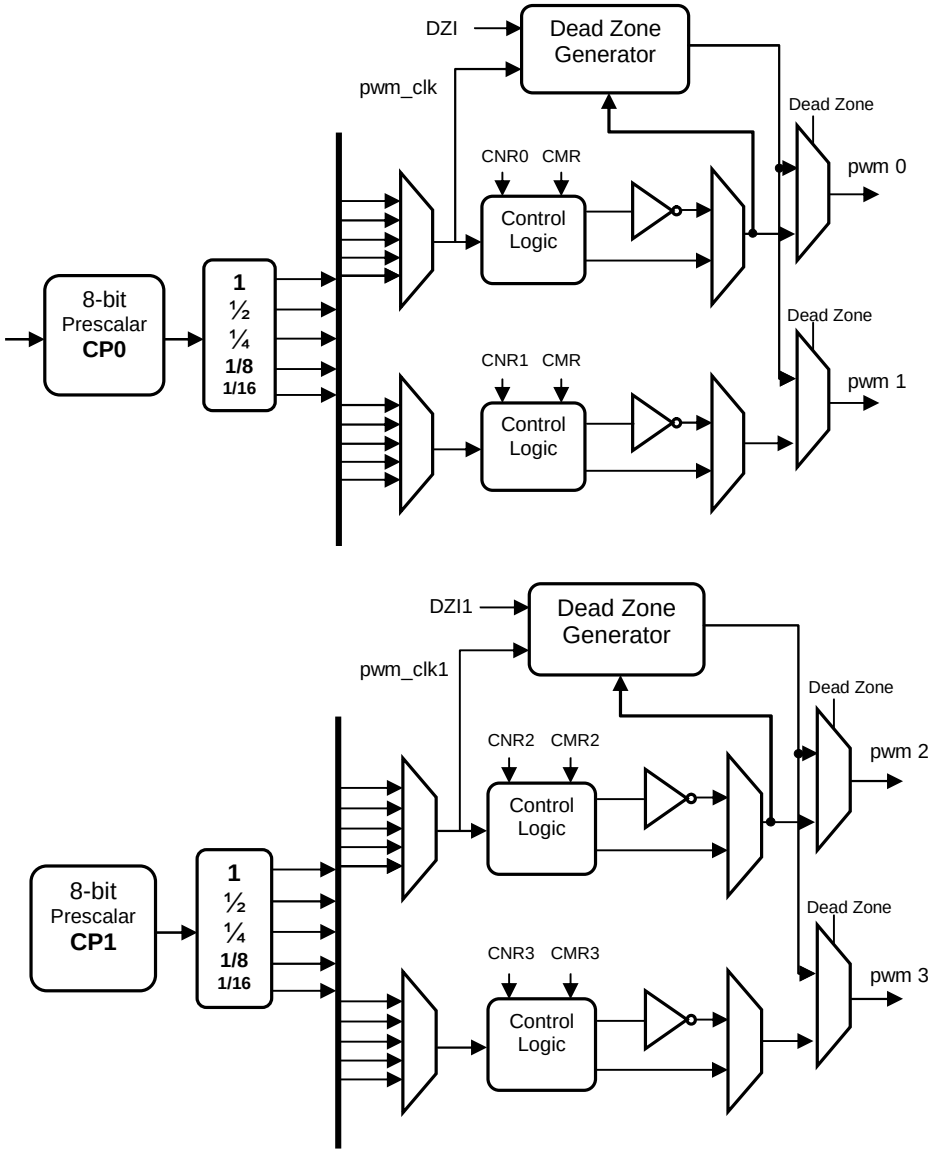
**PWM Features**

- Two 8-bit prescalers and Two clock dividers

- Four clock selectors
- Four 16-bit counters and four 16-bit comparators
- Two Dead-Zone generator
- Capture function

## *Block Diagram*

The following figure describes the architecture of pwm in one group. (channel0&1 are in one group and channel2&3 are in another group)




## *PWM Timer Control*

## n Prescaler and clock selector

The PWM has two groups (two channels in each group) of timers. The clock input of the group is according to the PWM Prescaler Register (**PPR**) value. The PWM prescaler divided the clock input by PPR+1 before it is fed to the counter. Please notice that when the PPR value equals zero, the prescaler output clock will stop. Furthermore, according to the PWM Clock Select Register (**CSR**) value, the clock input of PWM timer channel can be divided by 1,2,4,8 and 16.

Consider following examples, which explain the PWM timer period (Duty).

$$\text{period} = \frac{1}{(SourceClock) \div (PPR+1) \div CSR}$$

[Note 1]. PWM source clock can be APLL/UPLL/XIN.

When the PWM engine clock = 60 MHz, the maximum and minimum PWM timer counting period is described as follows.
Maximum period: PPR = 255 (since the length of PPR is 8bit) and CSR = 16

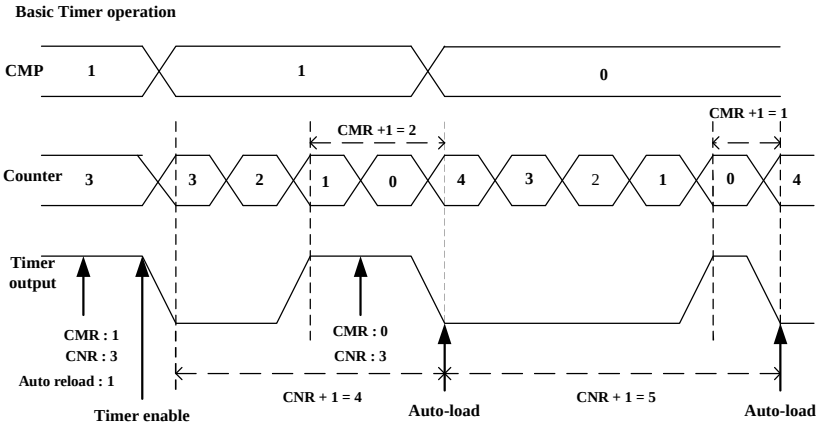
$$\text{period}_{\max} = \frac{1}{(60Mhz) \div (255+1) \div 16} = 68.266us$$

Minimum period: PCLK = 60 MHz, PPR=1 and CSR=1

$$\text{period}_{\min} = \frac{1}{(60Mhz) \div (1+1) \div 1} = 0.0333us$$

The maximum and minimum intervals between two interrupts depend on the $period_{max}$ , $period_{min}$ and PWM Counter Register(**CNRx**) length. The maximum interval between two interrupts is (65535)*(51.2us) since the length of CNR is 16bit. Please notice that the above calculation is based on the PCLK = 60MHz. Therefore, all of the values need to be recalculated when the PCLK is not equal to 60Mhz.

## n Basic Timer Operation



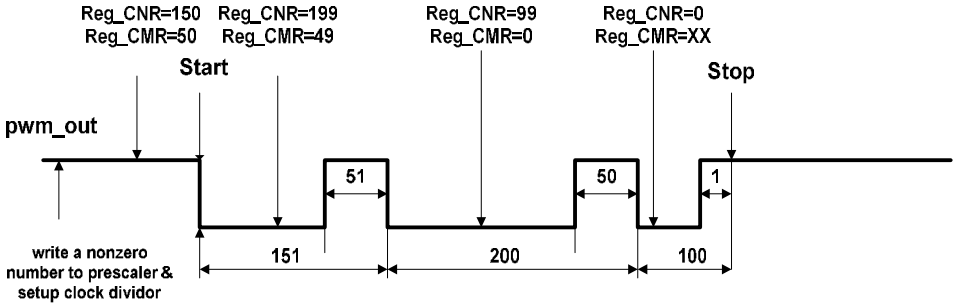
- **PWM Double Buffering and Automatic Reload**

W55FA95 PWM Timers have a double buffering function, enabling the reload value changed for next timer operation without stopping current timer operation. Although new timer value is set, current timer operation still operate successfully.

The counter value can be written into CNR0~3 and current counter value can be read from PDR0~3.

The auto-reload operation copies from CNR0~3 to down-counter when down-counter reaches zero. If CNR0~3 are set as zero, counter will be halt when counter count to zero. If auto-reload bit is set as zero, counter will be stopped immediately

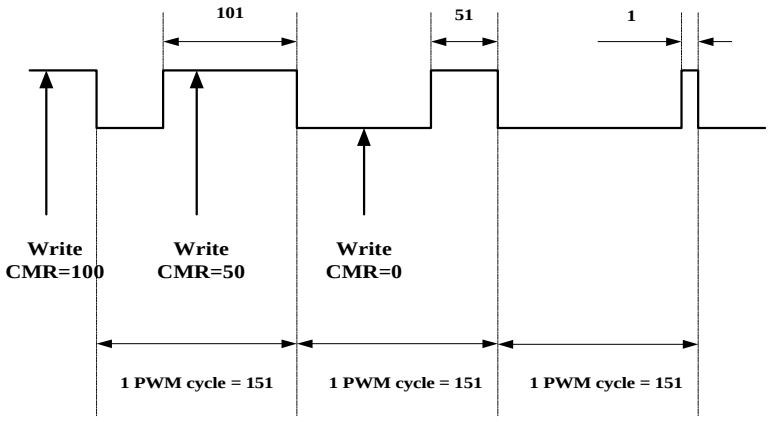
PWM double buffering




- **PWM Double Buffering and Automatic Reload**

The double buffering function allows CMR written at any point in current cycle. The loaded value will take effect from next cycle.

Modulate PWM controller ouput duty ratio(CNR = 150)

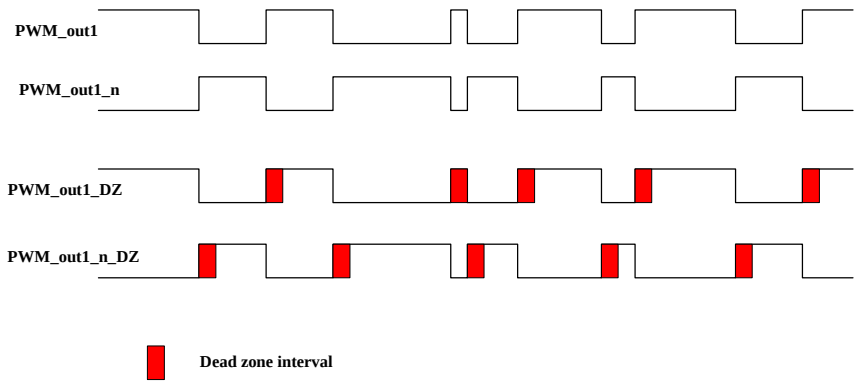



- **PWM Double Buffering and Automatic Reload**

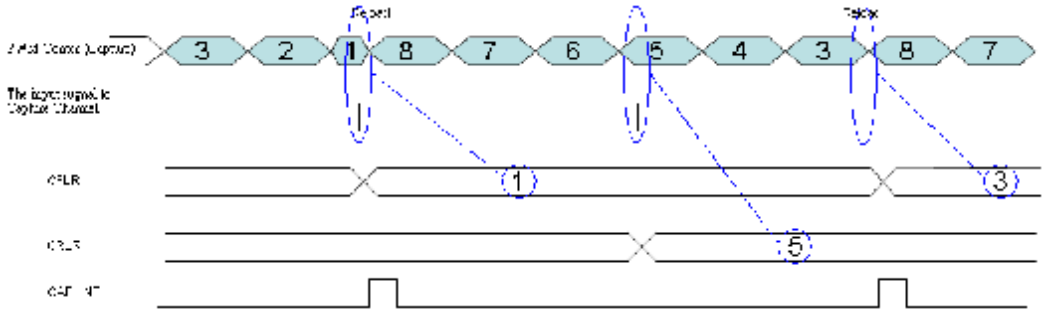
W55FA95 PWM is implemented with Dead Zone generator. They are built for power device protection. This function enables generation of a programmable time gap at the rising of PWM

output waveform. User can program PPR [31:24] and PPR [23:16] to determine the two Dead Zone interval respectively.

**Dead zone generator operation**




- **Capture Basic Timer Operation**




At this case, the CNR is 8:

1. When set falling interrupt enable, the pwm counter will be reload at time of interrupt occur.
2. The channel low pulse width is (CNR – CRLR).
3. The channel high pulse width is (CRLR - CFLR).
4. The channel cycle time is (CNR – CFLR).

## *PWM Library Constant Definition*

| Name | Value | Description |
|---|---|---|
| PWM_TIMER0 | 0x0 | PWM Timer 0 |
| PWM_TIMER1 | 0x1 | PWM Timer 1 |
| PWM_TIMER2 | 0x2 | PWM Timer 2 |
| PWM_TIMER3 | 0x3 | PWM Timer 3 |
| PWM_CAP0 | 0x0 | PWM Capture 0 |
| PWM_CAP1 | 0x1 | PWM Capture 1 |
| PWM_CAP2 | 0x2 | PWM Capture 2 |

| | | |
|---|---|---|
| PWM_CAP3 | 0x3 | PWM Capture 3 |
| PWM_CAP_NO_INT | 0 | No PWM Capture Interrupt |
| PWM_CAP_RISING_INT | 1 | PWM Capture Rising Interrupt |
| PWM_CAP_FALLING_INT | 2 | PWM Capture Falling Interrupt |
| PWM_CAP_RISING_FLAG | 6 | Capture rising interrupt flag |
| PWM_CAP_FALLING_FLAG | 7 | Capture falling interrupt flag |
| PWM_CLOCK_DIV_1 | 4 | Input clock divided by 1 |
| PWM_CLOCK_DIV_2 | 0 | Input clock divided by 2 |
| PWM_CLOCK_DIV_4 | 1 | Input clock divided by 4 |
| PWM_CLOCK_DIV_8 | 2 | Input clock divided by 8 |
| PWM_CLOCK_DIV_16 | 3 | Input clock divided by 16 |
| PWM_TOGGLE_MODE | TRUE | PWM Timer Toggle mode |
| PWM_ONE_SHOT_MODE | FALSE | PWM Timer One-shot mode |

### *PWM Library Property Definition*

The PWM library provides property structure to set PWM timer property.

| Name | Value | Description |
|---|---|---|
| ***u8Frequency*** | >= 0 | The timer/capture frequency[0] |
| ***u8HighPulseRatio*** | 1~100 | High pulse ratio |
| ***u8Mode*** | PWM_ONE_SHOT_MODE / PWM_TOGGLE_MODE | PWM Timer Trigger mode |
| ***bInverter*** | TRUE / FALSE | Inverter Enable / Inverter Disable |
| ***u8ClockSelector*** | PWM_CLOCK_DIV_1/ PWM_CLOCK_DIV_2/ PWM_CLOCK_DIV_4/ PWM_CLOCK_DIV_8/ PWM_CLOCK_DIV_16 | Clock Selector [1] |
| ***u16PreScale*** | 2 ~ 256 | Clock Prescale [1] |
| ***u32Duty*** | 0~65535 | Pulse duty [2] |

**[0] PWM provides two timer setting mode: Frequency-setting and Property-setting modes.**

- Frequency-setting mode (***u8Frequency*** > 0)

User doesn't need to set ***u8ClockSelector*** / ***u16PreScale*** / ***u32Duty*** fields. PWM library will set the proper values according to current APB clock automatically.

- Property-setting mode (***u8Frequency*** = 0)

- User must set ***u8ClockSelector*** / ***u16PreScale*** / ***u32Duty*** fields by himself. Please refer to the previous section "Prescaler and clock selector.

**[1] The value take effect only when Property-setting mode.**

**[2] The value takes effect when Property-setting mode or the Capture functions. It is the capture monitor period.**

## 1.2. PWM API

### *PWM_Open*

**Synopsis**

VOID PWM_Open (VOID)

**Description**

Enable PWM engine clock and reset PWM

**Parameter**

None

**Return Value**

None

**Example**

```
/* Enable PWM clock */
PWM_Open();
```

### *PWM_Close*

**Synopsis**

VOID PWM_Close (VOID)

**Description**

Disable PWM engine clock and the I/O enable

**Parameter**

None

**Return Value**

None

**Example**

/* Disable PWM clock */

PWM_Close();

## *PWM_SetClockSetting*

**Synopsis**

BOOL PWM_SetClockSetting(E_SYS_SRC_CLK eSrcClk, UINT32 u32PllDiver, UINT32 u32EngineDiver)

**Description**

This function is used to set PWM engine clock source and divie

**Parameter**

| | |
|---|---|
| eSrcClk | PWM clock source.<br>It could be eSYS_EXT=0, eSYS_APLL= 2 and eSYS_UPLL = 3. |
| u32PllDiver | PWM PLL Divider Selection (1~8)<br>() |
| u32EngineDiver | Engine Clock divider (1~256) |

**Return Value**

| | |
|---|---|
| TRUE | - Success. |
| FALSE | - Setting Fail.. |

**Note**

1. Parameter “u32PllDiver” is only be valid when eSrcClk is eSYS_APLL or eSYS_UPLL

**Example**

/* PWM Egine clock is UPLL / 4, and Engine Clock divider is 2 */

PWM_SetClockSetting(eSYS_UPLL, 4, 2);

## *PWM_GetEngineClock*

**Synopsis**

UINT32 PWM_GetEngineClock(E_SYS_SRC_CLK* peSrcClk)

**Description**

This function is used to get Current PWM engine clock

**Parameter**

peSrcClk Sytem clock source.

It could be eSYS_EXT=0, eSYS_APLL= 2 and eSYS_UPLL = 3.

**Return Value**

PWM Engine Clock (Hz)

**Example**

```
u32PWMClock = PWM_GetEngineClock(&eSrcClk) ;
sysprintf("PWM    Clock Source is ");
switch(eSrcClk)
{
        case eSYS_EXT:
        sysprintf("External Crystal\n");
                break;
        case eSYS_APLL:
                sysprintf("APLL\n");
                break;
        case eSYS_UPLL:
                sysprintf("UPLL\n");
                break;
}
sysprintf("PWM Clock is %dHz\n",u32PWMClock);
```

## *PWM_SetTimerClk*

**Synopsis**

FLOAT PWM_SetTimerClk (UINT8 u8Timer, PWM_TIME_DATA_T *sPt)

**Description**

This function is used to configure the frequency/pulse/mode/inverter function

**Parameter**

u8Timer The function to be set

PWM_TIMER0 ~ PWM_TIMER3: PWM timer 0 ~ 3

PWM_CAP0 ~ PWM_CAP3: PWM capture 0 ~ 3

sPt PWM property information

**Return Value**

= 0 - Setting Fail.

> 0 - Success. The actual frequency by PWM timer.

**Note**

2. The function will set the frequency property automatically (It will change the parameters to the values that it sets to hardware) when user set a nonzero frequency value
3. The function can set the proper frequency property (Clock selector/Prescale) for capture function and user needs to set the proper pulse duty by himself.

**Example**

```
//* Set PWM Timer 0 Configuration */
PWM_SetTimerClk(PWM_TIMER0,&sPt);
```

## *PWM_SetTimerIO*

**Synopsis**

VOID PWM_SetTimerIO (UINT8 u8Timer, BOOL bEnable)

**Description**

This function is used to enable/disable PWM timer/capture I/O function

**Parameter**

u8Timer The function to be set

PWM_TIMER0 ~ PWM_TIMER3: PWM timer 0 ~ 3

PWM_CAP0 ~ PWM_CAP3: PWM capture 0 ~ 3

bEnable Enable (TRUE) / Disable (FALSE)

**Return Value**

None

**Example**

```
/* Enable Output for PWM Timer 0 */
PWM_SetTimerIO(PWM_TIMER0,TRUE);
```

## *PWM_Enable*

**Synopsis**

VOID PWM_Enable (UINT8 u8Timer, BOOL bEnable)

**Description**

This function is used to enable PWM timer / capture function

**Parameter**

| | |
|---|---|
| u8Timer | The function to be set<br>PWM_TIMER0 ~ PWM_TIMER3: PWM timer 0 ~ 3<br>PWM_CAP0 ~ PWM_CAP3: PWM capture 0 ~ 3 |
| bEnable | Enable (TRUE) / Disable (FALSE) |

**Return Value**

None

**Example**

```
/* Enable Interrupt Sources of PWM Timer 0 and install call back function */
PWM_EnableInt(PWM_TIMER0,0);
```

## *PWM_IsTimerEnabled*

**Synopsis**

BOOL PWM_IsTimerEnabled (UINT8 u8Timer)

**Description**

This function is used to get PWM specified timer enable/disable state

**Parameter**

| | |
|---|---|
| u8Timer | The function to be set<br>PWM_TIMER0 ~ PWM_TIMER3: PWM timer 0 ~ 3 |

**Return Value**

| | |
|---|---|
| TURE | - The specified timer is enabled. |
| FALSE | - The specified timer is disabled. |

**Example**

```
/*    Check PWM Timer0 is enabled or not */
If (PWM_IsTimerEnabled(PWM_TIMER0))
        sysprintf("PWM Timer 0 is enabled\n");
else
        sysprintf("PWM Timer 0 isn't enabled\n");
```

## *PWM_SetTimerCounter*

**Synopsis**

VOID PWM_SetTimerCounter (UINT8 u8Timer, UINT16 u16Counter)

**Description**

This function is used to set the PWM specified timer counter

**Parameter**

| | |
|---|---|
| u8Timer | The function to be set<br>PWM_TIMER0 ~ PWM_TIMER3: PWM timer 0 ~ 3 |
| u16Counter | The timer value. (0~65535) |

**Return Value**

None

**Note**

If the counter is set to 0, the timer will stop.

**Example**

```
/* Set PWM Timer 0 counter as 0 */
PWM_SetTimerCounter(PWM_TIMER0,0);
```

## *PWM_GetTimerCounter*

**Synopsis**

UINT32 PWM_GetTimerCounter (UINT8 u8Timer)

**Description**

This function is used to get the PWM specified timer counter value

**Parameter**

| | |
|---|---|
| u8Timer | The function to be set<br>PWM_TIMER0 ~ PWM_TIMER3: PWM timer 0 ~ 3 |
| u16Counter | The timer value. (0~65535) |

**Return Value**

The specified timer-counter value

**Example**

```
/* Loop when Counter of PWM Timer0 isn't 0 */
while(PWM_GetTimerCounter(PWM_TIMER0));
```

### *PWM_EnableDeadZone*

**Synopsis**

VOID PWM_EnableDeadZone (UINT8 u8Timer, UINT8 u8Length, BOOL bEnableDeadZone)

**Description**

This function is used to set the dead zone length and enable/disable Dead Zone function

**Parameter**

| | |
|---|---|
| u8Timer | The function to be set<br>PWM_TIMER0 ~ PWM_TIMER3: PWM timer 0 ~ 3 |
| u8Length | Dead Zone Length : 0~255 |
| bEnableDeadZone | Enable DeadZone (TRUE) / Disable DeadZone (FALSE) |

**Return Value**

None

**Note**

1. If Deadzone for PWM_TIMER0 or PWM_TIMER1 is enabled, the output of PWM_TIMER1 is inverse waveform of PWM_TIMER0.
2. If Deadzone for PWM_TIMER2 or PWM_TIMER3 is enabled, the output of PWM_TIMER3 is inverse waveform of PWM_TIMER2.

**Example**

```
/* Enable Deadzone of PWM Timer 0 and set it to 100 units*/
PWM_EnableDeadZone(PWM_TIMER0, 100, TRUE)
```

### *PWM_EnableInt*

**Synopsis**

VOID PWM_EnableInt (UINT8 u8Timer, UINT8 u8Int)

**Description**

This function is used to enable the PWM timer/capture interrupt

**Parameter**

| | |
|---|---|
| u8Timer | The function to be set<br>PWM_TIMER0 ~ PWM_TIMER3: PWM timer 0 ~ 3<br>PWM_CAP0 ~ PWM_CAP3: PWM capture 0 ~ 3 |

u8Int Capture interrupt type (The parameter is valid only when capture function)

PWM_CAP_RISING_INT: The capture rising interrupt.

PWM_CAP_FALLING_INT: The capture falling interrupt.

PWM_CAP_ALL_INT: All capture interrupt.

**Return Value**

None

**Example**

```
/* Enable Interrupt Sources of PWM Timer 0 */
PWM_EnableInt(PWM_TIMER0,0);
/* Enable Interrupt Sources of PWM Capture3 */
PWM_EnableInt(PWM_CAP3, PWM_CAP_FALLING_INT);
```

### *PWM_DisableInt*

**Synopsis**

VOID PWM_DisableInt (UINT8 u8Timer, UINT8 u8Int)

**Description**

This function is used to disable the PWM timer/capture interrupt

**Parameter**

u8Timer The function to be set

PWM_TIMER0 ~ PWM_TIMER3: PWM timer 0 ~ 3

PWM_CAP0 ~ PWM_CAP3: PWM capture 0 ~ 3

u8Int Capture interrupt type (The parameter is valid only when capture function)

PWM_CAP_RISING_INT: The capture rising interrupt.

PWM_CAP_FALLING_INT: The capture falling interrupt.

PWM_CAP_ALL_INT: All capture interrupt.

**Return Value**

None

**Example**

```
/* Disable Capture Interrupt */
PWM_DisableInt(PWM_CAP3,PWM_CAP_ALL_INT);
```

### *PWM_InstallCallBack*

**Synopsis**

VOID PWM_InstallCallBack (UINT8 u8Timer, PFN_PWM_CALLBACK pfncallback, PFN_PWM_CALLBACK *pfnOldcallback)

**Description**

This function is used to install the specified PWM timer/capture interrupt call back function

**Parameter**

| | |
|---|---|
| u8Timer | The function to be set<br>PWM_TIMER0 ~ PWM_TIMER3: PWM timer 0 ~ 3<br>PWM_CAP0 ~ PWM_CAP3: PWM capture 0 ~ 3 |
| pfncallback | The callback function pointer for specified timer / capture. |
| pfnOldcallback | The previous callback function pointer for specified timer / capture. |

**Return Value**

None

**Example**

```
/* Install Callback function */
PWM_InstallCallBack(PWM_TIMER0, PWM_PwmIRQHandler, &pfnOldcallback);
```

### *PWM_ClearInt*

**Synopsis**

VOID PWM_ClearInt (UINT8 u8Timer)

**Description**

This function is used to clear the PWM timer/capture interrupt.

**Parameter**

| | |
|---|---|
| u8Timer | The function to be set<br>PWM_TIMER0 ~ PWM_TIMER3: PWM timer 0 ~ 3<br>PWM_CAP0 ~ PWM_CAP3: PWM capture 0 ~ 3 |

**Return Value**

None

**Example**

```
/* Clear the PWM Capture 3 Interrupt */
PWM_ClearInt(PWM_CAP3);
```

### *PWM_GetIntFlag*

**Synopsis**

BOOL PWM_GetIntFlag (UINT8 u8Timer)

**Description**

This function is used to get the PWM timer/capture interrupt flag

**Parameter**

u8Timer The function to be set

PWM_TIMER0 ~ PWM_TIMER3: PWM timer 0 ~ 3

PWM_CAP0 ~ PWM_CAP3: PWM capture 0 ~ 3

**Return Value**

TRUE - The specified interrupt occurs.

FLASE - The specified interrupt doesn't occur.

**Example**

/* Get PWM Timer 0 Interrupt flag*/

PWM_GetIntFlag(PWM_TIMER0);

### *PWM_GetCaptureIntStatus*

**Synopsis**

VOID PWM_GetCaptureIntStatus (UINT8 u8Capture, UINT8 u8IntType)

**Description**

Check if there's a rising / falling transition

**Parameter**

u8Timer The function to be set

PWM_CAP0 ~ PWM_CAP3: PWM capture 0 ~ 3

u8Int Capture interrupt type (The parameter is valid only when capture function)

PWM_CAP_RISING_INT: The capture rising interrupt.

PWM_CAP_FALLING_INT: The capture falling interrupt.

**Return Value**

None

**Example**

```
/* Wait for Interrupt Flag (Falling) */
while(PWM_GetCaptureIntStatus(PWM_CAP0, PWM_CAP_FALLING_FLAG)!=TRUE);
```

### *PWM_ClearCaptureIntStatus*

**Synopsis**

VOID PWM_ClearCaptureIntStatus (UINT8 u8Capture, UINT8 u8IntType)

**Description**

Clear the rising / falling transition interrupt flag

**Parameter**

| | |
|---|---|
| u8Timer | The function to be set<br>PWM_CAP0 ~ PWM_CAP3: PWM capture 0 ~ 3 |
| u8Int | Capture interrupt type (The parameter is valid only when capture function)<br>PWM_CAP_RISING_INT: The capture rising interrupt.<br>PWM_CAP_FALLING_INT: The capture falling interrupt. |

**Return Value**

None

**Example**

```
/* Clear the Capture Interrupt Flag */
PWM_ClearCaptureIntStatus(PWM_CAP0, PWM_CAP_FALLING_FLAG);
```

### *PWM_GetRisingCounter*

**Synopsis**

UINT16 PWM_GetRisingCounter (UINT8 u8Capture)

**Description**

The value which latches the counter when there’s a rising transition

**Parameter**

| | |
|---|---|
| u8Timer | The function to be set<br>PWM_CAP0 ~ PWM_CAP3: PWM capture 0 ~ 3 |

**Return Value**

This function is used to get value which latches the counter when there’s a rising transition

**Example**

```
/* Get the Rising Counter Data */
```

u32Count[u32i++] = PWM_GetRisingCounter(PWM_CAP0);

### *PWM_GetFallingCounter*

**Synopsis**

UINT16 PWM_GetFallingCounter (UINT8 u8Capture)

**Description**

The value which latches the counter when there's a falling transition

**Parameter**

u8Timer The function to be set

PWM_CAP0 ~ PWM_CAP3: PWM capture 0 ~ 3

**Return Value**

This function is used to get value which latches the counter when there's a falling transition

**Example**

/* Get the Falling Counter Data */

u32Count[u32i++] = PWM_GetFallingCounter(PWM_CAP0);

# 2. Revision History

| Version | Date | Description |
|---|---|---|
| V1 | Oct. 19, 2011 | ▪ Created |

## Important Notice

**Nuvoton products are not designed, intended, authorized or warranted for use as components in equipment or systems intended for surgical implantation, atomic energy control instruments, aircraft or spacecraft instruments, transportation instruments, traffic signal instruments, combustion control instruments, or for any other applications intended to support or sustain life. Furthermore, Nuvoton products are not intended for applications whereby failure could result or lead to personal injury, death or severe property or environmental damage.**

**Nuvoton customers using or selling these products for such applications do so at their own risk and agree to fully indemnify Nuvoton for any damages resulting from their improper use or sales.**