Application Note

32-bit Cortex™-M0 MCU
NuMicro® Family

How to use PWM in NUC1xx?

Table of Contents-

1 INTRODUCTION

This document explains how to use PWM Timer to create a specified frequency waveform and output to a buzzer, and use PWM Capture function to capture the property of input waveform. The PWM module has up to 4 sets of PWM generators which can be configured as 8 independent PWM outputs, PWM0~7, or 4 complementary PWM pairs, (PWM0, PWM1), (PWM2, PWM3), (PWM4, PWM5), (PWM6, PWM7) with 4 programmable dead-zone generators. The alternate feature of the PWM-timer is input Capture function. After Capture feature is enabled, the capture latches PWM counter to Capture Rising Latch Register when input channel has a rising transition and latches PWM counter to Capture Falling Latch Register when input channel has a falling transition. The Capture0 and PWM0 share one PWM-timer which is included in PWM0; and Capture1 and PWM1 share PWM1 timer, and etc. Therefore, PWM function and Capture function in the same channel can not be used at the same time.

1.1 PWM Function Features

- Four PWM generators, each one supports one 8-bit pre-scalar, one clock divider, PWM Timer, one dead-zone generator and two outputs.
- Up to 8 PWM channels or 4 PWM paired channels.
- 16 bits resolution.
- One-shot mode and Auto-load mode..
- Double buffer to avoid glitch at PWM output.

1.2 Capture Function Features

- Timer control logic shared with PWM generator. Therefore, the PWM output pin is switched as capture input mode If Capture function is enabled.
- 8 Capture input channels.
- Each channel support one rising latch register, one falling latch register and Capture interrupt flag.

1.3 Structure

The following figures illustrate the architecture of PWM in groups. PWM timer 0&1 are in one group and PWM timer 2&3 are in one, and so on.

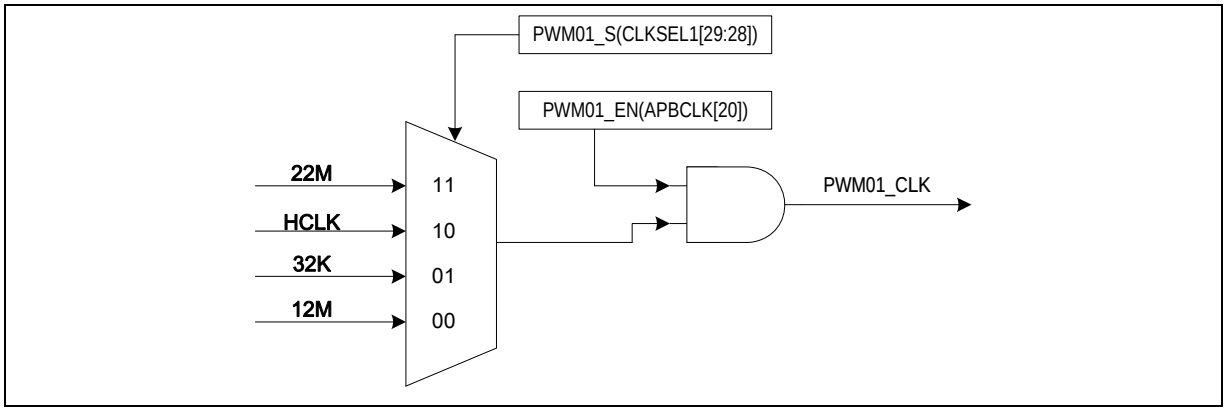

Figure 1 PWM 0&1 Clock Source Control

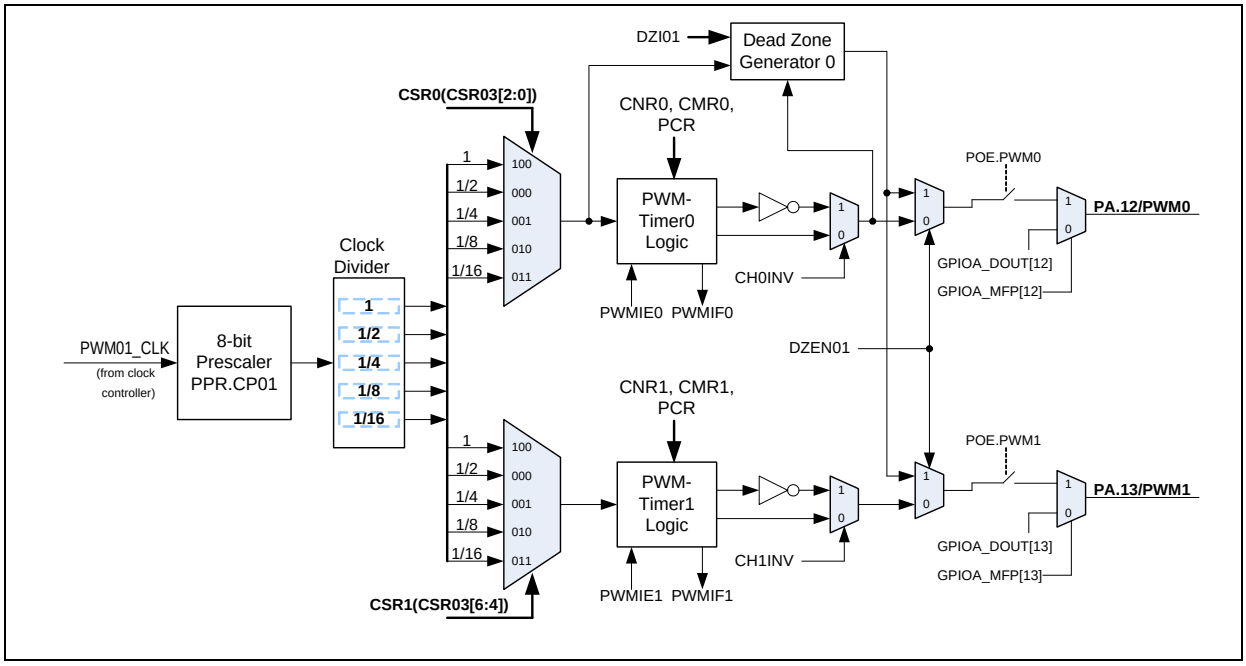

Figure 2 PWM 0&1 Architecture Diagram

1.4 PWM Operation

1.4.1 PWM-Timer Operation

The PWM period and duty control are decided by PWM down-counter register (CNR) and PWM comparator register (CMR). The PWM-timer timing operation is shown in Figure 4. The pulse width modulation follows the formula as below and the legend of PWM-timer comparator is shown as Figure 3. Note that the corresponding GPIO pins must be configured as PWM output.

- PWM frequency = PWMxy_CLK/(prescale+1)*(clock divider)/(CNR+1); where xy, could be 01, 23, 45 or 67, depends on selected PWM channel.
- Duty ratio = (CMR+1) / (CNR+1).
- CMR >= CNR: PWM output is always high.
- CMR < CNR: PWM low width= (CNR-CMR) unit[1]; PWM high width = (CMR+1) unit.
- If CMR = 0: PWM low width = (CNR) unit; PWM high width = 1 unit

Note: 1. Unit = one PWM clock cycle.

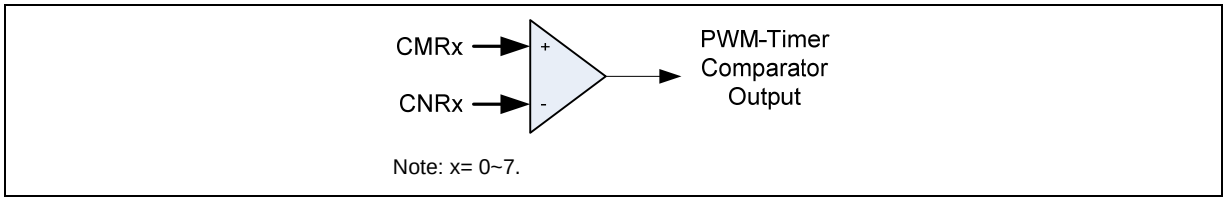

Figure 3 Internal Comparator Output of PWM-Timer

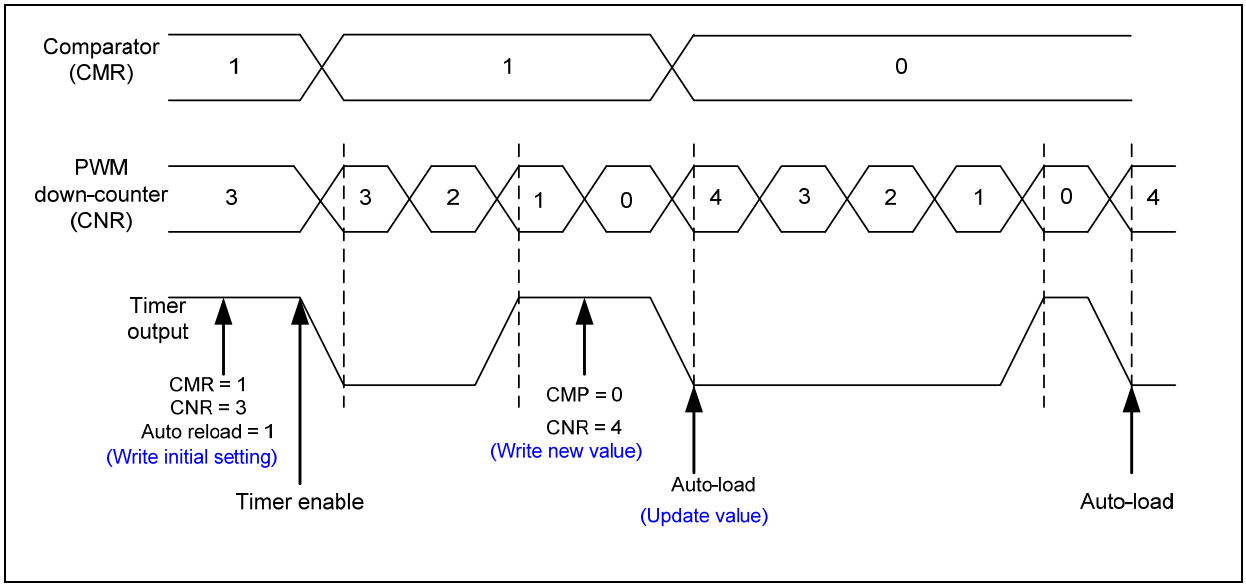

Figure 4 PWM-Timer Operation Timing

1.4.2 PWM Double Buffering, Auto-reload and One-shot Operation

NUC1XX series PWM Timers have double buffering function. The reload value is updated at the start of next period without affecting current timer operation. The PWM counter value can be written into CNR0~7 and current PWM counter value can be read from PDR0~7.

The bit CH0MOD in PWM Control Register (PCR) defines PWM0 operates in auto-reload or one-shot mode If CH0MOD is set to one, the auto-reload operation loads CNR0 to PWM counter when PWM counter reaches zero. If CNR0 are set to zero, PWM counter will be halt when PWM counter counts to zero. If CH0MOD is set as zero, counter will be stopped immediately. PWM1~7 performs the same function as PWM0.

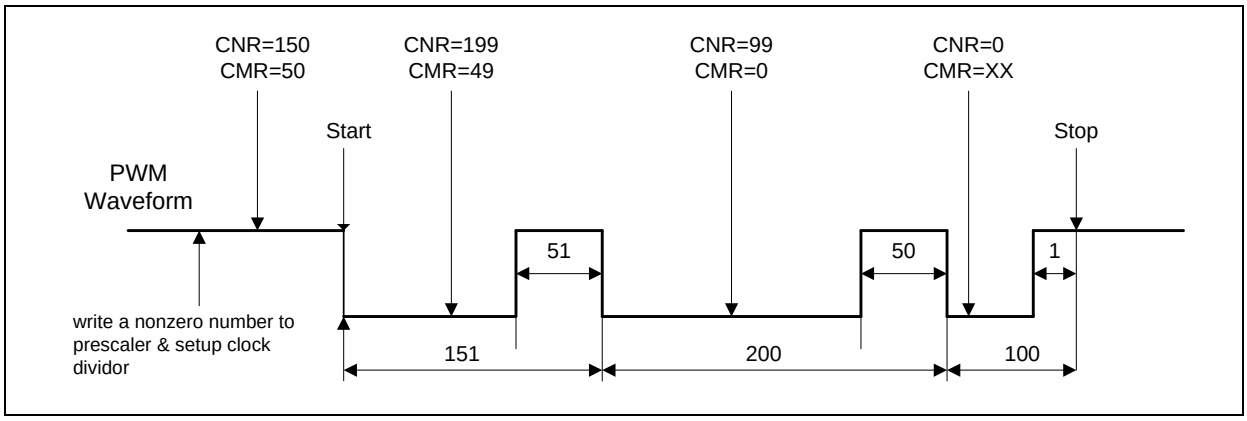

Figure 5 PWM Double Buffering Illustration

1.4.3 Modulate Duty Ratio

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

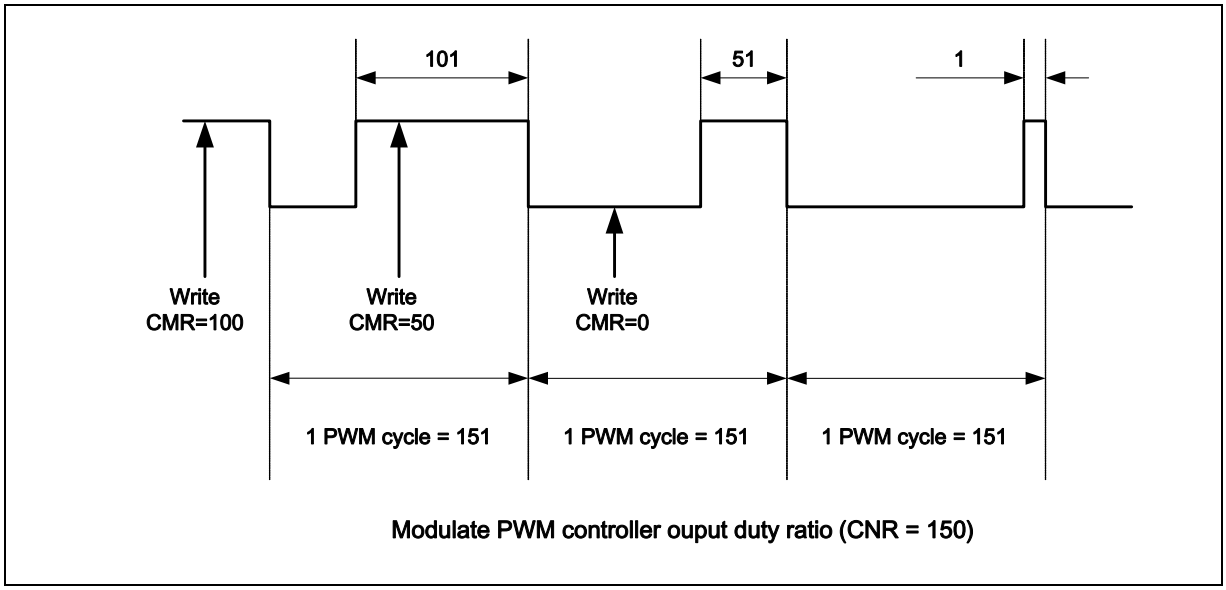

Figure 6 PWM Controller Output Duty Ratio

1.4.4 Dead-Zone Generator

NUC1XX series PWM is implemented with Dead Zone generator. They are built for power device protection. This function generates a programmable time gap to delay PWM rising output. User can rogram PPRx.DZI to determine the Dead Zone interval.

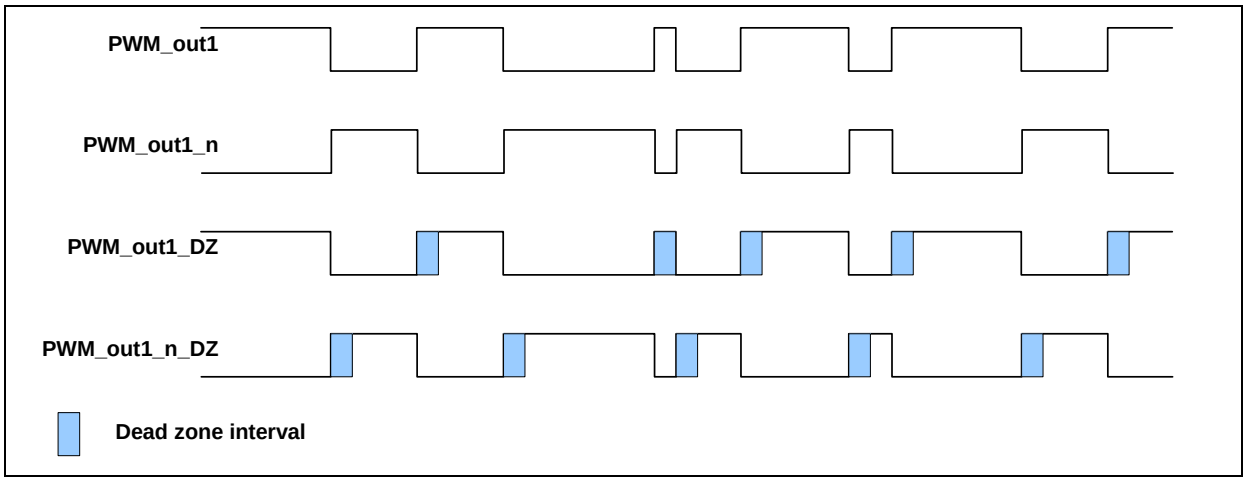

Figure 7 Dead Zone Generation Operations

1.4.5 Capture Operation

The Capture 0 and PWM 0 share one timer that included in PWM 0; and the Capture 1 and PWM 1 share another timer, and etc. The capture always latches PWM-counter to **CRLR** when input channel

has a **rising** transition and latches 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 etc. Whenever Capture issues Interrupt 0/1/2/3, the PWM counter 0/1/2/3 will be reload at this moment. Note that the corresponding GPIO pins must be configured as input type (GPIOA_PMD) before Capture function is enabled.

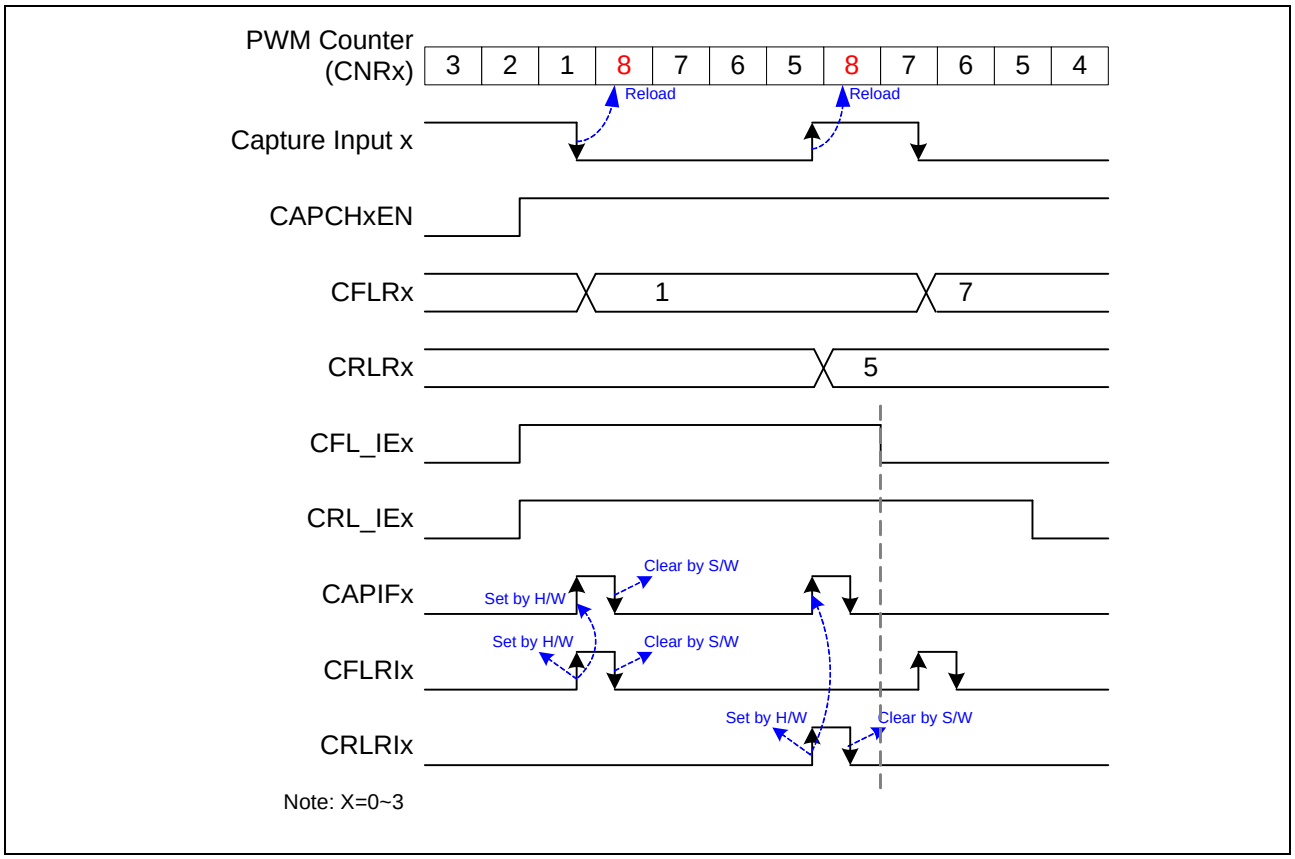

Figure 1-1 Capture Operation Timing

At this case, the CNR is 8:

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

1.4.6 PWM-Timer Start Procedure

The following procedure is recommended for starting a PWM drive.

1. Setup clock selector (CSR)
2. Setup prescaler (PPR)
3. Setup inverter on/off, dead zone generator on/off, auto-reload/one-shot mode, and PWM-Timer off. (PCR)

4. Setup comparator register (CMR) for setting PWM duty.
5. Setup PWM counter register (CNR) for setting PWM period.
6. Setup interrupt enable register (PIER)
7. Setup PWM output enable (POE)
8. Setup the corresponding GPIO pins to PWM function.
9. Enable PWM timer (PCR)

1.4.7 PWM-Timer Stop Procedure

Method 1:

Set 16-bit down counter (CNR) as 0, and monitor PDR. When PDR reaches to 0, disable PWM-Timer (CHxEN in PCR). ***(Recommended)***

Method 2:

Set 16-bit down counter (CNR) as 0. When interrupt request happen, disable PWM-Timer (CHxEN in PCR). ***(Recommended)***

Method 3:

Disable PWM-Timer directly ((CHxEN in PCR). ***(Not recommended)***

1.4.8 Capture Start Procedure

1. Setup clock selector (CSR)
2. Setup prescaler (PPR)
3. Setup channel enabled, rising/falling interrupt enable and input signal inverter on/off (CCR0, CCR1)
4. Setup PWM counter register (CNR)
5. Set Capture Input Enable Register (CAPENR)
6. Setup the corresponding GPIO pins to PWM function
7. Enable PWM timer (PCR)

2 CODE SECTION

2.1 Main Function (in Smpl_DrvPWM.c)

Refer chapter of Calling Sequence for its calling sequence.

The following processing presents two main function of PWM-timer: PWM function and Capture function. First, there are 7 specified frequency waveforms to buzzer from PWM0. Secondly, PWM3 as capture operation to capture the waveform property of PWM1 output and calculate the low pulse width, high pulse width and cycle time of PWM1 waveform.

```
int32_t main (void)
{
        S_DRVPWM_TIME_DATA_T sPt;
        STR_UART_T sParam;
        uint8_t u8Item;
        uint8_t u8Timer, u8CapTimer;
        int32_t i32Loop = 1;
        int32_t i32TestLoop = 1;

/*------------------------------Deleted-----------------------------*/

        /* Enable PWM clock */
        DrvPWM_Open();

        /* Set PWM pins */
        DrvGPIO_InitFunction(FUNC_PWM01);
        DrvGPIO_InitFunction(FUNC_PWM23);
        DrvGPIO_InitFunction(FUNC_PWM45);
        DrvGPIO_InitFunction(FUNC_PWM67);

        UNLOCKREG();
        DrvSYS_SetHCLKSource(0);
        LOCKREG();

/*------------------------------Deleted-----------------------------*/


                /* PWM Timer property */
```

```
                sPt.u8Mode = DRVPWM_TOGGLE_MODE;
                sPt.u32Frequency = g_u16Frequency;
                sPt.u8HighPulseRatio = 1;       /* High Pulse peroid : Total Pulse
peroid = 1 : 100 */
                sPt.i32Inverter = 0;
                u8Timer = DRVPWM_TIMER0;


                /* Select PWM engine clock */

               DrvPWM_SelectClockSource(u8Timer, DRVPWM_HCLK);

                /* Set PWM Timer0 Configuration */
                DrvPWM_SetTimerClk(u8Timer, &sPt);

                /* Enable Output for PWM Timer0 */
                DrvPWM_SetTimerIO(u8Timer, 1);

                /* Enable Interrupt Sources of PWM Timer0 and install call back
function */
                DrvPWM_EnableInt(u8Timer, 0, DRVPWM_PwmIRQHandler);


                /* Enable the PWM Timer 0 */
                DrvPWM_Enable(u8Timer, 1);

                while (g_u8PWMCount);


                /*---------------------------------------------------------------------------------*/
                /* Stop PWM Timer0 (Recommended procedure method 2)        */
                /* Set PWM Timer counter as 0, When interrupt request happen,
disable PWM Timer                       */
                /* Set PWM Timer counter as 0 in Call back function                */

                /*---------------------------------------------------------------------------------*/
                /* Disable the PWM Timer 0 */
                DrvPWM_Enable(u8Timer, 0);
```

```
    printf("PWM Capture Test\n");
    printf("Use PWM Capture 3 to capture the PWM Timer1 Waveform\n");

    /*---------------------------------------------------------------------------------------------------*/
    /* Set the PWM Timer1 as output function.                                                            */
    /*---------------------------------------------------------------------------------------------------*/

    /* PWM Timer property for Output waveform */
    sPt.u8Mode = DRVPWM_TOGGLE_MODE;
    sPt.u32Frequency = 250;                         /* 250 Hz */
    sPt.u8HighPulseRatio = 30;              /* High Pulse peroid : Total Pulse peroid = 30 : 100 */
    sPt.i32Inverter = 0;
    u8Timer          = DRVPWM_TIMER1;


    /* Select PWM01 engine clock */
    DrvPWM_SelectClockSource(u8Timer, DRVPWM_HCLK);

    /* Set PWM Timer1 Configuration */
    DrvPWM_SetTimerClk(u8Timer, &sPt);

    /* Enable Output for PWM Timer1 */
    DrvPWM_SetTimerIO(u8Timer, 1);


    /* Enable the PWM Timer1 */
    DrvPWM_Enable(u8Timer, 1);

    /*---------------------------------------------------------------------------------------------------*/
    /* Set the PWM Capture 3 for capture function                                                        */
    /*---------------------------------------------------------------------------------------------------*/

    /* PWM Timer property for Capture */
```

```
        sPt.u8Mode = DRVPWM_TOGGLE_MODE;
        sPt.u32Frequency = 100;                 /* Set the proper
frequency to capture data (Less than the input data)*/
        sPt.u8HighPulseRatio = 50;              /* High Pulse peroid : Total
Pulse peroid = 50 : 100 (Set a non-zero value) */
        sPt.u32Duty = 0x10000;                  /* Set the counter to the
maximum value */
        sPt.i32Inverter = 0;
        u8CapTimer = DRVPWM_CAP3;

        /* Select PWM engine clock */
        DrvPWM_SelectClockSource(u8CapTimer, DRVPWM_HCLK);

        /* Set PWM Timer 3 for Capture */
        DrvPWM_SetTimerClk(u8CapTimer, &sPt);

        /* Enable Interrupt Sources of PWM Capture3 */
        DrvPWM_EnableInt(u8CapTimer, DRVPWM_CAP_FALLING_INT,
NULL);

        /* Enable Input function for PWM Capture 3 */
        DrvPWM_SetTimerIO(u8CapTimer, 1);

        /* Enable the PWM Capture3 */
        DrvPWM_Enable(u8CapTimer, 1);

        /* Capture the Input Waveform Data */
        CalPeriodTime(u8CapTimer);

        /*---------------------------------------------------------------------------------------*/
        /* Stop PWM Timer 1 (Recommended procedure method 1)         */
        /* Set PWM Timer counter as 0, When PWM internal counter reaches
to 0, disable PWM Timer*/
        /*---------------------------------------------------------------------------------------*/
        /* Set PWM Timer 1 counter as 0 */
        DrvPWM_SetTimerCounter(u8Timer, 0);
```

```
                /* Wait PWM Timer1 Interrupt */

                while (DrvPWM_GetTimerCounter(u8Timer));

                /* Disable PWM Timer 1 */
                DrvPWM_Enable(u8Timer, 0);

                /* Disable Output function for PWM Timer 1 */
                DrvPWM_SetTimerIO(u8Timer, 0);

                /*---------------------------------------------------------------------------------------*/
                /* Stop PWM Timer 3 (Recommended procedure method 1)        */
                /* Set PWM Timer counter as 0, When PWM internal counter reaches to 0, disable PWM Timer*/
                /*---------------------------------------------------------------------------------------*/

                /* Set PWM Capture 3 counter as 0 */
                DrvPWM_SetTimerCounter(u8CapTimer & 0x0F, 0);

                /* Wait PWM Capture 3 Counter reach to 0 */
                while (DrvPWM_GetTimerCounter(u8CapTimer));

                /* Clear the PWM Capture 3 Interrupt */
                DrvPWM_ClearInt(u8CapTimer);

                /* Disable PWM Capture 3 */
                DrvPWM_Enable(u8CapTimer, 0);

                /* Disable Input function for PWM Capture 3 */
                DrvPWM_SetTimerIO(u8CapTimer, 0);

/*-----------------------------Deleted-----------------------------*/
        printf("PWM sample is complete.\n");
        DrvPWM_Close();
        return 0;
}
```

2.2 PWM Callback Function

The call back function is used to count PWM Timer 0 pulse. When the pulse counter reaches to a specific value, the call back function will set CNR as 0 and clear a flag for timer stop.

```
/*---------------------------------------------------------------------------------------------------------*/
/* PWM Timer Callback function                                                                             */
/*---------------------------------------------------------------------------------------------------------*/
void DRVPWM_PwmIRQHandler()
{
        if (s_u32Pulse == 1 * g_u16Frequency /10)
        {
                /*-------------------------------------------------------------------------------------*/
                /* Stop PWM Timer 0 (Recommended procedure method 2)         */
                /* Set PWM Timer counter as 0, When interrupt request happen,
disable PWM Timer                          */
                /*-------------------------------------------------------------------------------------*/

                DrvPWM_SetTimerCounter(DRVPWM_TIMER0, 0);
        }

        if (s_u32Pulse == 1 * g_u16Frequency /10 + 1)
                g_u8PWMCount = 0;
        s_u32Pulse++;
}
```

2.3 Capture Function

The capture function is to calculate the input waveform information.

Note: When the falling transition or rising transition interrupt is enabled, the bit, CAPIFx, x=0~7, is set by hardware, software can clear the bit by write a one to it. If the bit is not cleared, PWM counter will not reload when next capture interrupt occurs.

```
void CalPeriodTime(uint8_t u8Capture)
{
        uint16_t u32Count[4];
```

```
        uint32_t u32i;
        uint16_t u16RisingTime, u16FallingTime, u16HighPeroid, u16LowPeroid,
u16TotalPeroid;


        /* Clear the Capture Interrupt Flag (Time A) */
        DrvPWM_ClearCaptureIntStatus(u8Capture,
DRVPWM_CAP_FALLING_FLAG);


        /* Wait for Interrupt Flag  (Falling) */
        while (DrvPWM_GetCaptureIntStatus(u8Capture,
DRVPWM_CAP_FALLING_FLAG) != 1);


        /* Clear the Capture Interrupt Flag (Time B)*/
        DrvPWM_ClearCaptureIntStatus(u8Capture,
DRVPWM_CAP_FALLING_FLAG);


        u32i = 0;


        while (u32i < 4)
        {
                /* Wait for Interrupt Flag (Falling) */
                while(DrvPWM_GetCaptureIntStatus(u8Capture,
DRVPWM_CAP_FALLING_FLAG) != 1);


                /* Clear the Capture Interrupt Flag */
                DrvPWM_ClearCaptureIntStatus(u8Capture,
DRVPWM_CAP_FALLING_FLAG);


                /* Clear the Capture Rising Interrupt Flag */
                DrvPWM_ClearCaptureIntStatus(u8Capture,
DRVPWM_CAP_RISING_FLAG);


                /* Get the Falling Counter Data */
                u32Count[u32i++] = DrvPWM_GetFallingCounter(u8Capture);


                /* Wait for Capture Rising Interrupt Flag */
                while(DrvPWM_GetCaptureIntStatus(u8Capture,
DRVPWM_CAP_RISING_FLAG) != 1);
```

```
                /* Clear the Capture Rising Interrupt Flag */
                DrvPWM_ClearCaptureIntStatus(u8Capture,
DRVPWM_CAP_RISING_FLAG);

                /* Get the Rising Counter Data */
                u32Count[u32i++] = DrvPWM_GetRisingCounter(u8Capture);
        }

        u16RisingTime = u32Count[1];

        u16FallingTime = u32Count[0];

        u16HighPeroid = u32Count[1] - u32Count[2];

        u16LowPeroid = 0x10000 - u32Count[1];

        u16TotalPeroid = 0x10000 - u32Count[2];

        printf("Test Result:\nRising Time = %d, Falling Time = %d.\nHigh Period =
%d, Low  Period = %d, Total Period = %d.\n\n",
                u16RisingTime, u16FallingTime, u16HighPeroid, u16LowPeroid,
u16TotalPeroid);
}
```

2.4 API Usage Reference

- PWM Driver Reference Guide.doc

3 EXECUTION ENVIRONMENT SETUP AND RESULT

3.1 Test Smpl_DrvPWM

The PWM sample code, Smpl_DrvPWM, could be built by Keil MDK tool and download to NUC1xx series DEV Board through ICE. Then user is able to execute the code in ICE environment or reset the DEV board to execute the code which had been programmed in on-chip Program Flash.

Then, select test items, PWM waveform test and PWM capture test.

3.2 Result of PWM Waveform Test

Hear different note frequency (Do, Re, Mi, Fa, Sol, La, and Si) from buzzer. Or measure GPIO A12 and observe the property of waveform by an oscilloscope. The frequency and duty cycle should be the same with the selected tone

3.3 Result of PWM Capture Test

PWM3 is used to capture the waveform property of PWM1 output. First, GPIO A13 (PWM1 function pin) is connected with GPIO A15 (PWM3 function pin). The PWM3 will get counter data when rising/falling transition occurs. According to counter data, calculate the waveform property, waveform period, duty cycle and etc.

4 REVISION HISTORY

REV.	DATE	DESCRIPTION
1.00	March, 2010	Created.

Important Notice

Nuvoton products are not designed, intended, authorized or warranted for use as components in systems or equipment intended for surgical implantation, atomic energy control instruments, airplane or spaceship instruments, transportation instruments, traffic signal instruments, combustion control instruments, or for other applications intended to support or sustain life. Further more, Nuvoton products are not intended for applications wherein failure of Nuvoton products could result or lead to a situation wherein personal injury, death or severe property or environmental damage could occur.

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

Please note that all data and specifications are subject to change without notice. All the trademarks of products and companies mentioned in this datasheet belong to their respective owners.