应用指南

32位 Cortex™-M0 单片机
NuMicro® 系列

NUC1xx中如何使用PWM?

目录

1 简介 2
1.1 PWM功能特性 2
1.2 捕捉功能特性 2
1.3 结构 2
1.4 PWM操作 3
1.4.1 PWM定时器操作 3
1.4.2 PWM双缓存, 自动重载以及单次触发操作 4
1.4.3 调制占空比 5
1.4.4 死区发生器 5
1.4.5 捕捉操作 5
1.4.6 PWM定时器启动步骤 6
1.4.7 PWM停止流程 7
1.4.8 捕捉启动流程 7
2 代码部分 8
2.1 主函数(在Smpl_DrvPWM.c中) 8
2.2 PWM回调函数 13
2.3 捕捉功能 13
2.4 API使用参考 15
3 运行环境设置及结果 16
3.1 测试Smpl_DrvPWM 16
3.2 PWM波形测试结果 16
3.3 PWM捕捉测试结果 16
4 修订历史 17

1 简介

本文档讲述了如何利用PWM定时器产生一个特定频率的波形并输出到蜂鸣器，以及利用PWM捕捉功能捕捉输入波形的特性。根据IC型号不同PWM模块最多可有4套PWM产生器，可以配置为8个独立的PWM输出，PWM0~7，或4组互补的带可编程死区发生器的PWM输出(PWM0, PWM1), (PWM2, PWM3), (PWM4, PWM5), (PWM6, PWM7) 。PWM定时器的另一个功能是输入捕捉功能，捕捉功能使能后，当输入通道电平有上升沿时PWM计数器将被锁存到上升捕捉锁存器中，当输入通道电平有下降沿时PWM计数器将被锁存到下降捕捉锁存器中。捕捉通道0和PWM通道0共用一个PWM计时器，捕捉通道1和PWM通道1共用一个PWM计时器，以此类推，所以PWM功能和捕捉功能不能在同一通道中同时使用。

1.1 PWM功能特性

- 4个PWM发生器，每个发生器有1个8位预分频器，1个时钟除频器，PWM定时器，1个死区发生器以及两路输出.
- 最多8个PWM通道或4对PWM通道
- 16位分辨率
- 支持单次触发模式和自动重载模式
- 双缓存可以避免PWM输出无效波形

1.2 捕捉功能特性

- 捕捉功能与PWM共用定时器控制逻辑，如果捕捉功能使能，PWM输出转换为捕捉功能输入
- 8个捕捉输入通道
- 每个通道拥有1个上升锁存器，1个下降锁存器以及捕捉中断标志

1.3 结构

下图描述了一组PWM的结构，PWM0和1是一组，PWM2和3是一组以此类推

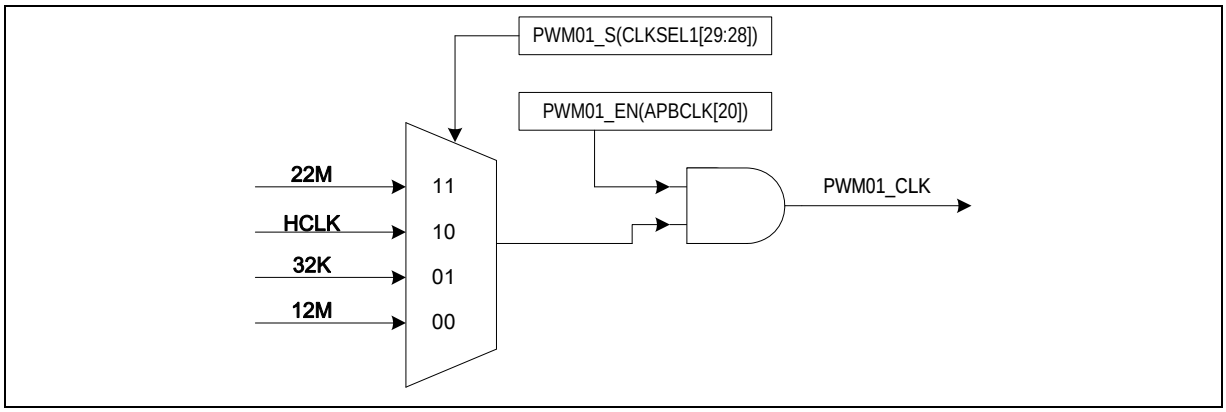

图 1 PWM 0&1 时钟源控制

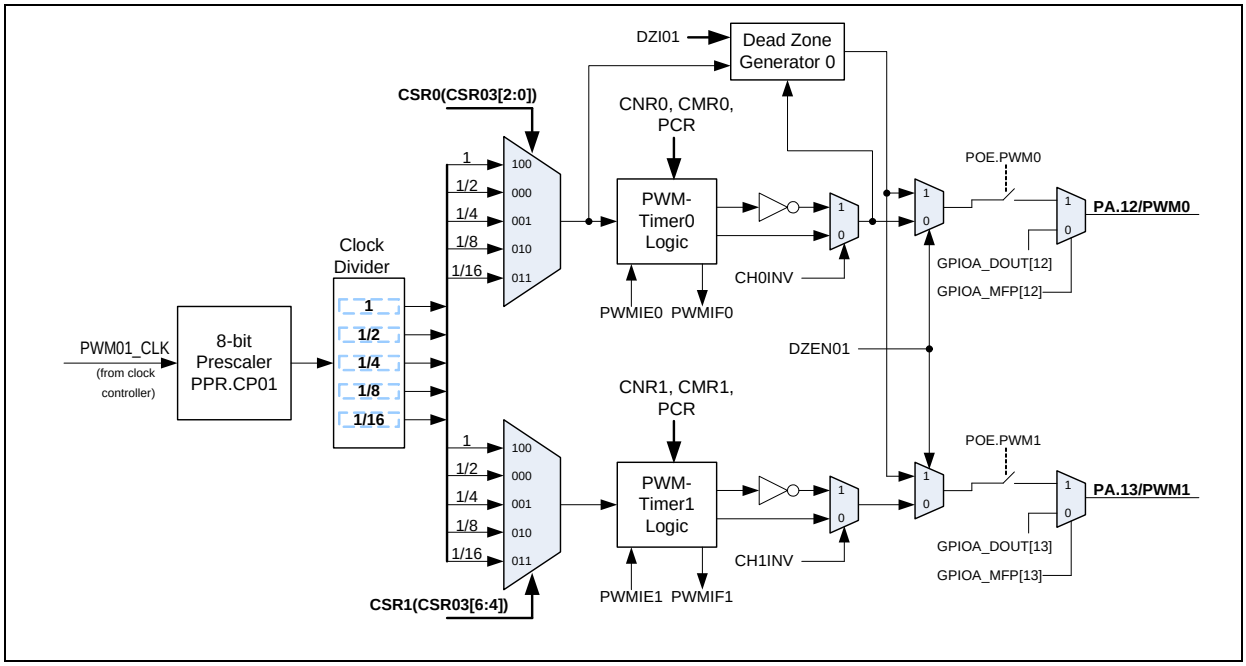

图 2　PWM 0&1 结构图

1.4　PWM操作

1.4.1　PWM定时器操作

PWM向下计数器(CNR)和比较寄存器(CMR)决定了PWM周期和占空比。图4描述了PWM定时器时序操作。脉冲宽度调节遵循下面的公式，PWM比较器的原理也在图3中说明了，请注意相应的GPIO口必须设置为PWM输出

- PWM频率= PWMxy_CLK/(prescale+1)*(clock divider)/(CNR+1); 其中xy可以是01, 23, 45或67, 这取决于所选的PWM通道
- 占空比 = (CMR+1) / (CNR+1).
- CMR >= CNR: PWM 一直输出高
- CMR < CNR: PWM低脉冲宽度= (CNR-CMR) unit[1]; PWM 高脉冲宽度= (CMR+1) unit.
- 如果CMR = 0: PWM低脉冲宽度 = (CNR) unit; PWM 高脉冲宽度= 1 unit

注意: 1. Unit = 1个PWM时钟周期

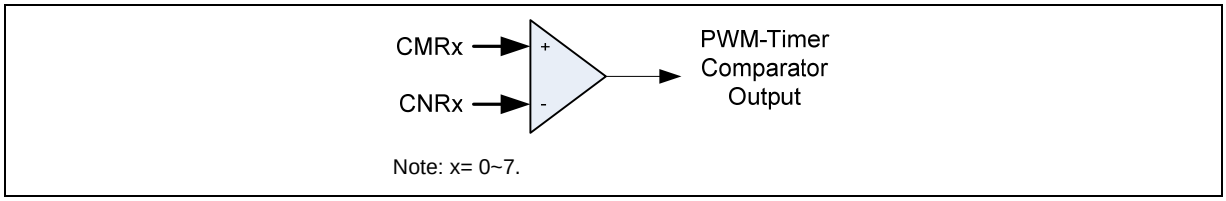

图 3　PWM 定时器内部比较器输出

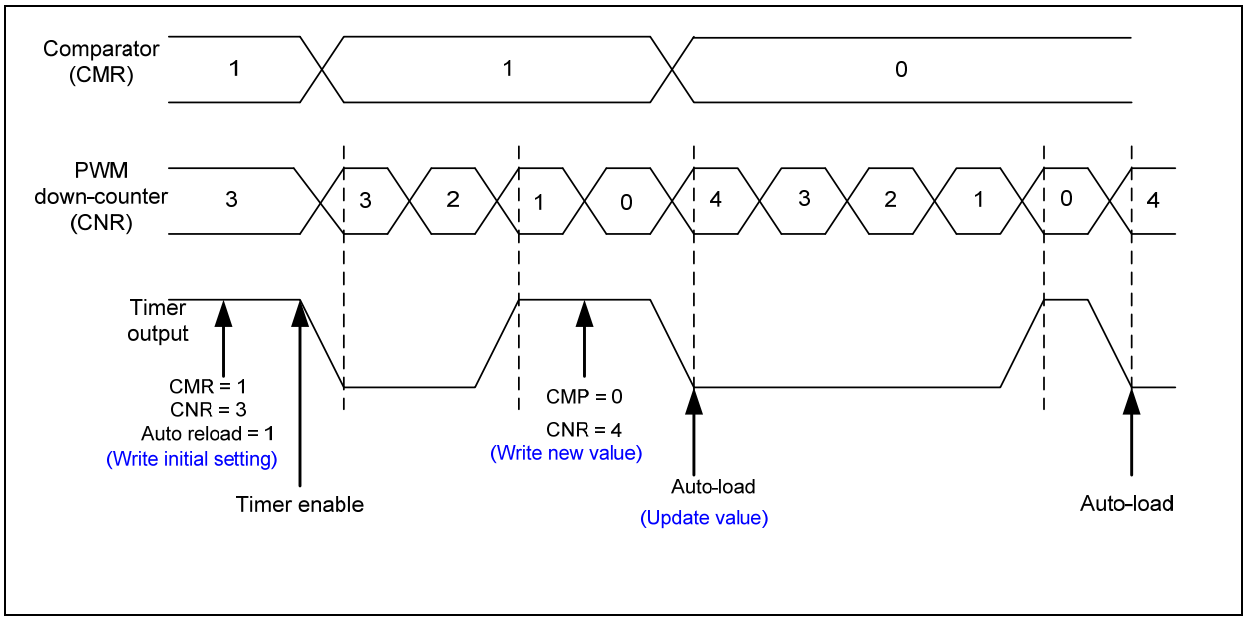

图 4　PWM 定时器操作时序

1.4.2　PWM双缓存, 自动重载以及单次触发操作

NUC1XX 系列PWM定时器具有双缓存功能。寄存器预先设定的值，在一个周期完成后，可以自动重载而不影响当前定时器操作。PWM计数器值写入CNR0~7，并可从PDR0~7内读出当前计数器的值。

PWM 控制寄存器(PCR) 的CH0MOD 位定义PWM0是自动重载模式或是单触发模式。自动重载模式下，当PWM计数器计到0，MCU自动重载CNR0 值到PWM 计数器。如果CNR0 设定为0，PWM计数器计数到0后，将进入暂停状态。如果此时CH0MOD设定为0 ，计数器停止运行。PWM1~PWM7 运行状态与PWM0 相同.

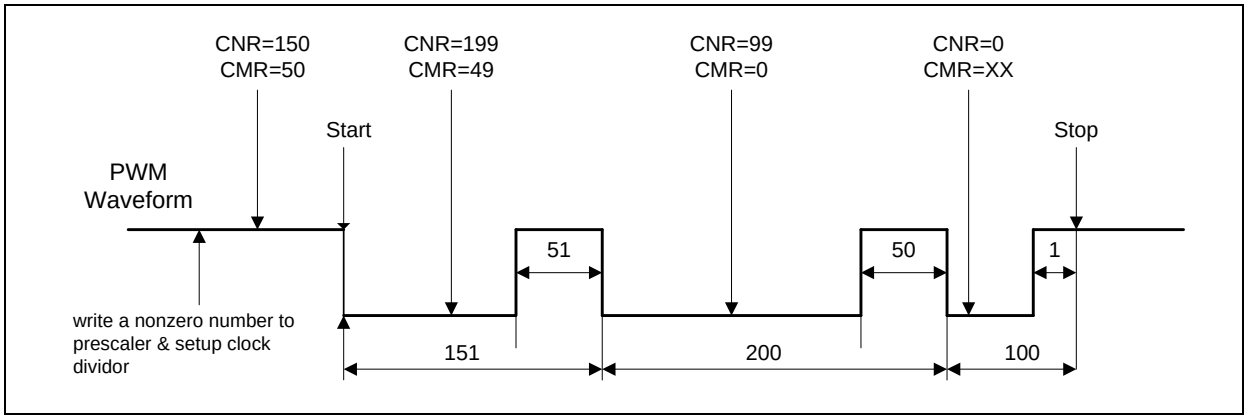

图 5　PWM双缓存示意图

1.4.3 调制占空比

双缓存允许CMR在当前周期任意时刻改写，下一个周期写入值生效。

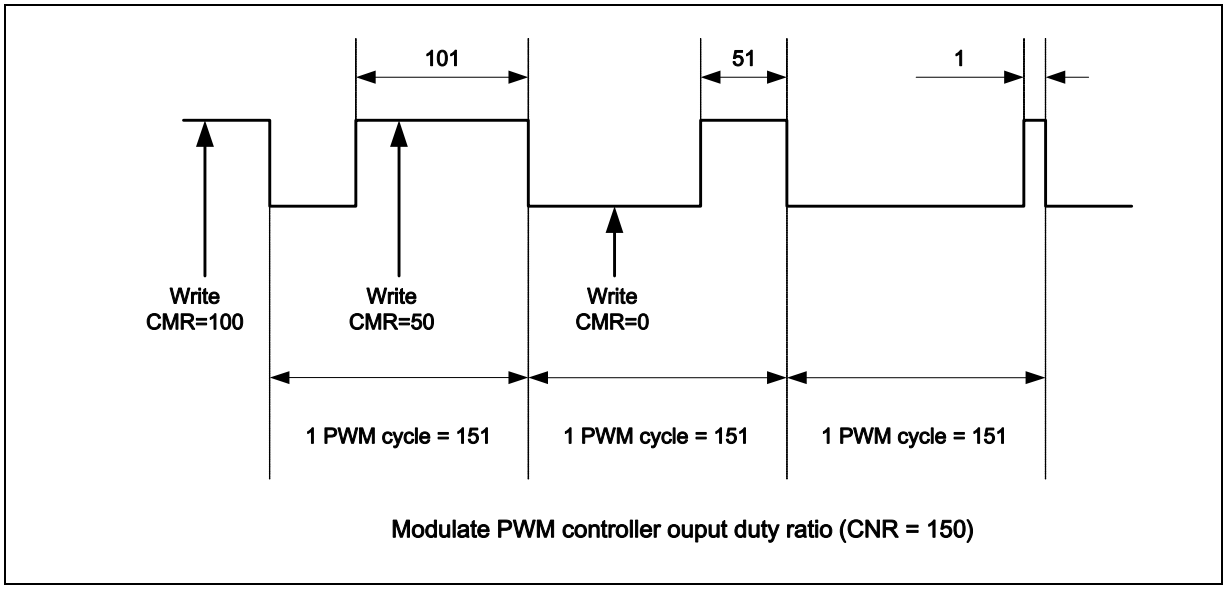

图 6 PWM 控制器输出占空比

1.4.4 死区发生器

NUC1XX 系列提供PWM死区发生器，用于保护电源器件。用户可通过设定PPRx.DZI定义在PWM的上升沿插入延迟时间

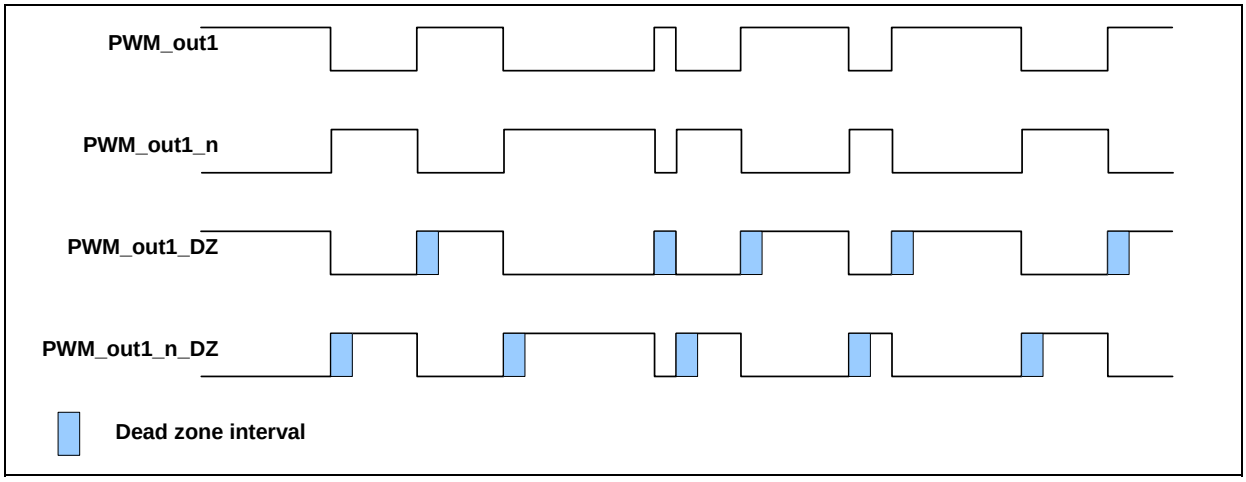

图 7 死区发生器操作图

1.4.5 捕捉操作

捕捉器0和PWM0使用同一个定时器，捕捉器1和PWM1使用另一组定时器，以此类推。当输入通道电平有上升沿时PWM计数器将被锁存到**CRLR** 中，当输入通道电平有下降沿时PWM计数器将被锁存到**CFLR** 中。设定CCR0[1] (上升沿触发中断有效)和CCR0[2](下降沿触发中断有效),可以设定捕捉器通道0

产生中断的条件。同样设定CCR0[17] 和CCR0[18]，可以设定通道1，以此类推。一旦捕捉功能发出中断0/1/2...，PWN计数器0/1/2...将马上重载。注意在捕捉功能使能前，相应的管脚必须配置为输入模式(GPIOA_OMD)。

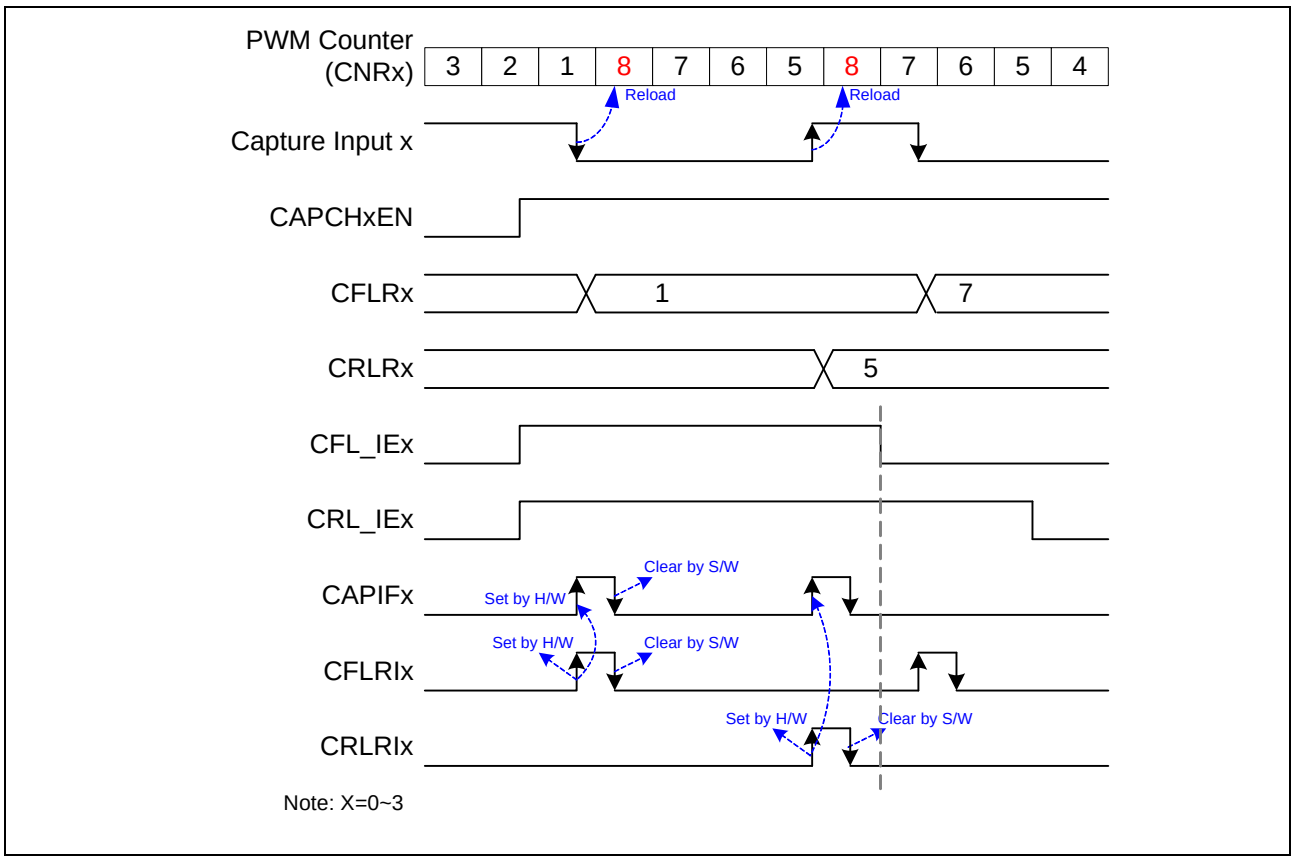

图 8 捕捉操作时序

该例中，CNR为8:

1. 当上升沿与下降沿中断同时使能时，PWM计数器将在中断发生时重载
2. 通道低脉宽为(CNR – CRLR).
3. 通道高脉宽为(CNR– CFLR).

1.4.6 PWM定时器启动步骤

建议如下步骤启动PWM定时器

1. 设置时钟选择(CSR)
2. 设置预分频(PPR)
3. 设置反向开/关，死区发生器开/关，自动重载/单次触发模式以及PWM定时器关闭（PCR）
4. 根据PWM占空比设置比较寄存器(CMR)

5. 根据PWM周期设置PWM计数器(CNR)
6. 设置中断使能寄存器(PIER)
7. 设置PWM输出使能(POE)
8. 设置相应的GPIO口为PWM功能
9. 使能PWM定时器(PCR)

1.4.7 PWM停止流程

方式1:

设定16位向下计数计数器(CNR)为0 ， 并查看PDR状态，当PDR达到0 ， 关闭PWM定时器 (PCR的CHxEN位). ***(***推荐***)***

方式2:

设定16位向下计数计数器(CNR)为0，当中断发生，在中断内关闭PWM定时器(PCR的CHxEN位). ***(***推荐***)***

方式3:

直接关闭PWM定时器(PCR的CHxEN位). (不推荐)

1.4.8 捕捉启动流程

1. 设置时钟选择（CSR）
2. 设置预分频(PPR)
3. 设置通道使能，上升/下降沿中断使能及输入信号反向开/关(CCR0, CCR1)
4. 设置PWM计数器(CNR)
5. 设置捕捉输入使能寄存器(CAPENR)
6. 设置相应GPIO口为PWM功能
7. 使能PWM定时器(PCR)

2 代码部分

2.1 主函数(在Smpl_DrvPWM.c中)

下面的代码代表PWM定时器的两个主要功能：PWM功能和捕捉功能。首先是7个特定的频率波形从PWM0输入蜂鸣器，接下来PWM3作为捕捉操作，捕捉PWM1输出的波形特性，并计算PWM1输出的低脉冲宽度，高脉冲宽度及周期

```
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回调函数

回调函数用来计算PWM脉冲，当脉冲计数器达到一个特定值时，回调函数将设置CNR为0并清除停止定时器的标志。

```
/*---------------------------------------------------------------------------------------------------------*/
/* 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 捕捉功能

捕捉功能用来计算输入波形的信息

注意：当下降或上升沿中断使能时，CAPIFx, x=0~7 位被硬件置位，软件可以通过对该位写1清零，如果此位没有清零，PWM计数器将不会在下一个捕捉中断发生时重载。

```
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使用参考

- PWM Driver Reference Guide.doc

3 运行环境设置及结果

3.1 测试Smpl_DrvPWM

PWM示例程序，Smpl_DrvPWM,可以通过Keil MDK工具编译并通过ICE下载到NUC1xx系列DEV板，用户可以在ICE环境下执行代码或复位DEV板执行已经下载到片内Flash的程序，然后根据提示选择测试项，PWM波形测试和PWM捕捉测试

3.2 PWM波形测试结果

蜂鸣器中可以听到不同音符的频率(Do, Re, Mi, Fa, Sol, La, 和Si)，或者用示波器测试GPIO A12的波形，频率和占空比将与选择的音调一致

3.3 PWM捕捉测试结果

PWM3用来测试PWM1输出波形特性，首先将GPIO A13 (PWM1功能脚)与GPIO A15 (PWM3功能脚)连接，PWM3将在上升/下降沿时获得计数器的值，根据这些值计算出波形特性：周期，占空比等等

4 修订历史

版本	日期	描述
0.01	2010，3	初次发布

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.