

PWM模块

程晨闻
电气工程学院



➤ 脉宽调制

– 应用

- 幅值 --> 脉宽
- 电力电子开关的控制

– TM4C1294的PWM模块

- 4个PWM发生器
- 16位计数器
- 每个发生器可产生2路PWM输出
- 比较事件
- 死区



➤ 内容概要

– TM4C1294 PWM模块的使用方法



➤ TM4C1294 PWM模块的使用方法

- 使用PWM_GEN_0产生**单相中心对称互补导通**的PWM信号 (PWM0和PWM1)
 1. **开启**PWM模块的**时钟**
 2. **开启**PWM的输出引脚所在的**GPIO模块的时钟**，并把PWM模块的输出信号**与GPIO引脚相连**
 3. 配置PWM计数器的**计数模式**和PWM信号**产生方式**
 4. 设置PWM载波的**周期**和PWM的**占空比**
 5. 使能并设置**死区**
 6. **使能**PWM信号的**输出**
 7. 使能并配置**中断**
 8. **使能PWM发生器**
 9. 编写**中断服务函数**，根据功能要求更新占空比



➤ TM4C1294 PWM模块的使用方法

- 使用PWM_GEN_0产生单相中心对称互补导通的PWM信号 (PWM0和PWM1)

1. 开启PWM模块的时钟

TM4C1294微控制器有一个PWM模块（包含4个PWM发生器）
这个PWM模块记做PWM0

```
SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM0);
```



➤ TM4C1294 PWM模块的使用方法

- 使用PWM_GEN_0产生单相中心对称互补导通的PWM信号 (PWM0和PWM1)

2. 开启PWM的输出引脚所在的**GPIO模块的时钟**，并把PWM模块的输出信号**与GPIO引脚相连**

四个PWM发生器的输出信号**M0PWMY**(Y=0,1,2,3,4,5,6,7)与GPIO模块共享MCU的实际引脚

Pin Name	Pin Number	Pin Mux / Pin Assignment	Pin Type	Buffer Type	Description
M0PWM0	42	PF0 (6)	O	TTL	Motion Control Module 0 PWM 0. This signal is controlled by Module 0 PWM Generator 0.
M0PWM1	43	PF1 (6)	O	TTL	Motion Control Module 0 PWM 1. This signal is controlled by Module 0 PWM Generator 0.
M0PWM2	44	PF2 (6)	O	TTL	Motion Control Module 0 PWM 2. This signal is controlled by Module 0 PWM Generator 1.
M0PWM3	45	PF3 (6)	O	TTL	Motion Control Module 0 PWM 3. This signal is controlled by Module 0 PWM Generator 1.
M0PWM4	49	PG0 (6)	O	TTL	Motion Control Module 0 PWM 4. This signal is controlled by Module 0 PWM Generator 2.
M0PWM5	50	PG1 (6)	O	TTL	Motion Control Module 0 PWM 5. This signal is controlled by Module 0 PWM Generator 2.
M0PWM6	63	PK4 (6)	O	TTL	Motion Control Module 0 PWM 6. This signal is controlled by Module 0 PWM Generator 3.
M0PWM7	62	PK5 (6)	O	TTL	Motion Control Module 0 PWM 7. This signal is controlled by Module 0 PWM Generator 3.

➤ TM4C1294 PWM模块的使用方法

- 使用PWM_GEN_0产生单相中心对称互补导通的PWM信号 (PWM0和PWM1)

2. 开启PWM的输出引脚所在的**GPIO模块的时钟**，并把PWM模块的输出信号**与GPIO引脚相连**

```
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);  
GPIOPinTypePWM(GPIO_PORTF_BASE, GPIO_PIN_0 | GPIO_PIN_1);  
GPIOPinConfigure(GPIO_PF0_M0PWM0); //configure the PF0 pin to M0PWM0  
GPIOPinConfigure(GPIO_PF1_M0PWM1); //configure the PF1 pin to M0PWM1
```



➤ TM4C1294 PWM模块的使用方法

- 使用PWM_GEN_0产生单相中心对称互补导通的PWM信号 (PWM0和PWM1)

3. 配置PWM计数器的**计数模式**和PWM信号**产生方式**

中心对称，需要产生**三角载波**

PWMn Control (**PWMnCTL**)寄存器，用于PWM发生器模块的配置

PWMn Control (PWMnCTL)

PWM0 base: 0x4002.8000

Offset 0x040

Type RW, reset 0x0000.0000

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	reserved													LATCH	MINFLTPER	FLTSRC
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RW	RW	RW
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	DBFALLUPD		DBRISEUPD		DBCTLUPD		GENBUPD		GENAUPD		CMPBUPD	CMPAUPD	LOADUPD	DEBUG	MODE	ENABLE
Type	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

MODE: 0: 向下计数，锯齿载波

1: 上下计数，三角载波



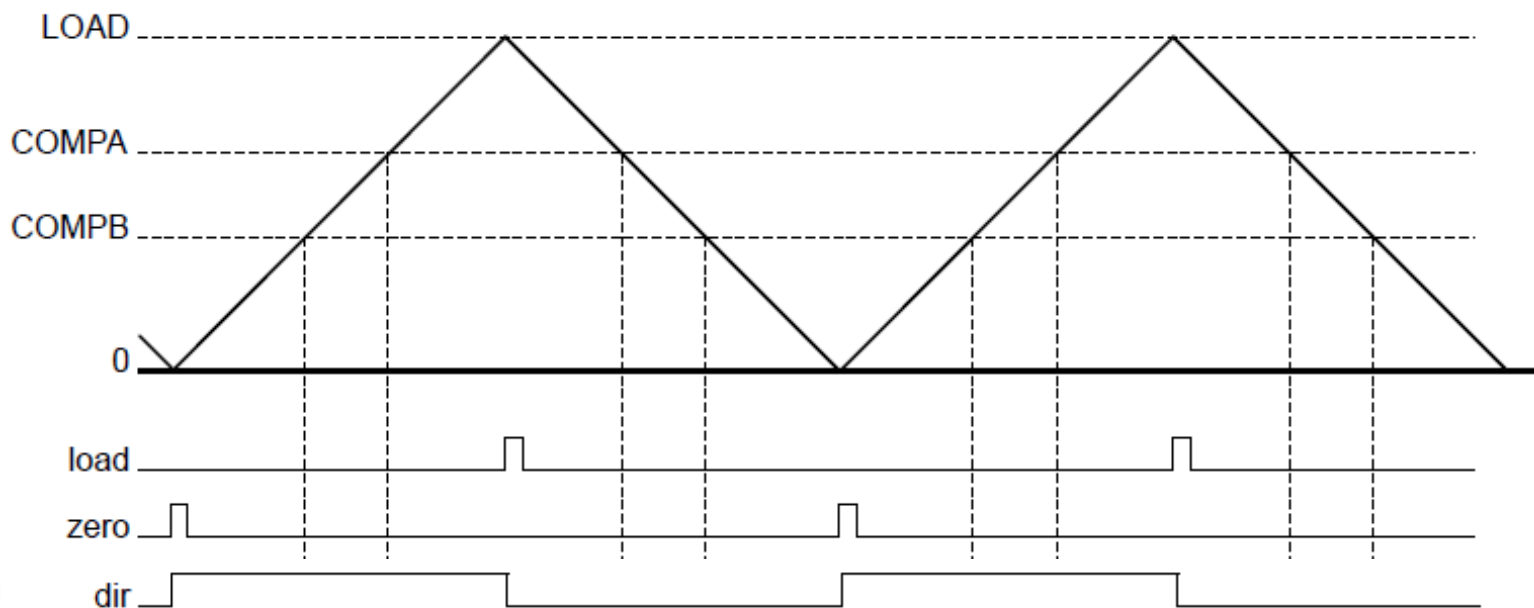
➤ TM4C1294 PWM模块的使用方法

- 使用PWM_GEN_0产生单相中心对称互补导通的PWM信号 (PWM0和PWM1)

3. 配置PWM计数器的**计数模式**和PWM信号**产生方式**

中心对称，需要产生**三角载波**

PWMn Control (**PWMnCTL**)寄存器，用于PWM发生器模块的配置



MODE=1: 上下计数，三角载波

9/45



➤ TM4C1294 PWM模块的使用方法

- 使用PWM_GEN_0产生单相中心对称互补导通的PWM信号 (PWM0和PWM1)

3. 配置PWM计数器的计数模式和PWM信号产生方式

PWMn Generator A Control (PWMnGENA) 寄存器，根据load、zero、AUp、ADown、BUUp、BDown这几个信号，改变pwmA和pwmB的电平

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	reserved															
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	reserved				ACTCMPBD		ACTCMPBU		ACTCMPAD		ACTCMPAU		ACTLOAD		ACTZERO	
Type	RO	RO	RO	RO	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

ACTZERO: zero信号

ACTCMPBU: BUUp信号

ACTLOAD: load信号

ACTCMPBD: BDown信号

ACTCMPAU: AUUp信号

ACTCMPAD: ADown

pwmA的电平

- 0: 不变
1: 翻转 (取反)
2: 输出低电平
3: 输出高电平

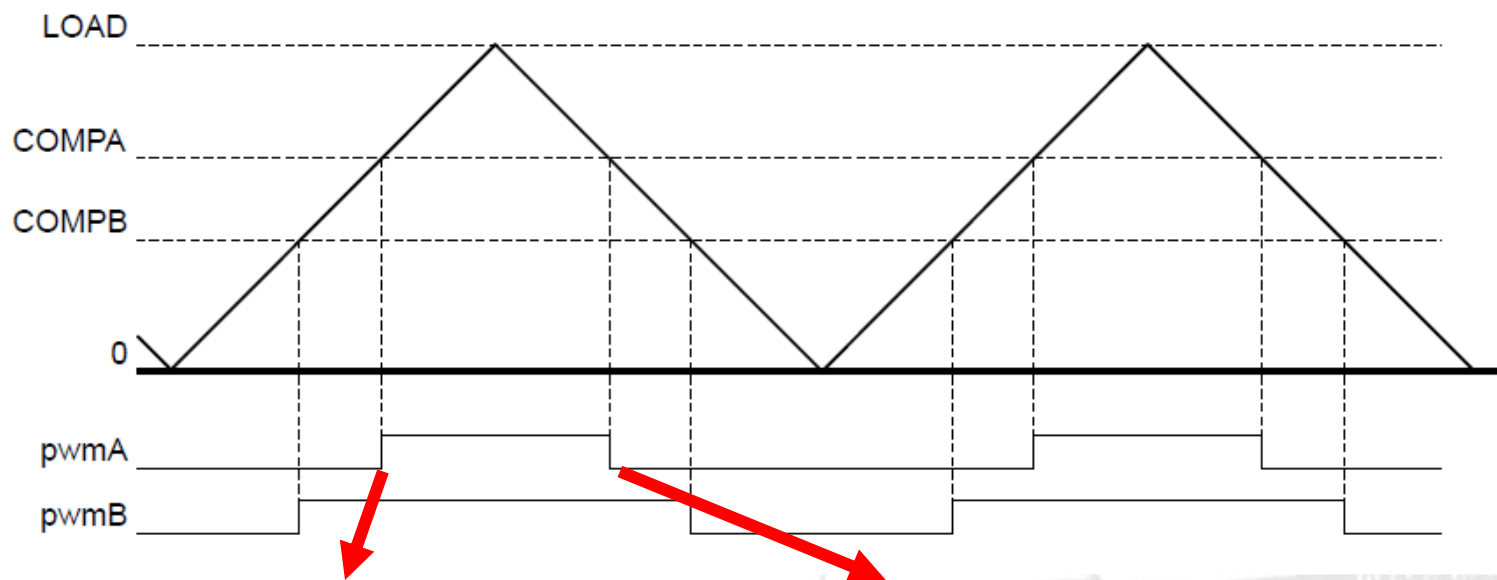


➤ TM4C1294 PWM模块的使用方法

- 使用PWM_GEN_0产生单相中心对称互补导通的PWM信号 (PWM0和PWM1)

3. 配置PWM计数器的计数模式和PWM信号产生方式

中心对称的PWM信号



在Aup时刻，把pwmA信号
设置为高电平，即
 $ACTCMPAU=0x03$ 。

在Adown时刻，把pwmA
信号设置为低电平，即
 $ACTCMPAD=0x02$ 。



➤ TM4C1294 PWM模块的使用方法

- 使用PWM_GEN_0产生单相中心对称互补导通的PWM信号 (PWM0和PWM1)

3. 配置PWM计数器的**计数模式**和PWM信号**产生方式**

TivaWare提供了**PWMGenConfigure**函数，配置计数模式和PWM信号产生方式



```
void PWMGenConfigure(uint32_t ui32Base, uint32_t ui32Gen, uint32_t ui32Config)
{
```

```
    // ...
```

```
    ui32Gen = PWM_GEN_BADDR(ui32Base, ui32Gen);
```

```
    // Change the global configuration of the generator.
```

```
    HWREG(ui32Gen + PWM_O_X_CTL) = ((HWREG(ui32Gen + PWM_O_X_CTL) &
                                     ~(PWM_X_CTL_MODE | PWM_X_CTL_DEBUG |
                                       PWM_X_CTL_LATCH | PWM_X_CTL_MINFLTPER |
                                       PWM_X_CTL_FLTSRC |
                                       PWM_X_CTL_DBFALLUPD_M |
                                       PWM_X_CTL_DBRISEUPD_M |
                                       PWM_X_CTL_DBCTLUPD_M |
                                       PWM_X_CTL_GENBUPD_M |
                                       PWM_X_CTL_GENAUPD_M |
                                       PWM_X_CTL_LOADUPD | PWM_X_CTL_CMPAUPD |
                                       PWM_X_CTL_CMPBUPD)) | ui32Config);
```

```
    // ...
```

```
    if(ui32Config & PWM_X_CTL_MODE)
    {
```

```
        //
```

```
        // In up/down count mode, set the signal high on up count comparison
        // and low on down count comparison (that is, center align the
        // signals).
```

```
        //
```

```
        HWREG(ui32Gen + PWM_O_X_GENA) = (PWM_X_GENA_ACTCMPAU_ONE |
                                           PWM_X_GENA_ACTCMPAD_ZERO);
```

```
        HWREG(ui32Gen + PWM_O_X_GENB) = (PWM_X_GENB_ACTCMPBU_ONE |
                                           PWM_X_GENB_ACTCMPBD_ZERO);
```

```
    }
```

```
    else
```

```
    // ...
```

```
}
```

```
#define PWM_X_CTL_MODE          0x00000002 // Counter Mode
#define PWM_X_GENA_ACTCMPAU_ONE 0x00000030 // Drive pwmA High
#define PWM_X_GENA_ACTCMPAD_ZERO 0x00000080 // Drive pwmA Low
```



➤ TM4C1294 PWM模块的使用方法

- 使用PWM_GEN_0产生单相中心对称互补导通的PWM信号 (PWM0和PWM1)

3. 配置PWM计数器的计数模式和PWM信号产生方式

TivaWare提供了PWMGenConfigure函数，计数模式和PWM信号产生方式

PWMGenConfigure函数通过参数ui32Config设置指定PWM发生器（0，1，2，3）的PWMnCTL寄存器改变计数模式

如果是上下计数（三角载波），就在PWMnGENA寄存器中，在Aup时刻，把pwmA信号设置为高电平，在Adown时刻，把pwmA信号设置为低电平。以此产生中心对称的PWM信号。

配置PWM发生器PWM_GEN_0的计数方式为上下计数，生成三角载波，产生中心对称的PWM信号：

PWMGenConfigure(PWM0_BASE, PWM_GEN_0, PWM_GEN_MODE_UP_DOWN);



➤ TM4C1294 PWM模块的使用方法

- 使用PWM_GEN_0产生单相中心对称互补导通的PWM信号 (PWM0和PWM1)

4. 设置PWM载波的周期和PWM的占空比

PWM的载波周期由PWMn Load (PWMn**LOAD**)寄存器决定

- 如果使用锯齿载波，载波周期为 $(\text{PWMnLOAD} + 1) / \text{SYSTEMCLOCK}$
- 如果使用三角载波，载波周期为 $(2 * \text{PWMnLOAD}) / \text{SYSTEMCLOCK}$

TivaWare提供了**PWMGenPeriodSet**函数，帮助设置载波周期



```

void
PWMGenPeriodSet(uint32_t ui32Base, uint32_t ui32Gen,
                uint32_t ui32Period)
{
    //...
    //
    // Compute the generator's base address.
    ui32Gen = PWM_GEN_BADDR(ui32Base, ui32Gen);
    //
    // Set the reload register based on the mode.
    if(HWREG(ui32Gen + PWM_O_X_CTL) & PWM_X_CTL_MODE)
    {
        //
        // In up/down count mode, set the reload register to half the requested
        // period.
        ASSERT((ui32Period / 2) < 65536);
        HWREG(ui32Gen + PWM_O_X_LOAD) = ui32Period / 2;
    }
    else
    {
        //
        // In down count mode, set the reload register to the requested period
        // minus one.
        ASSERT((ui32Period <= 65536) && (ui32Period != 0));
        HWREG(ui32Gen + PWM_O_X_LOAD) = ui32Period - 1;
    }
}

```



➤ TM4C1294 PWM模块的使用方法

- 使用PWM_GEN_0产生单相中心对称互补导通的PWM信号 (PWM0和PWM1)

4. 设置PWM载波的周期和PWM的占空比

PWM的载波周期由PWMn Load (PWMn**LOAD**)寄存器决定

TivaWare提供了**PWMGenPeriodSet**函数，帮助设置载波周期

PWMGenPeriodSet函数首先检测计数模式（载波模式）

如果计数模式为向下计数（锯齿载波），该函数把参数**ui32Period**的值减1后，赋给PWMn**LOAD**寄存器

如果计数模式为上下计数（三角载波），那么该函数把参数**ui32Period**除2后，赋给PWMn**LOAD**寄存器

也就是说，不管载波是何种波形，**PWMGenPeriodSet**函数的**ui32Period**参数都是一整个载波周期的计数时长

PWMGenPeriodSet函数的第二个参数为PWM发生器的代号，可以是

- PWM_GEN_0
- PWM_GEN_1
- PWM_GEN_2
- PWM_GEN_3



➤ TM4C1294 PWM模块的使用方法

- 使用PWM_GEN_0产生单相中心对称互补导通的PWM信号 (PWM0和PWM1)

4. 设置PWM载波的周期和PWM的占空比

PWM的载波周期由PWMn Load (PWMn**LOAD**)寄存器决定

TivaWare提供了**PWMGenPeriodSet**函数，帮助设置载波周期

```
ui32Load = (ui32PWMClock / PWM_FREQUENCY);  
PWMGenPeriodSet(PWM0_BASE, PWM_GEN_0, ui32Load);
```



➤ TM4C1294 PWM模块的使用方法

- 使用PWM_GEN_0产生单相中心对称互补导通的PWM信号 (PWM0和PWM1)

4. 设置PWM载波的周期和PWM的占空比

PWM的占空比由COMP_A和COMP_B决定

COMP_A对应PWM_n Compare A (PWM_nCOMP_A)寄存器

COMP_B对应PWM_n Compare B (PWM_nCOMP_B)寄存器

TivaWare提供了PWMPulseWidthSet函数，设置占空比



```

void
PWMPulseWidthSet(uint32_t ui32Base, uint32_t ui32PWMOut,
                  uint32_t ui32Width)
{
    uint32_t ui32GenBase, ui32Reg;
    // Compute the generator's base address.
    ui32GenBase = PWM_OUT_BADDR(ui32Base, ui32PWMOut);
    // If the counter is in up/down count mode, divide the width by two.
    if(HWREG(ui32GenBase + PWM_O_X_CTL) & PWM_X_CTL_MODE)
    {
        ui32Width /= 2;
    }
    // Get the period.
    ui32Reg = HWREG(ui32GenBase + PWM_O_X_LOAD);
    // Make sure the width is not too large.
    ASSERT(ui32Width < ui32Reg);
    // Compute the compare value.
    ui32Reg = ui32Reg - ui32Width;
    // Write to the appropriate registers.
    if(PWM_IS_OUTPUT_ODD(ui32PWMOut))
    {
        HWREG(ui32GenBase + PWM_O_X_CMPB) = ui32Reg;
    }
    else
    {
        HWREG(ui32GenBase + PWM_O_X_CMPA) = ui32Reg;
    }
}

```

➤ TM4C1294 PWM模块的使用方法

- 使用PWM_GEN_0产生单相中心对称互补导通的PWM信号 (PWM0和PWM1)

4. 设置PWM载波的周期和PWM的占空比

PWM的占空比由COMP_A和COMP_B决定

COMP_A对应PWM_n Compare A (PWM_nCOMP_A)寄存器

COMP_B对应PWM_n Compare B (PWM_nCOMP_B)寄存器

TivaWare提供了PWMPulseWidthSet函数，设置占空比

PWMPulseWidthSet函数会自动判断载波的模式，如果是三角载波，则自动把ui32Width参数的值除2

PWMPulseWidthSet函数在赋值给PWM_nCOMP_A和 PWM_nCOMP_B寄存器之前，会读出PWM_nLOAD寄存器的值，然后用PWM_nLOAD寄存器的值减去ui32Width后，再赋值给PWM_nCOMP_A和 PWM_nCOMP_B寄存器



占空比就变为 $\text{ui32Width} / \text{LOAD}$



➤ TM4C1294 PWM模块的使用方法

- 使用PWM_GEN_0产生单相中心对称互补导通的PWM信号 (PWM0和PWM1)

4. 设置PWM载波的周期和PWM的占空比

占空比就变为 $ui32Width / LOAD$

PWM0引脚的占空比为50%:

```
PWMPulseWidthSet(PWM0_BASE, PWM_OUT_0, ui32Load/2);
```

PWM0引脚的占空比为25%:

```
PWMPulseWidthSet(PWM0_BASE, PWM_OUT_0, ui32Load/4);
```



➤ TM4C1294 PWM模块的使用方法

- 使用PWM_GEN_0产生单相中心对称互补导通的PWM信号 (PWM0和PWM1)

5. 使能并设置死区

PWMn Dead-Band Control (**PWMnDBCTL**)寄存器决定是否启用死区发生器

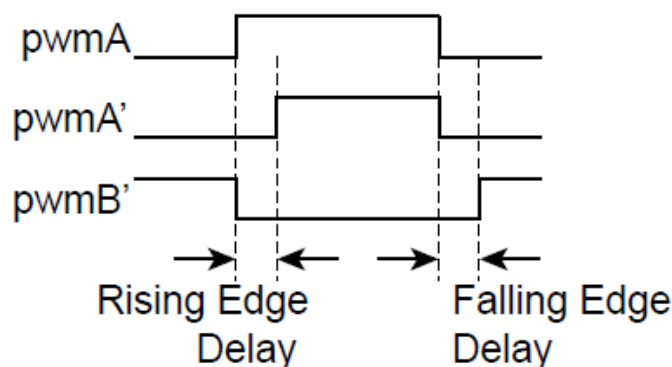
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	reserved															
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	reserved															ENABLE
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RW
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

ENABLE位写1，启用死区发生器，pwmB信号被忽略。

➤ TM4C1294 PWM模块的使用方法

- 使用PWM_GEN_0产生单相中心对称互补导通的PWM信号 (PWM0和PWM1)

5. 使能并设置死区



PWMn Dead-Band Rising-Edge Delay (**PWMnDBRISE**)寄存器

PWMn Dead-Band Falling-Edge-Delay (**PWMnDBFALL**)寄存器

延时的系统时钟周期的个数最大值**4095**



➤ TM4C1294 PWM模块的使用方法

- 使用PWM_GEN_0产生单相中心对称互补导通的PWM信号 (PWM0和PWM1)

5. 使能并设置死区

TivaWare提供了PWMDeadBandEnable函数，启动死区发生器的同时，设置PWMnDBRISE寄存器和PWMnDBFALL寄存器

```
void
PWMDeadBandEnable(uint32_t ui32Base, uint32_t ui32Gen,
                  uint16_t ui16Rise, uint16_t ui16Fall)
{
    //
    // Compute the generator's base address.
    ui32Gen = PWM_GEN_BADDR(ui32Base, ui32Gen);
    // Write the dead band delay values.
    HWREG(ui32Gen + PWM_O_X_DBRISE) = ui16Rise;
    HWREG(ui32Gen + PWM_O_X_DBFALL) = ui16Fall;
    // Enable the deadband functionality.
    HWREG(ui32Gen + PWM_O_X_DBCTL) |= PWM_X_DBCTL_ENABLE;
}
```

```
#define PWM_X_DBCTL_ENABLE 0x00000001 // Dead-Band Generator Enable
```



➤ TM4C1294 PWM模块的使用方法

- 使用PWM_GEN_0产生单相中心对称互补导通的PWM信号 (PWM0和PWM1)

5. 使能并设置死区

TivaWare提供了PWMDeadBandEnable函数，启动死区发生器的同时，设置PWMnDBRISE寄存器和PWMnDBFALL寄存器

设置PWM发生器0的死区为1us，如果系统时钟为120MHz

```
PWMDeadBandEnable(PWM0_BASE, PWM_GEN_0, 120, 120);
```



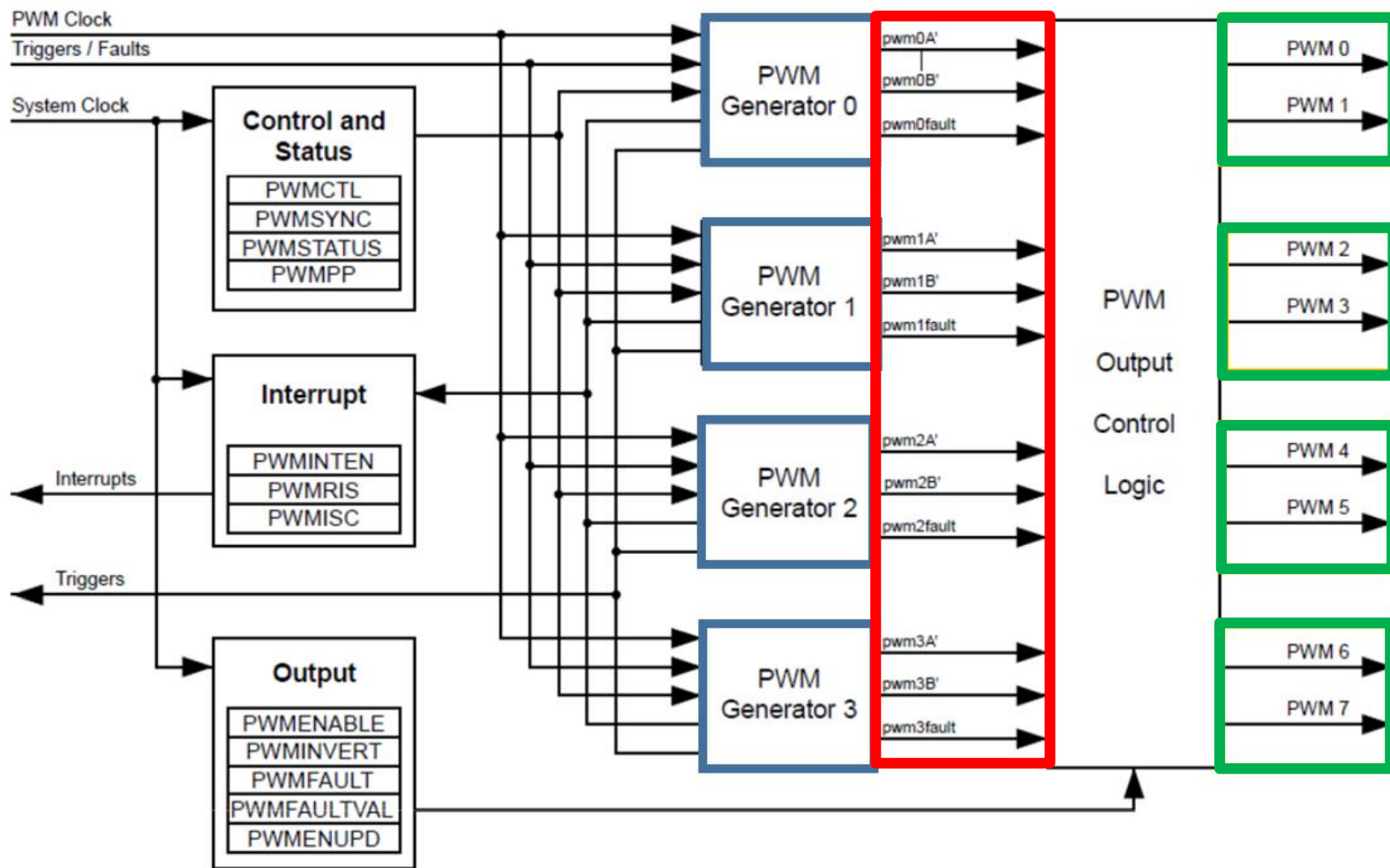
➤ TM4C1294 PWM模块的使用方法

- 使用PWM_GEN_0产生单相中心对称互补导通的PWM信号 (PWM0和PWM1)

6. 使能PWM信号的输出

PWM输出控制逻辑模块决定了





➤ TM4C1294 PWM模块的使用方法

- 使用PWM_GEN_0产生单相中心对称互补导通的PWM信号 (PWM0和PWM1)

6. 使能PWM信号的输出

PWM输出控制逻辑模块决定了 **pwmA'** 信号和 **pwmB'** 信号能否输出到 **PWM0** 到 **PWM7** 信号上

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	reserved															
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	reserved								PWM7EN	PWM6EN	PWM5EN	PWM4EN	PWM3EN	PWM2EN	PWM1EN	PWM0EN
Type	RO	RO	RO	RO	RO	RO	RO	RO	RW	RW	RW	RW	RW	RW	RW	RW
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

PWMENABLE寄存器的每一位，决定了一个PWM引脚的输出控制

➤ TM4C1294 PWM模块的使用方法

- 使用PWM_GEN_0产生单相中心对称互补导通的PWM信号 (PWM0和PWM1)

6. 使能PWM信号的输出

TivaWare提供了**PWMOutputState**函数，控制PWM0到PWM7引脚的输出

```
void
PWMOutputState(uint32_t ui32Base, uint32_t ui32PWMOutBits,
               bool bEnable)
{
    // ...
    // Read the module's ENABLE output control register and set or clear the
    // requested bits.
    //
    if(bEnable == true)
    {
        HWREG(ui32Base + PWM_O_ENABLE) |= ui32PWMOutBits;
    }
    else
    {
        HWREG(ui32Base + PWM_O_ENABLE) &= ~(ui32PWMOutBits);
    }
}
```



➤ TM4C1294 PWM模块的使用方法

- 使用PWM_GEN_0产生单相中心对称互补导通的PWM信号 (PWM0和PWM1)

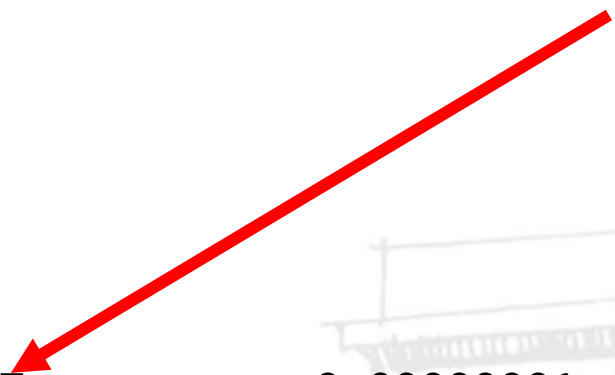
6. 使能PWM信号的输出

TivaWare提供了PWMOutputState函数，控制PWM0到PWM7引脚的输出

使能PWM0引脚和PWM1引脚的输出

```
PWMOutputState(PWM0_BASE, PWM_OUT_0_BIT|PWM_OUT_1_BIT, true);
```

```
#define PWM_OUT_0_BIT 0x00000001 // Bit-wise ID for PWM0
#define PWM_OUT_1_BIT 0x00000002 // Bit-wise ID for PWM1
```



➤ TM4C1294 PWM模块的使用方法

- 使用PWM_GEN_0产生单相中心对称互补导通的PWM信号 (PWM0和PWM1)

7. 使能并配置 中断

PWM模块的每个**PWM发生器**，都可以向中断管理器发出中断请求

```
IntMasterEnable();
```

```
IntEnable(INT_PWM0_0);
```



➤ TM4C1294 PWM模块的使用方法

- 使用PWM_GEN_0产生单相中心对称互补导通的PWM信号 (PWM0和PWM1)

7. 使能并配置 **中断**

PWM模块中的PWM Interrupt Enable (**PWMINTEN**)寄存器，决定是否允许四个PWM发生器发出中断请求

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	reserved												INTFAULT3	INTFAULT2	INTFAULT1	INTFAULT0
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RW	RW	RW	RW
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	reserved												NTPWM3	INTPWM2	INTPWM1	INTPWM0
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RW	RW	RW	RW
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

PWMINTEN寄存器的低四位，每一位控制一个PWM发生器的中断请求

➤ TM4C1294 PWM模块的使用方法

- 使用PWM_GEN_0产生单相中心对称互补导通的PWM信号(PWM0和PWM1)

7. 使能并配置中断

TivaWare提供了PWMIntEnable函数，设置PWMINTEN寄存器，开启PWM发生器的中断请求

```
void
PWMIntEnable(uint32_t ui32Base, uint32_t ui32GenFault)
{
    // ...
    // Read the module's interrupt enable register and enable interrupts
    // for the specified PWM generators.
    //
    HWREG(ui32Base + PWM_O_INTEN) |= ui32GenFault;
}
```

PWMIntEnable(PWM0_BASE, PWM_INT_GEN_0);

←

#define PWM_INT_GEN_0	0x00000001	// Generator 0 interrupt
#define PWM_INT_GEN_1	0x00000002	// Generator 1 interrupt
#define PWM_INT_GEN_2	0x00000004	// Generator 2 interrupt
#define PWM_INT_GEN_3	0x00000008	// Generator 3 interrupt



➤ TM4C1294 PWM模块的使用方法

- 使用PWM_GEN_0产生单相中心对称互补导通的PWM信号 (PWM0和PWM1)

7. 使能并配置中断

一个PWM发生器中，load、zero、AUp、ADown、BUp、BDow这几个信号，既可以**触发中断**，可以在作为**触发源**，与其他模块（如ADC模块）联动。

PWMn Interrupt and Trigger Enable (**PWMnINTEN**)寄存器，可以选择哪些信号可以触发中断，哪些信号可以作为触发源，输出到其他模块中。

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	reserved															
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	reserved		TRCMPBD	TRCMPBU	TRCMPAD	TRCMPAU	TRCNTLOAD	TRCNTZERO	reserved		INTCMPBD	INTCMPBU	INTCMPAD	INTCMPAU	INTCNTLOAD	INTCNTZERO
Type	RO	RO	RW	RW	RW	RW	RW	RW	RO	RO	RW	RW	RW	RW	RW	RW
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

➤ TM4C1294 PWM模块的使用方法

- 使用PWM_GEN_0产生单相中心对称互补导通的PWM信号 (PWM0和PWM1)

7. 使能并配置中断

TivaWare提供了PWMGenIntTrigEnable函数，设置PWMnINTEN寄存器

```
void
PWMGenIntTrigEnable(uint32_t ui32Base, uint32_t ui32Gen,
                    uint32_t ui32IntTrig)
{
    //
    // Check the arguments.
    //
    ASSERT((ui32Base == PWM0_BASE) || (ui32Base == PWM1_BASE));
    ASSERT(_PWMGenValid(ui32Gen));
    ASSERT((ui32IntTrig & ~(PWM_INT_CNT_ZERO | PWM_INT_CNT_LOAD |
                             PWM_INT_CNT_AU | PWM_INT_CNT_AD | PWM_INT_CNT_BU |
                             PWM_INT_CNT_BD | PWM_TR_CNT_ZERO |
                             PWM_TR_CNT_LOAD | PWM_TR_CNT_AU | PWM_TR_CNT_AD |
                             PWM_TR_CNT_BU | PWM_TR_CNT_BD)) == 0);

    //
    // Enable the specified interrupts/triggers.
    //
    HWREG(PWM_GEN_BADDR(ui32Base, ui32Gen) + PWM_O_X_INTEN) |= ui32IntTrig;
}
```

➤ TM4C1294 PWM模块的使用方法

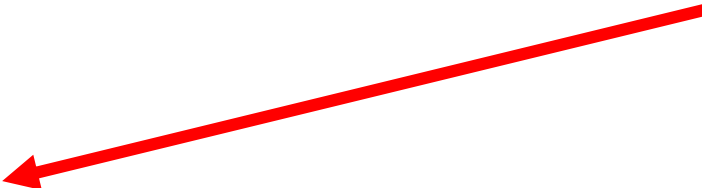
- 使用PWM_GEN_0产生单相中心对称互补导通的PWM信号 (PWM0和PWM1)

7. 使能并配置 中断

TivaWare提供了 **PWMGenIntTrigEnable** 函数，设置PWMn**INTEN**寄存器

使能PWM发生器0的zero中断

PWMGenIntTrigEnable(PWM0_BASE, PWM_GEN_0, PWM_INT_CNT_ZERO);



```
#define PWM_INT_CNT_ZERO      0x00000001 // Int if COUNT = 0
#define PWM_INT_CNT_LOAD     0x00000002 // Int if COUNT = LOAD
#define PWM_INT_CNT_AU       0x00000004 // Int if COUNT = CMPA U
#define PWM_INT_CNT_AD       0x00000008 // Int if COUNT = CMPA D
#define PWM_INT_CNT_BU       0x00000010 // Int if COUNT = CMPA U
#define PWM_INT_CNT_BD       0x00000020 // Int if COUNT = CMPA D
```



➤ TM4C1294 PWM模块的使用方法

- 使用PWM_GEN_0产生单相中心对称互补导通的PWM信号 (PWM0和PWM1)

7. 使能并配置 中断

TivaWare提供了 **PWMGenIntRegister** 函数为PWM发生器注册中断服务函数

```
PWMGenIntRegister(PWM0_BASE, PWM_GEN_0, PWM0_isr);
```



➤ TM4C1294 PWM模块的使用方法

- 使用PWM_GEN_0产生单相中心对称互补导通的PWM信号 (PWM0和PWM1)

8. 使能PWM发生器

设置完成后，需要使能PWM发生器，让其计数器开始工作

PWMn Control (**PWMnCTL**)寄存器，用于PWM发生器模块的配置

PWMn Control (PWMnCTL)

PWM0 base: 0x4002.8000

Offset 0x040

Type RW, reset 0x0000.0000

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	reserved													LATCH	MINFLTPER	FLTSRC
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RW	RW	RW
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	DBFALLUPD		DBRISEUPD		DBCTLUPD		GENBUPD		GENAUPD		CMPBUPD	CMPAUPD	LOADUPD	DEBUG	MODE	ENABLE
Type	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

➤ TM4C1294 PWM模块的使用方法

- 使用PWM_GEN_0产生单相中心对称互补导通的PWM信号 (PWM0和PWM1)

8. 使能PWM发生器

TivaWare提供了PWMGenEnable函数，给PWMnCTL寄存器的ENABLE位置1，使能指定的PWM发生器

```
void
PWMGenEnable(uint32_t ui32Base, uint32_t ui32Gen)
{
    //
    // Check the arguments.
    ASSERT((ui32Base == PWM0_BASE) || (ui32Base == PWM1_BASE));
    ASSERT(_PWMGenValid(ui32Gen));

    #define PWM_X_CTL_ENABLE          0x00000001 // PWM Block Enable

    //
    // Enable the PWM generator.
    //
    HWREG(PWM_GEN_BADDR(ui32Base, ui32Gen) + PWM_O_X_CTL) |= PWM_X_CTL_ENABLE;
}
```

PWMGenEnable(PWM0_BASE, PWM_GEN_0);



➤ TM4C1294 PWM模块的使用方法

- 使用PWM_GEN_0产生单相中心对称互补导通的PWM信号 (PWM0和PWM1)

9. 编写**中断服务函数**，根据功能要求更新占空比

PWMn Interrupt Status and Clear (**PWMnISC**)寄存器

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	reserved															
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	reserved										INTCMPBD	INTCMPBU	INTCMPAD	INTCMPAU	INTCNTLOAD	INTCNTZERO
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RW1C	RW1C	RW1C	RW1C	RW1C	RW1C
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

读取**PWMnISC**寄存器，可以查看中断的来源

写**PWMnISC**寄存器，可以清除中断

➤ TM4C1294 PWM模块的使用方法

- 使用PWM_GEN_0产生单相中心对称互补导通的PWM信号 (PWM0和PWM1)

9. 编写**中断服务函数**，根据功能要求更新占空比

```
void PWM0_isr(void){  
  
    PWMGenIntClear(PWM0_BASE, PWM_GEN_0, PWM_INT_CNT_ZERO);  
    .....//根据功能要求计算占空比, 设置duty变量  
    PWMPulsewidthSet(PWM0_BASE, PWM_OUT_0, duty);  
}
```

由于开启了死区发生器，所以改变PWM0信号的占空比的同时，也改变了PWM1信号的占空比

PWM0和PWM1为一组带死区的中心对称互补导通PWM信号



➤ 扩展

- PWM模块自动触发ADC采样
- PWM的故障输入



➤ 小节

- 脉冲宽度调制概述
- PWM模块的工作原理
- PWM模块的使用方法：产生中心对称互补导通的PWM信号



谢谢!

