

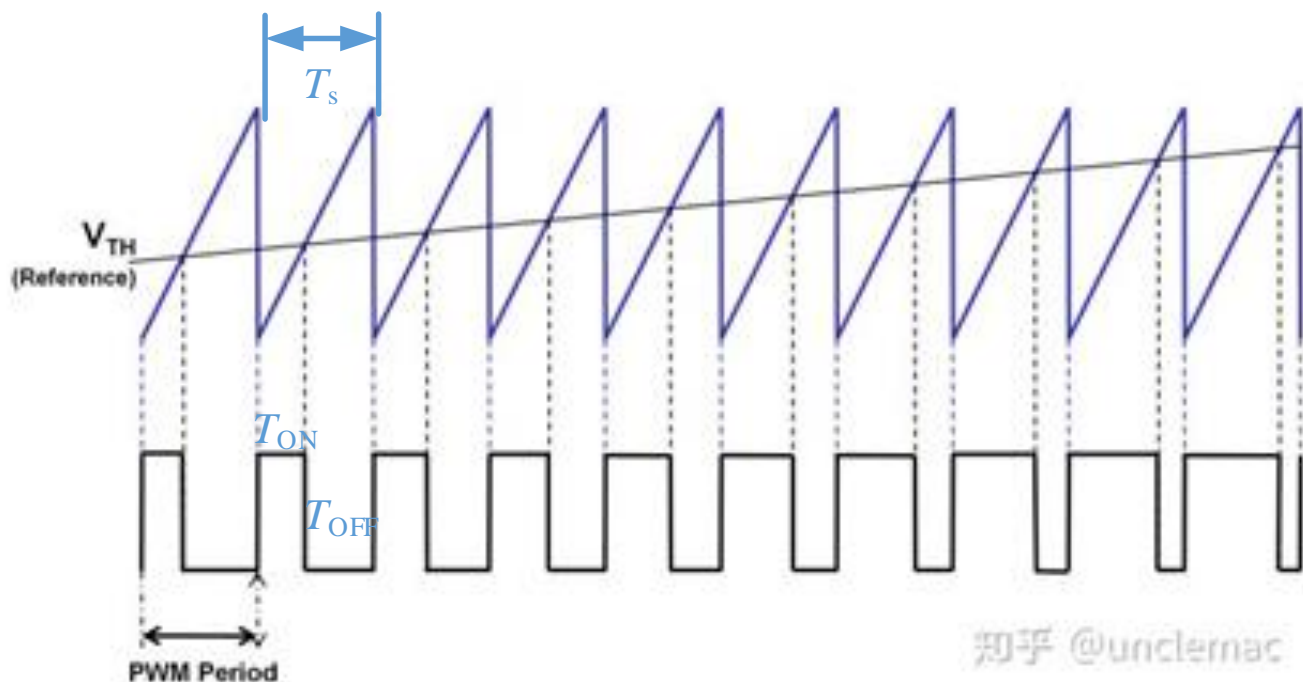
# PWM模块



- 
- **脉冲宽度调制**（Pulse Width Modulation，缩写：**PWM**），简称**脉宽调制**，是将模拟信号变换为脉冲电压的一种技术。
  - 一般变换后脉冲的**周期固定**，但一个周期内**脉冲的宽度**会依模拟信号的大小而改变，从而使一个周期内的脉冲的**平均电压**与给定模拟信号呈线性关系。

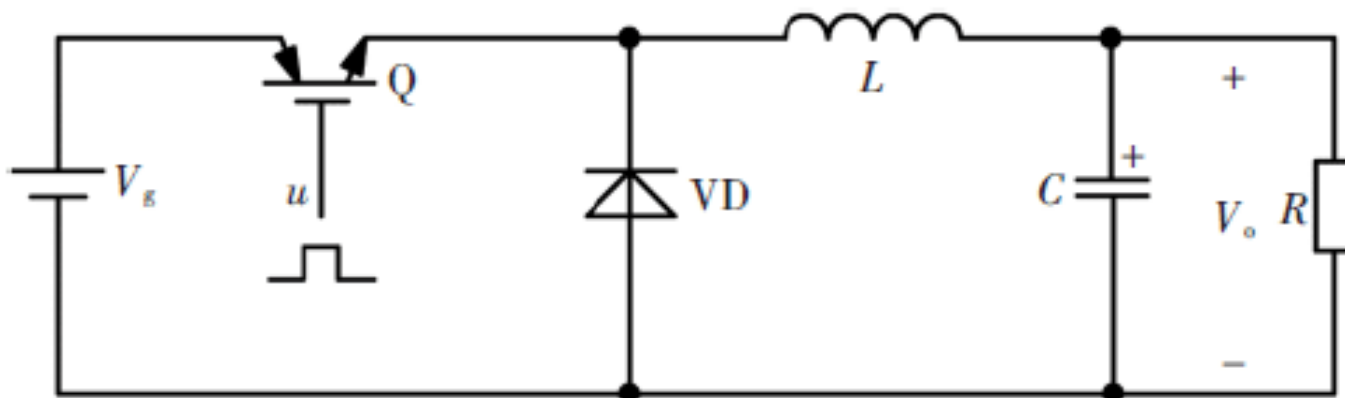


- PWM脉冲一般由载波和调制波比较获得
- 载波一般为锯齿波或三角波。
- 锯齿波或三角波的周期被称为载波周期 (PWM period) 或开关周期 (Switching period)，记为 ( $T_s$ )。
- 调制波为一个载波周期内，期望输出的电压平均值。
- 一个载波周期内，输出高电平的时间所占的比例，称作占空比(Duty Cycle)，  $D = T_{ON} / T_s$ 。



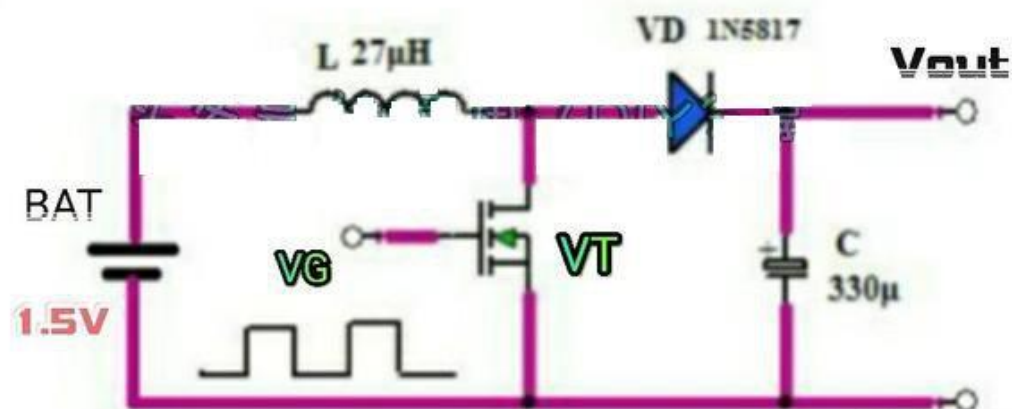
## ➤ PWM技术的应用

### - Buck变换器:



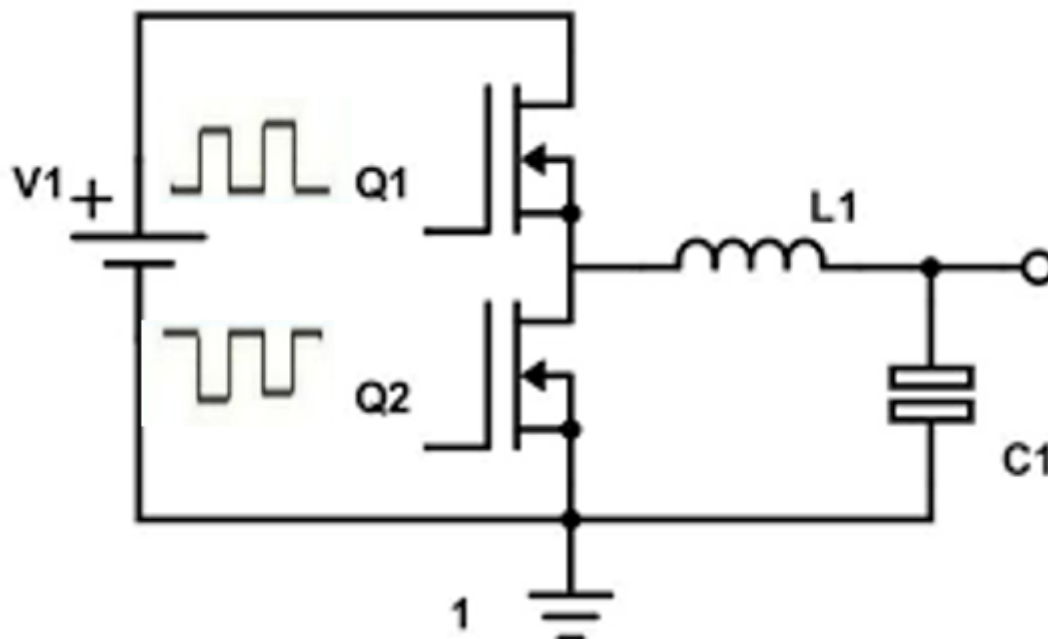
## ➤ PWM技术的应用

### - Boost变换器



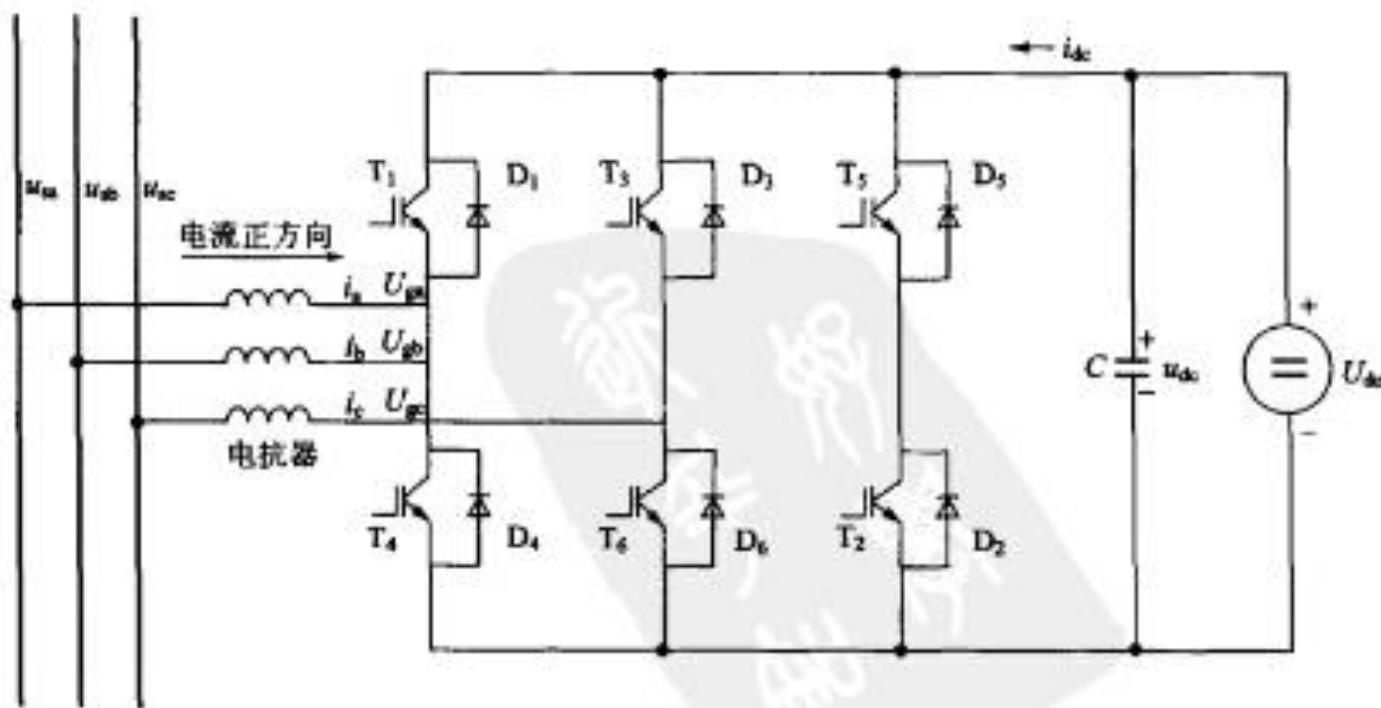
## ➤ PWM技术的应用

- 同步整流变换器



## ➤ PWM技术的应用

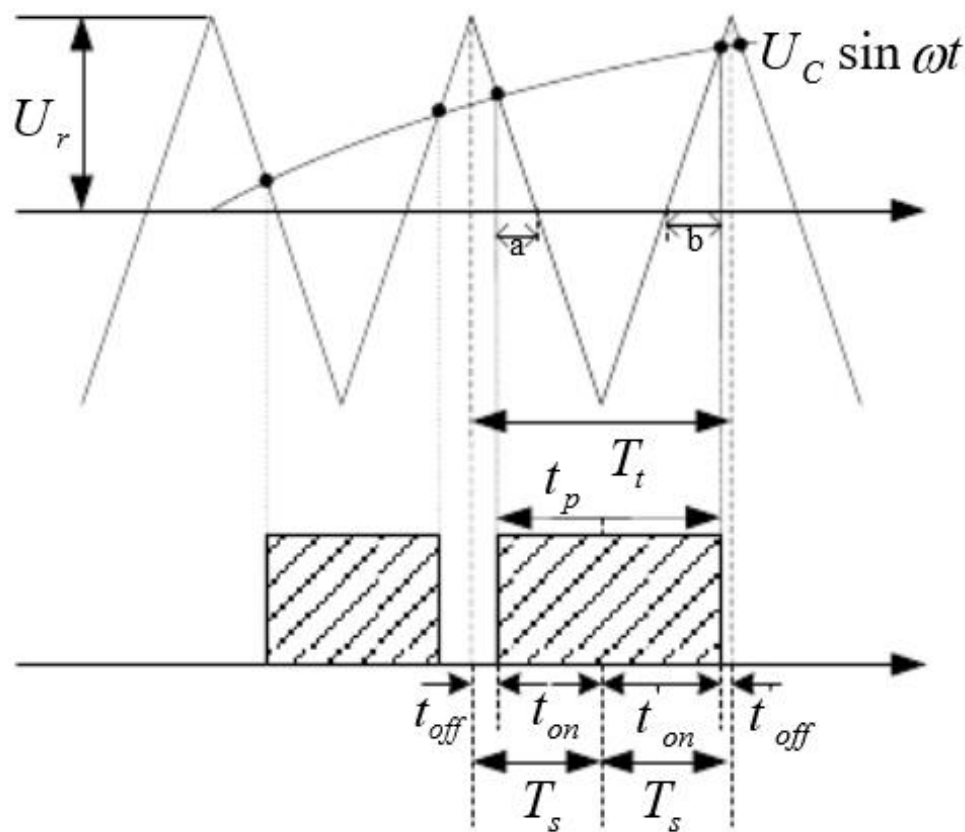
### - 电压源逆变器:



## ➤ PWM脉冲的生成方式:

### – 自然采样法

- 常用模拟器件实现
- 载波与调制波实时比较

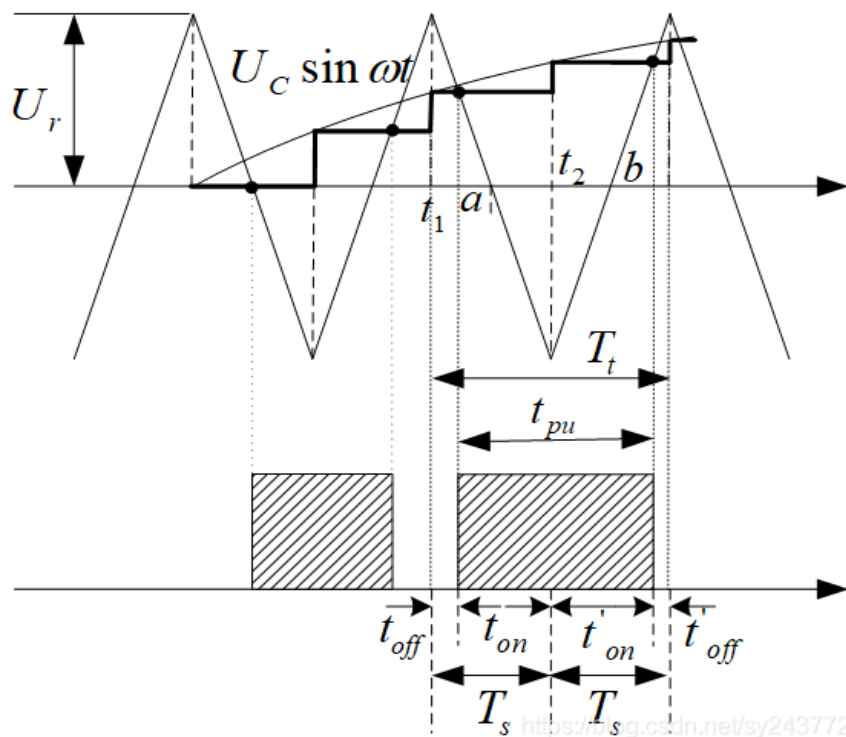




# ➤ PWM脉冲的生成方式:

## - 规则采样法

- 利用微控制器实现PWM的常用技术，广泛应用于测量，通信，功率控制与变换等许多领域
- 按照一定的规则对调制波进行采样,其余时间调制波不变



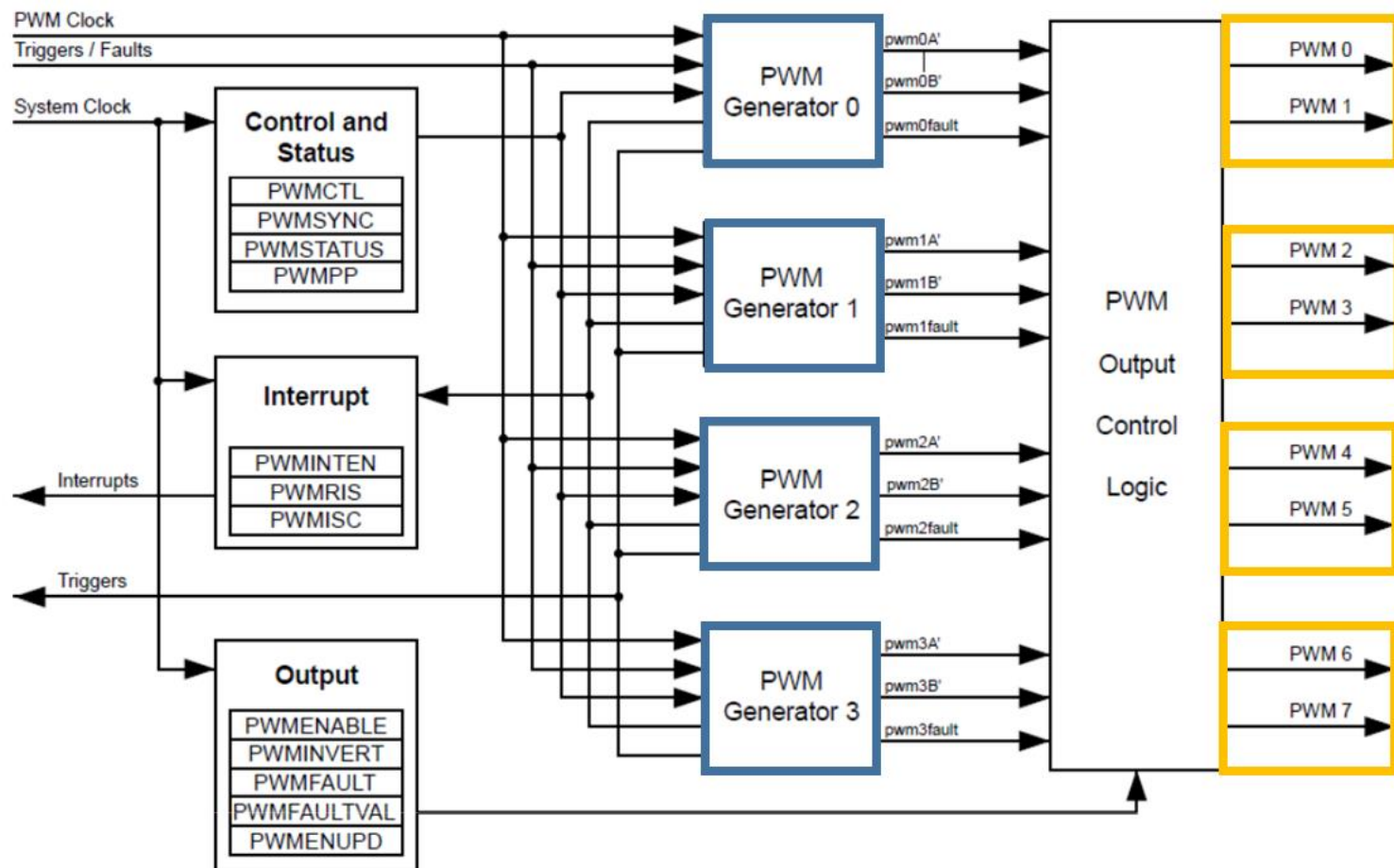
## ➤ TM4C 1294的PWM模块

- 提供了四个PWM发生器
- 使用规则采样法生成8路PWM输出信号
  - 四个PWM发生器分别是PWM\_GEN\_0, PWM\_GEN\_1, PWM\_GEN\_2和PWM\_GEN\_3
  - PWM\_GEN\_0产生PWM0和PWM1。
  - PWM\_GEN\_1产生PWM2和PWM3。
  - PWM\_GEN\_2产生PWM4和PWM5。
  - PWM\_GEN\_3产生PWM6和PWM7。



## ➤ TM4C 1294的PWM模块

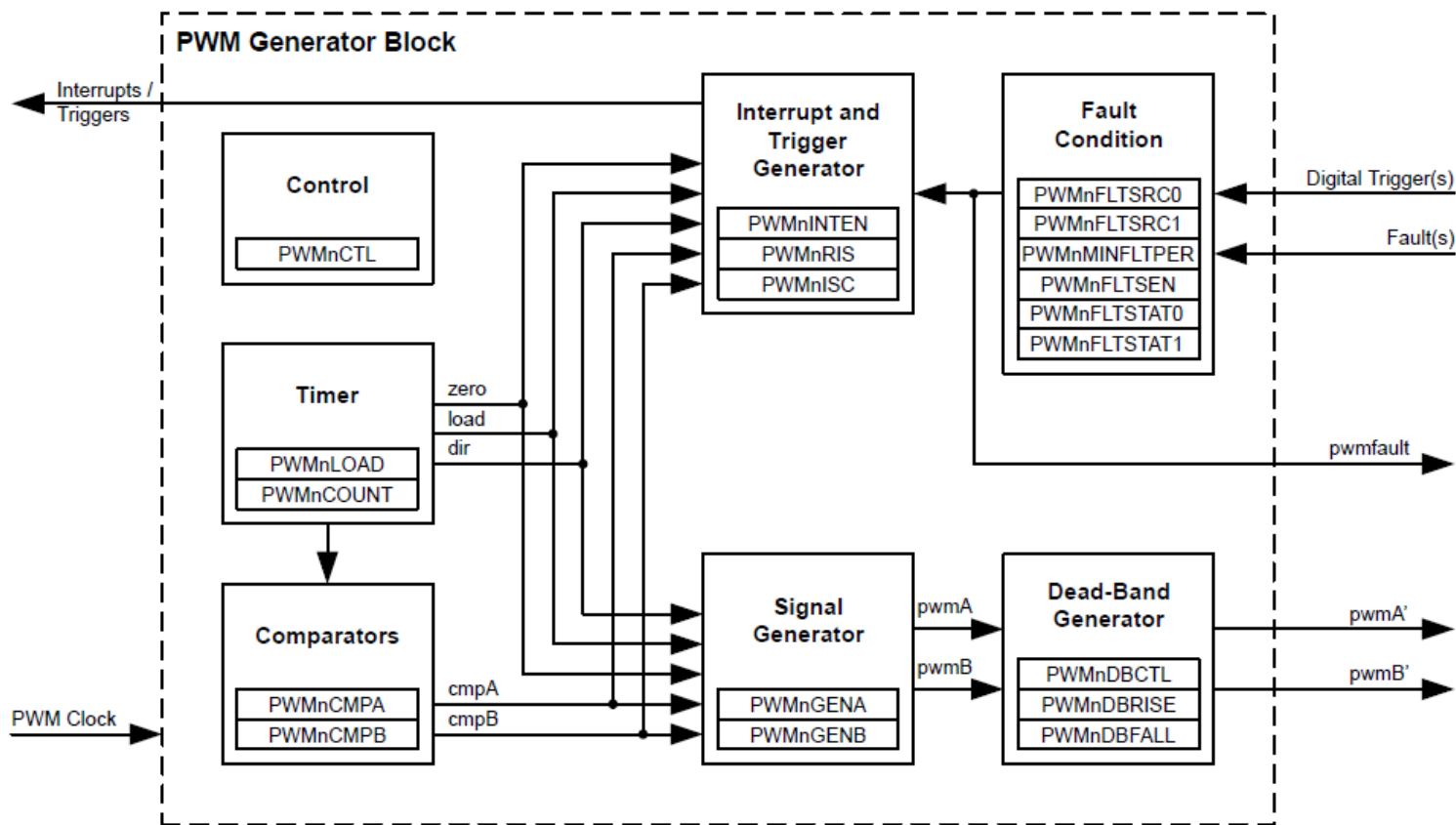
- 提供了四个PWM发生器
- 使用规则采样法生成8路PWM输出信号



# ➤ TM4C 1294的PWM模块

## - PWM发生器

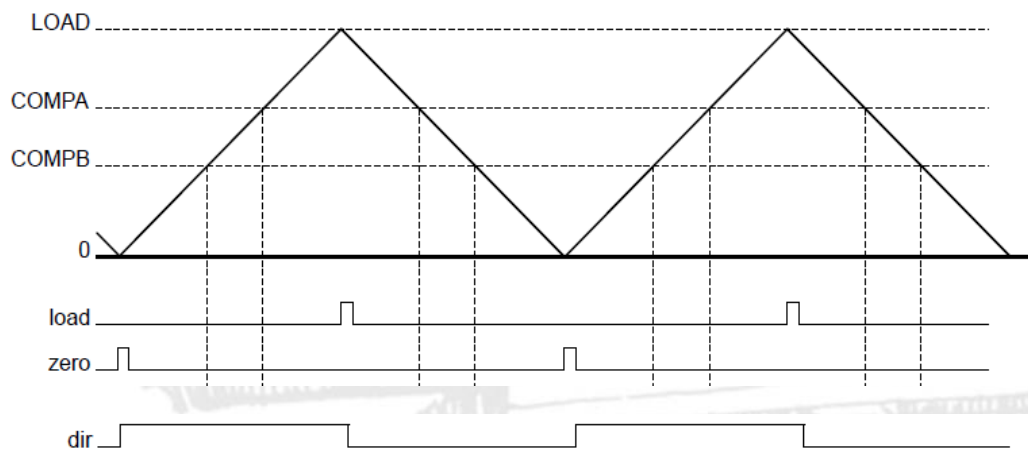
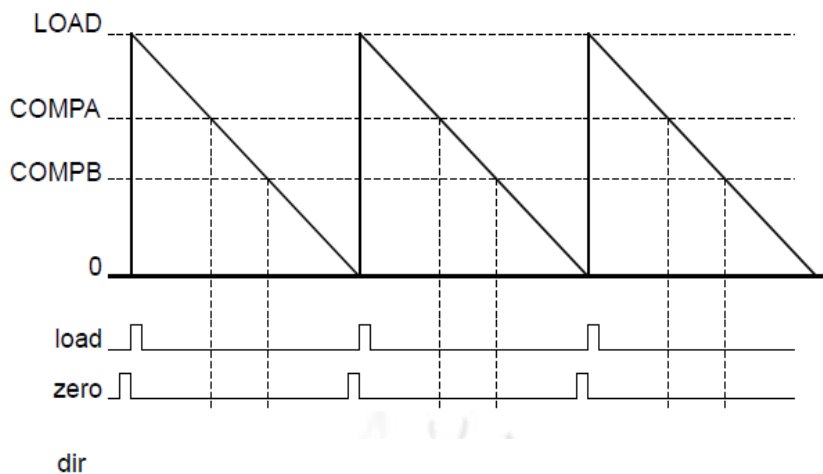
- 由计数器模块、比较器模块、信号发生器模块、死区发生器模块、控制寄存器、中断控制器、故障处理等模块组成



# ➤ TM4C 1294的PWM模块

## – PWM发生器----计数器

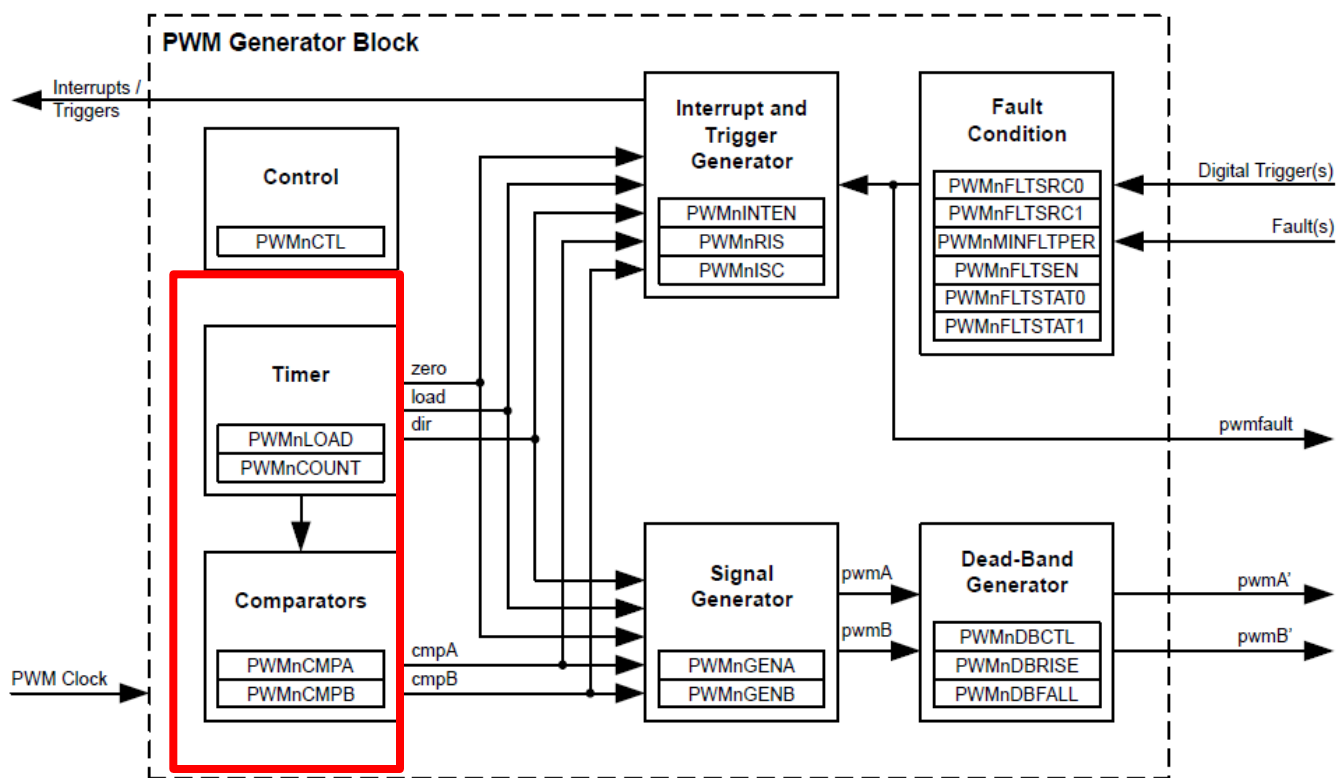
- 用计数器模拟载波，可以模拟锯齿波，也可以模拟三角波
- 16位计数
- 计数器的计数值 (COUNT) 从0开始计数，计数的最大值计做LOAD，
- 计数到0，发出zero信号，计数到LOAD，发出load信号。
- Dir信号指示计数方向



# ➤ TM4C 1294的PWM模块

## - PWM发生器----比较器

- PWM发生器的比较器模块中，有两个比较寄存器，计做COMP<sub>A</sub>和COMP<sub>B</sub>，用于模拟调制波。



# ➤ TM4C 1294的PWM模块

## – PWM发生器----比较器

- PWM发生器的比较器模块中，有两个比较寄存器，计做COMPA和COMPB，用于模拟调制波。
- 计数器在计数的过程中，不断的跟这两个寄存器的值比较
  - 如果计数值 (COUNT) 与COMPA相等，就产生compA事件。
    - » 如果当前计数器正在向下计数，那么产生的compA事件称作 $A_{down}$ ，
    - » 如果当前计数器正在向上计数，那么产生的compA事件，称作 $A_{Up}$ 。
  - 如果计数值 (COUNT) 与COMPB相等，就产生compB事件。
    - » 如果当前计数器正在向下计数，那么产生的compB事件称作 $B_{down}$
    - » 如果计数器正在向上计数，那么产生的compB事件，称作 $B_{Up}$ 。

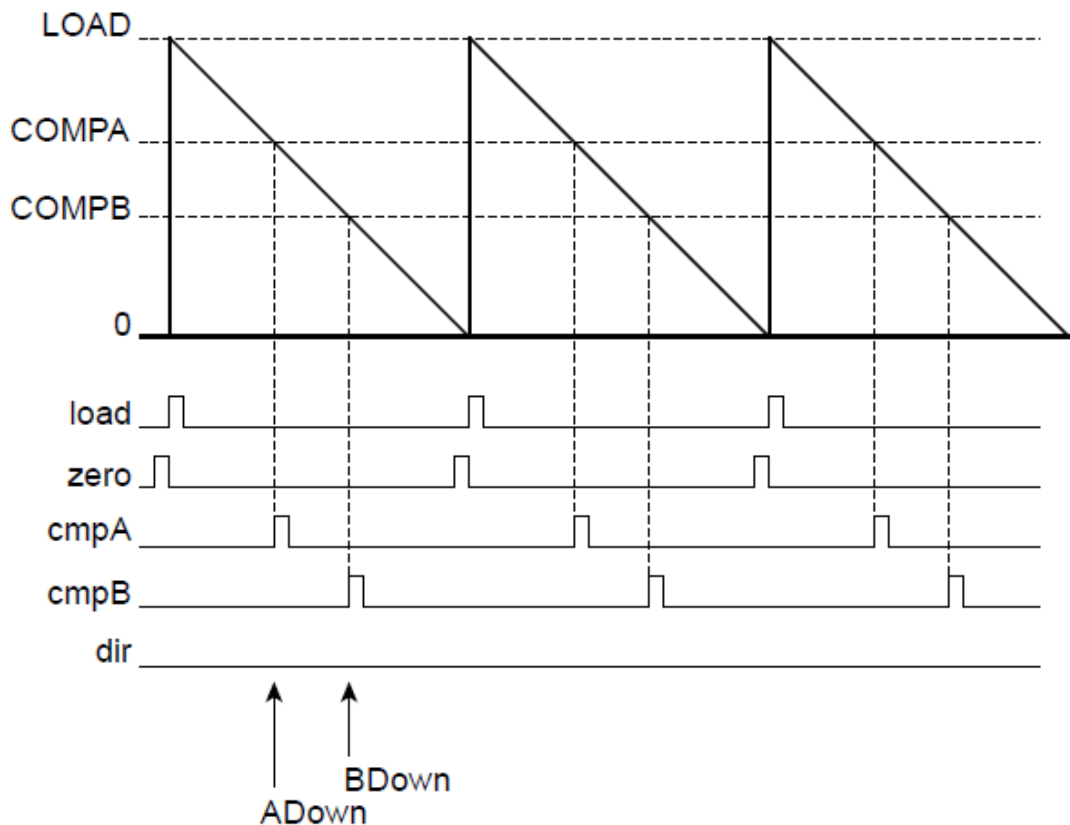




# ➤ TM4C 1294的PWM模块

## – PWM发生器----比较器

- PWM发生器的比较器模块中，有两个比较寄存器，计做COMPA和COMPB，用于模拟调制波。
- 计数器在计数的过程中，不断的跟这两个寄存器的值比较

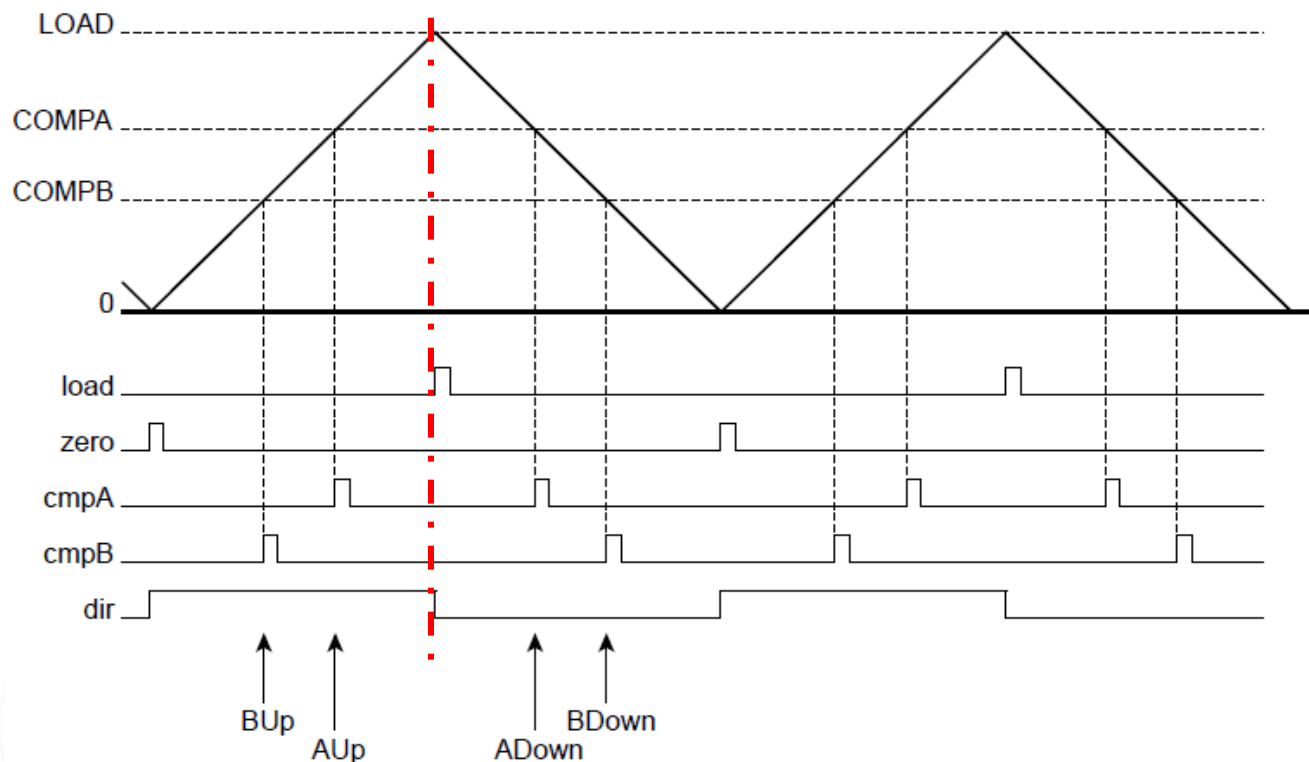




# ➤ TM4C 1294的PWM模块

## – PWM发生器----比较器

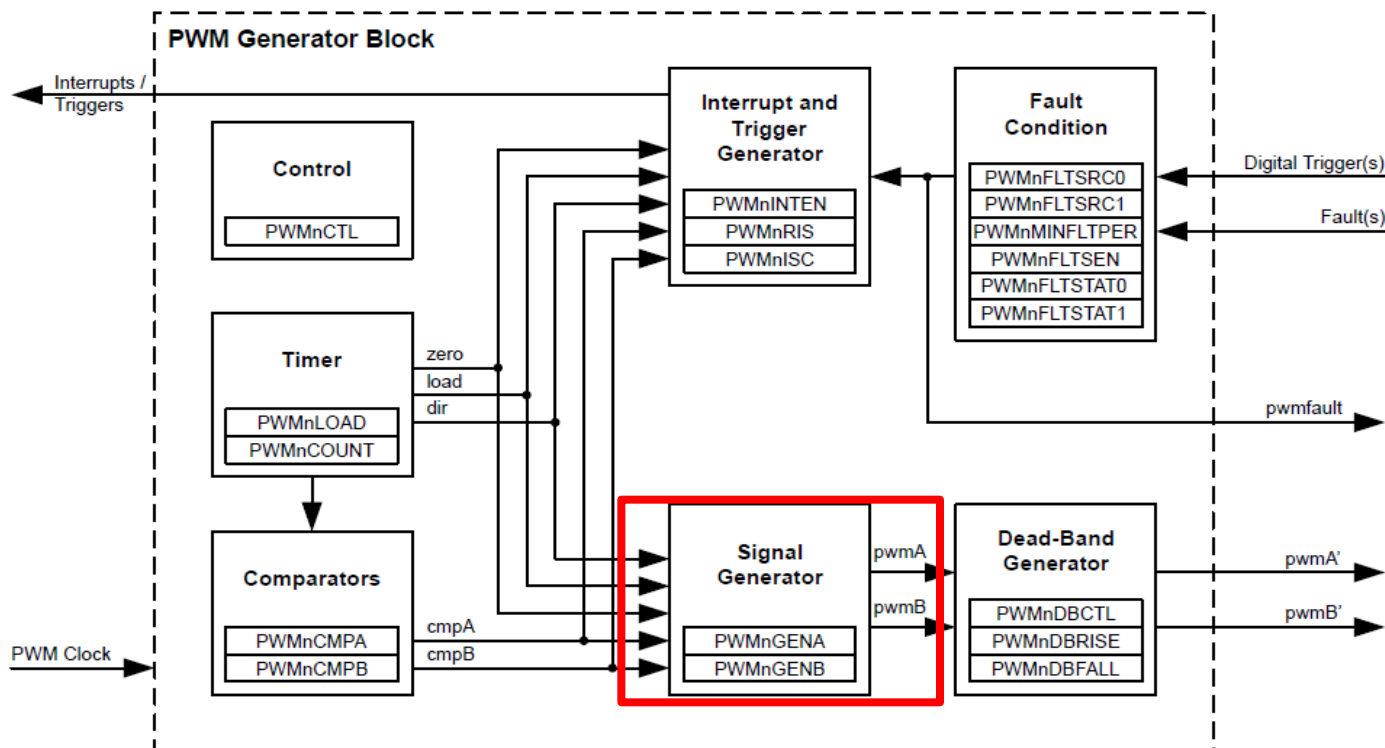
- load、zero、compA (AUp、ADown)、compB (BUp、BDown)、dir这几个事件的生成。



# ➤ TM4C 1294的PWM模块

## - PWM发生器---信号发生器

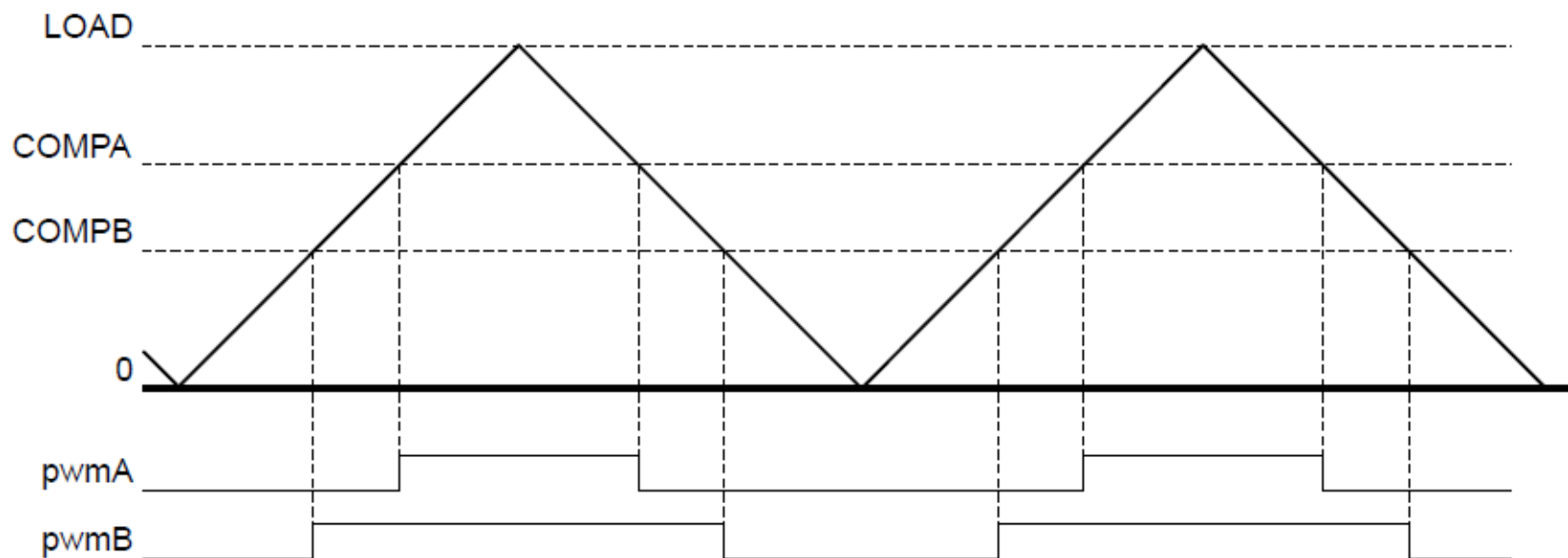
- load、zero、compA ( $A_{Up}$ 、 $A_{Down}$ )、compB ( $B_{Up}$ 、 $B_{Down}$ )、dir这几个事件在信号发生器中经过处理，产生PWM信号pwmA和pwmB。



# ➤ TM4C 1294的PWM模块

## – PWM发生器---信号发生器

- load、zero、compA ( $A_{Up}$ 、 $A_{Down}$ )、compB ( $B_{Up}$ 、 $B_{Down}$ )、dir这几个事件在信号发生器中经过处理，产生PWM信号pwmA和pwmB。



$A_{Up}$ : 设置pwmA为高电平

$A_{down}$ : 设置pwmA为低电平

占空比 $D = (LOAD - COMP A) / LOAD$

$B_{Up}$ : 设置pwmB为高电平

$B_{down}$ : 设置pwmB为低电平

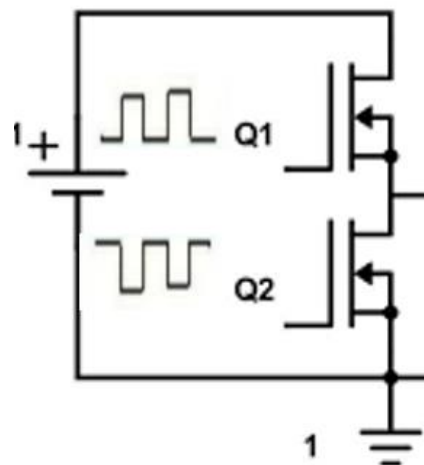
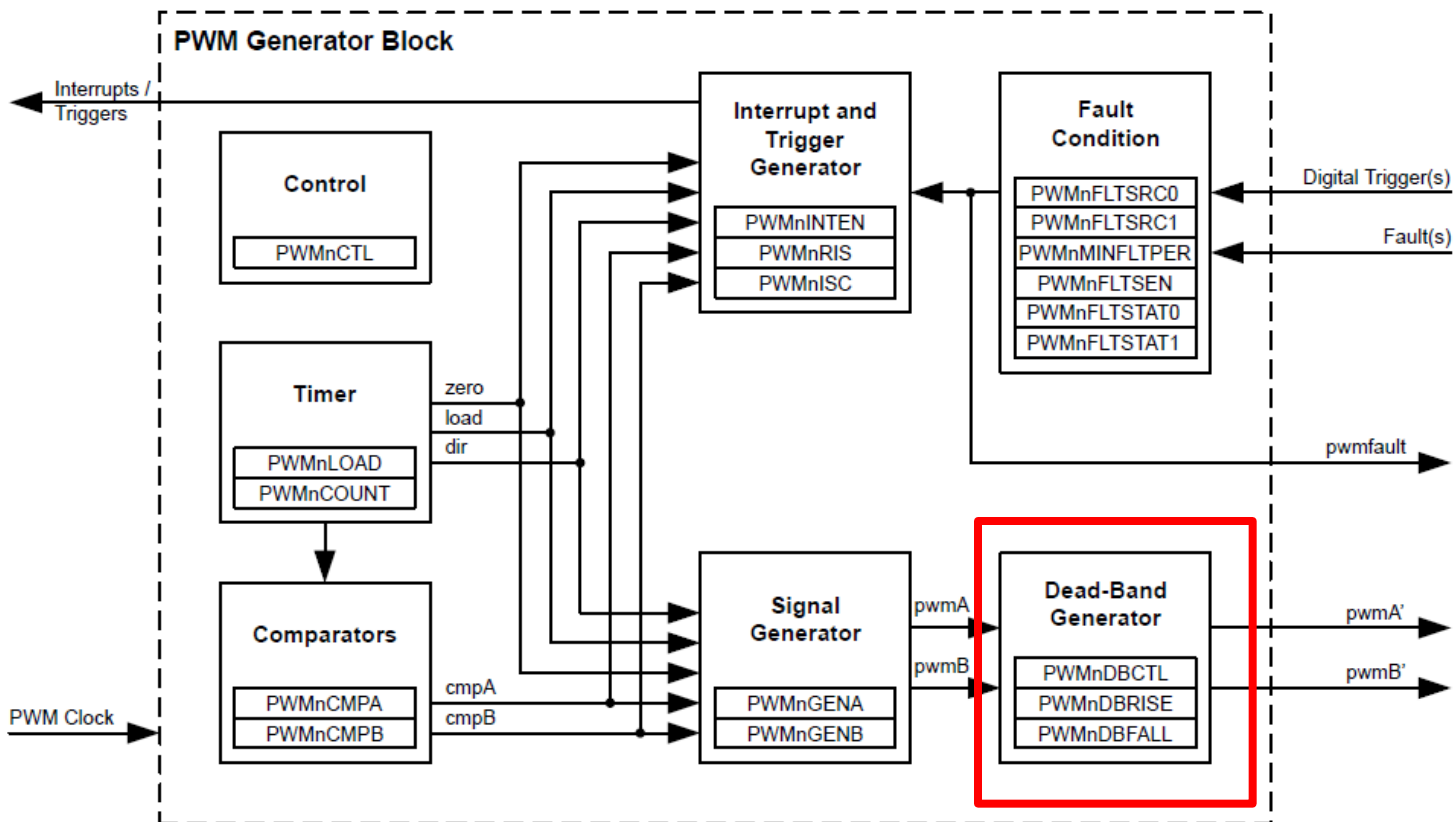
$D = (LOAD - COMP B) / LOAD$



# ➤ TM4C 1294的PWM模块

## - PWM发生器---死区发生器

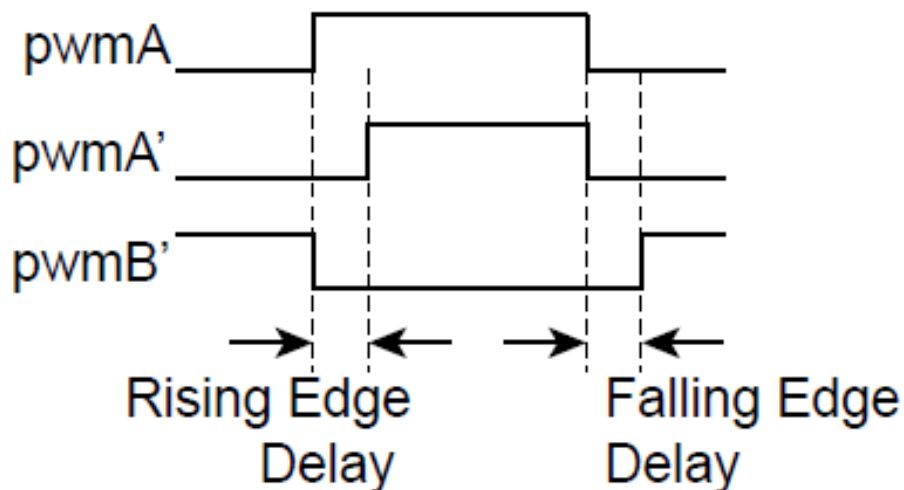
- 同步整流变换器和电压源逆变器需要带死区的互补PWM信号，驱动一个桥臂的上管和下管。



# ➤ TM4C 1294的PWM模块

## – PWM发生器----死区发生器

- 死区发生器对pwmA和pwmB信号进行整形，插入死区，产生pwmA'和pwmB'信号。
  - 如果不使用死区功能，死区发生器处于直通模式，pwmA信号直接变成pwmA'信号。pwmB信号直接变成pwmB'信号。
  - 如果使用死区功能，pwmB信号被忽略。死区发生器发生器实际上是一个延迟开通单元，延迟开通的时间可以设定。



## ➤ TM4C 1294的PWM模块

### – PWM输出控制逻辑

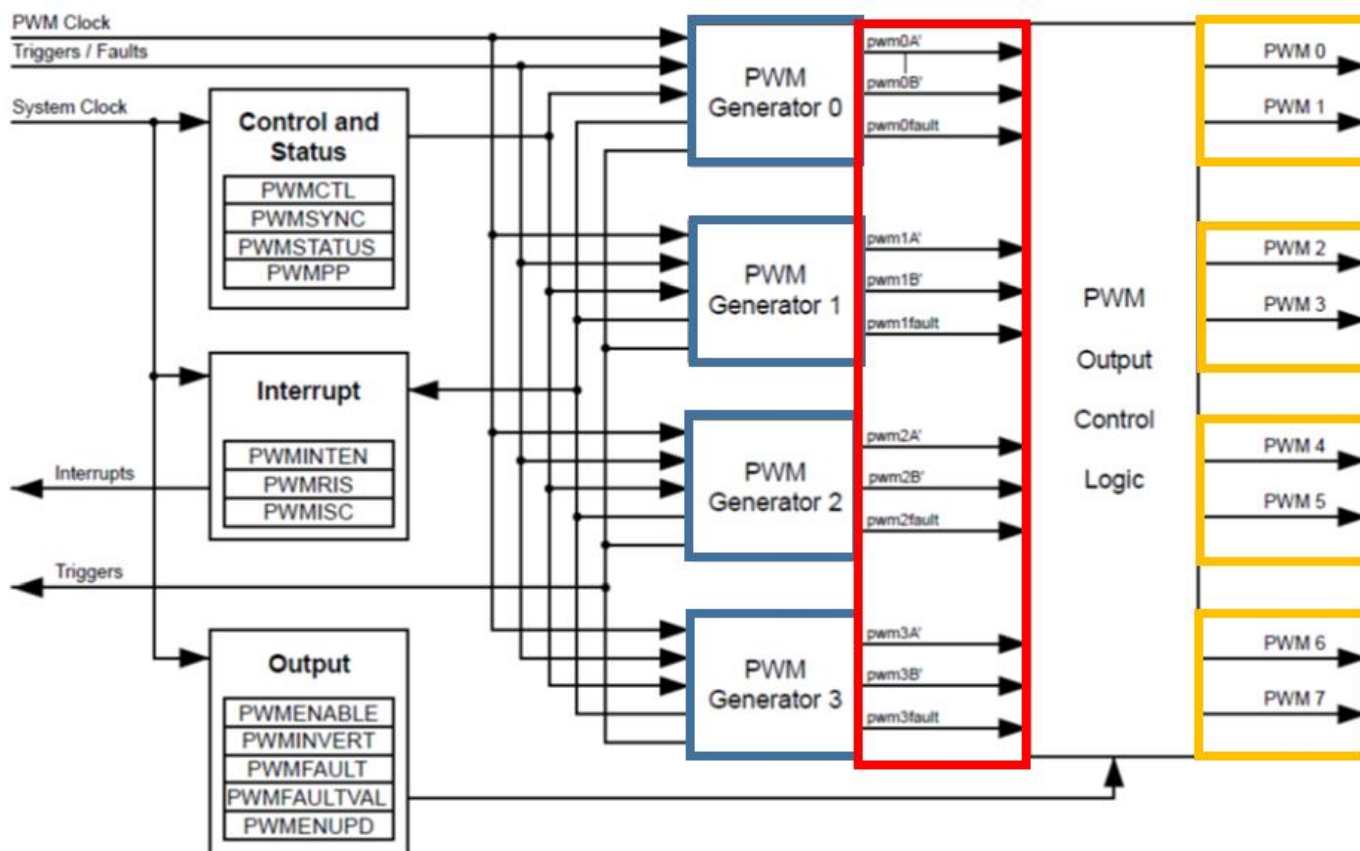
- PWM发生器模块最终产生的信号为pwmA'和pwmB'
  - PWM\_GEN\_0产生的pwmA'和pwmB'信号计做pwm0A'和pwm0B'
  - PWM\_GEN\_1产生的pwmA'和pwmB'信号计做pwm1A'和 pwm1B'
  - PWM\_GEN\_2产生的pwmA'和pwmB'信号计做pwm2A'和pwm2B'
  - PWM\_GEN\_3产生的pwmA'和pwmB'信号计做pwm3A'和pwm3B'
- PWM输出控制逻辑模块可以控制pwmXA'和pwmXB'  
(X=0,1,2,3) 信号最终是否输出到MOPWMY (Y=0,1,2,3,4,5,6,7) 引脚上。



# ➤ TM4C 1294的PWM模块

## – PWM输出控制逻辑

- PWM输出控制逻辑模块可以控制pwmXA'和pwmXB' (X=0,1,2,3) 信号最终是否输出到MOPWMY





# ➤ TM4C 1294的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。





# ➤ TM4C 1294的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模块计做PWM0

**SysCtlPeripheralEnable**(SYSCTL\_PERIPH\_PWM0);



# ➤ TM4C 1294的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MOPWMY(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.



# ➤ TM4C 1294的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
```



# ➤ TM4C 1294的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

产生中心对称，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



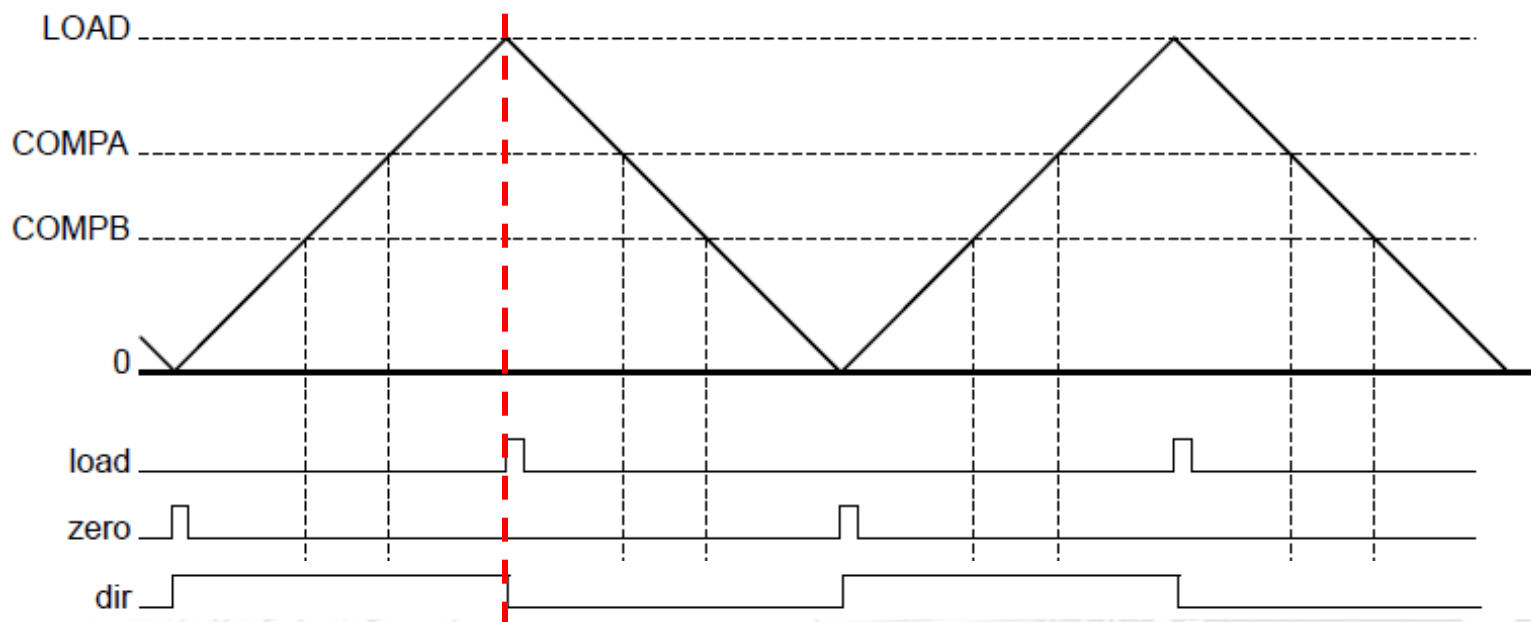
# ➤ TM4C 1294的PWM模块的使用方法

- 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

产生**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**



# ➤ TM4C 1294的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、BUp、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: BUp信号

ACTLOAD: load信号

ACTCMPBD: BDown信号

ACTCMPAU: AUp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

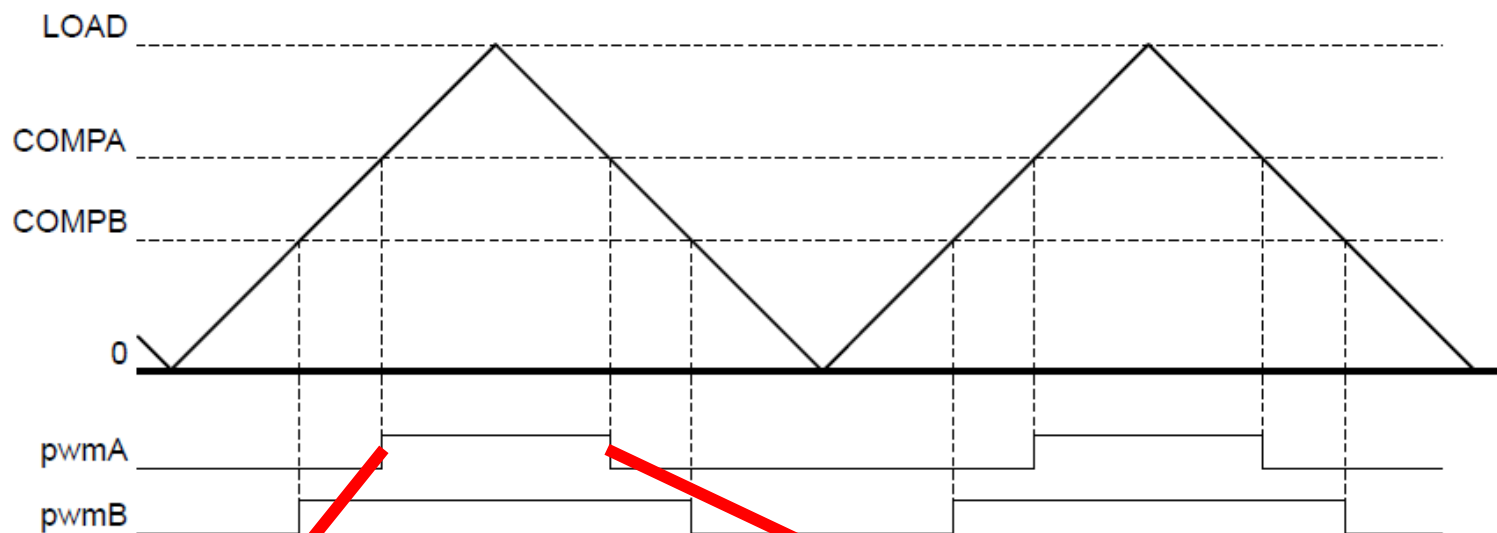


# • TM4C 1294的PWM模块的使用方法

- 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。





# ➤ TM4C 1294的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





```

void PWMGenConfigure(uint32_t ui32Base, uint32_t ui32Gen, uint32_t ui32Config)
{
    ui32Gen = PWM_GEN_BADDR(ui32Base, ui32Gen);
    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)
    {
        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);

        .....
    }
}

```



# ➤ TM4C 1294的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

PWMGenConfigure函数自动判断计数模式

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);
```



# ➤ TM4C 1294的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载波周期和PWM的占空比

PWM的载波周期由PWMn Load (PWMnLOAD)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  $(2 * \text{PWMnLOAD}) / \text{SYSTEMCLOCK}$



# ➤ TM4C 1294的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载波周期和PWM的占空比

PWM的载波周期由PWMn Load (PWMnLOAD)寄存器决定

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



```
void PWMGenPeriodSet(uint32_t ui32Base, uint32_t ui32Gen,
                    uint32_t ui32Period)
{
    .....
    ui32Gen = PWM_GEN_BADDR(ui32Base, ui32Gen);
    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;
    }
}
```



## ➤ TM4C 1294的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载波周期和PWM的占空比

PWM的载波周期由PWMn Load (PWMnLOAD)寄存器决定

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

PWMPGenPeriodSet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PWMnLOAD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PWMnLOAD寄存器。

也就是说，不管载波是何种波形，PWMPGenPeriodSet函数的ui32Period参数都是整个载波周期的计数时长。

PWMPGenPeriodSet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



# ➤ TM4C 1294的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载波周期和PWM的占空比

PWM的载波周期由PWMn Load (PWMnLOAD)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);
```





# ➤ TM4C 1294的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载波周期和PWM的

PWM的占空比由COMP A和COMP B决定

COMP A和COMP B分别对应PWMn Compare A (PWMnCMPA)和PWMn Compare B (PWMnCMPB)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;
    .....
    ui32GenBase = PWM_OUT_BADDR(ui32Base, ui32PWMOut);
    if(HWREG(ui32GenBase + PWM_O_X_CTL) & PWM_X_CTL_MODE)
    {
        ui32Width /= 2;
    }

    ui32Reg = HWREG(ui32GenBase + PWM_O_X_LOAD);
    ASSERT(ui32Width < ui32Reg);
    ui32Reg = ui32Reg - ui32Width;
    if(PWM_IS_OUTPUT_ODD(ui32PWMOut))
    {
        HWREG(ui32GenBase + PWM_O_X_CMPB) = ui32Reg;
    }
    else
    {
        HWREG(ui32GenBase + PWM_O_X_CMPA) = ui32Reg;
    }
}
```



# ➤ TM4C 1294的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载波周期和PWM的占空比

PWM的占空比由COMP A和COMP B决定

COMP A和COMP B分别对应PWMn Compare A (PWMnCMPA)和PWMn Compare B (PWMnCMB)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PWMnCMPA和PWMnCMB寄存器之前，会读出PWMnLOAD寄存器的值，然后用PWMnLOAD寄存器的值减去ui32Width后，再赋值给PWMnCMPA和PWMnCMB寄存器



占空比就变为  $ui32Width / LOAD$



# ➤ TM4C 1294的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);
```



# ➤ TM4C 1294的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

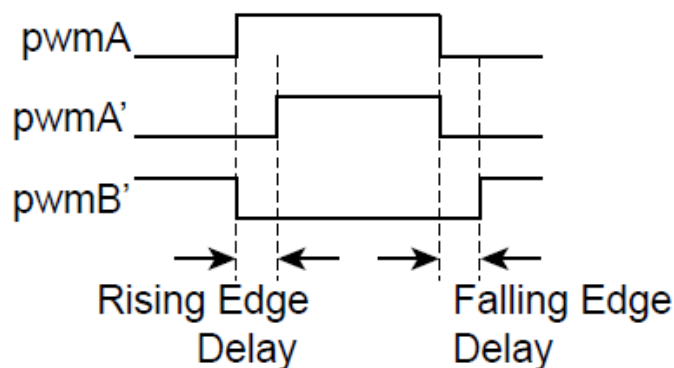
ENABLE位写1，启用死区发生器，pwmB信号被忽略。



# ➤ TM4C 1294的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**



# ➤ TM4C 1294的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PWMDDeadBandEnable函数，启动死区发生器的同时，设置PWMnDBRISE寄存器和PWMnDBFALL寄存器

```
void PWMDDeadBandEnable(uint32_t ui32Base, uint32_t ui32Gen,  
                        uint16_t ui16Rise, uint16_t ui16Fall)  
{  
    .....  
    ui32Gen = PWM_GEN_BADDR(ui32Base, ui32Gen);  
    HWREG(ui32Gen + PWM_O_X_DBRISE) = ui16Rise;  
    HWREG(ui32Gen + PWM_O_X_DBFALL) = ui16Fall;  
    HWREG(ui32Gen + PWM_O_X_DBCTL) |= PWM_X_DBCTL_ENABLE;  
}
```



# ➤ TM4C 1294的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);



# ➤ TM4C 1294的PWM模块的使用方法

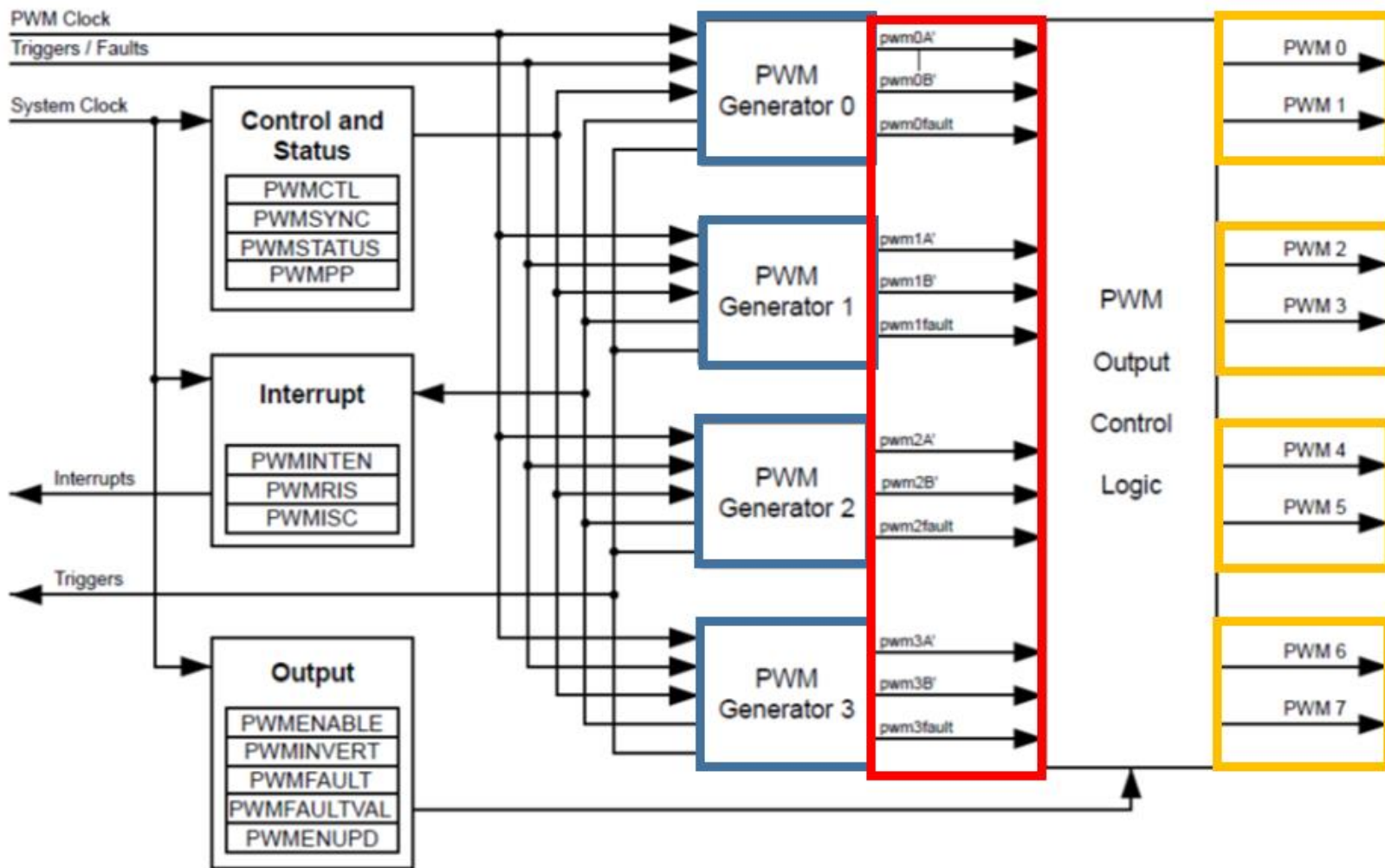
- 使用PWM\_GEN\_0产生单相**中心对称互补导通**的PWM信号  
(PWM0和PWM1)

## 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PWM1信号上。







# ➤ TM4C 1294的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 Output Enable (**PWMENABLE**)寄存器，决定了pwmA'信号和pwmB'信号能否输出到PWM0到PWM1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。



# ➤ TM4C 1294的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PWM1引脚的输出

```
void PWMOutputState(uint32_t ui32Base, uint32_t ui32PWMOutBits,  
                    bool bEnable)  
{  
    .....  
    if(bEnable == true)  
    { HWREG(ui32Base + PWM_O_ENABLE) |= ui32PWMOutBits; }  
    else  
    { HWREG(ui32Base + PWM_O_ENABLE) &= ~(ui32PWMOutBits); }  
}
```



# ➤ TM4C 1294的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PWM1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



# ➤ TM4C 1294的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);
```



# ➤ TM4C 1294的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



# ➤ TM4C 1294的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)

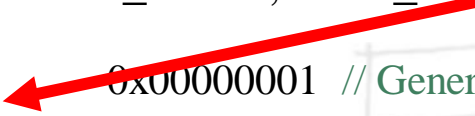
## 7. 使能并配置中断

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)
{
    .....
    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
```





# ➤ TM4C 1294的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、BDown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	





# ➤ TM4C 1294的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)  
{  
    .....  
    HWREG(PWM_GEN_BADDR(ui32Base, ui32Gen) + PWM_O_X_INTEN) |= ui32IntTrig;  
}
```




# ➤ TM4C 1294的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  
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
```



# ➤ TM4C 1294的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);



# ➤ TM4C 1294的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PWM发生器的**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



# ➤ TM4C 1294的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)
{
    //
    // Enable the PWM generator.
    //
    HWREG(PWM_GEN_BADDR(ui32Base, ui32Gen) + PWM_O_X_CTL) |= PWM_X_CTL_ENABLE;
    // PWM_X_CTL_ENABLE 0x01
}
```

**PWMGenEnable**(PWM0\_BASE, PWM\_GEN\_0);



# ➤ TM4C 1294的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PWMnISC寄存器，可以查看中断的来源

写PWMnISC寄存器，可以清除中断



# ➤ TM4C 1294的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

