# 第九章 运动控制外设-PWM 和 QEI

## 1 脉冲宽度调节器 PWM

### 1.1 PWM 简介

脉冲宽度调制PWM是一种利用数字编码来表示模拟信号电平的技术。利用高分辨率计数器产生一个方波，调节方波的占空比相当于调节了模拟信号的电平大小。典型应用包括开关电源和电机控制。

**PWM原理**

由面积等效原则，面积相同，不同形状的信号经过惯性环节，其响应基本相同。如图9-1中的a（矩形）、b（三角）、c（正弦）、d（无穷大）四个信号经过惯性环节后，其输出电流i(t)对不同窄脉冲时的响应波形几乎完全相同。脉冲越窄，响应波形的差异也越小。可以得出结论，面积相同的信号，其结果是等效的。

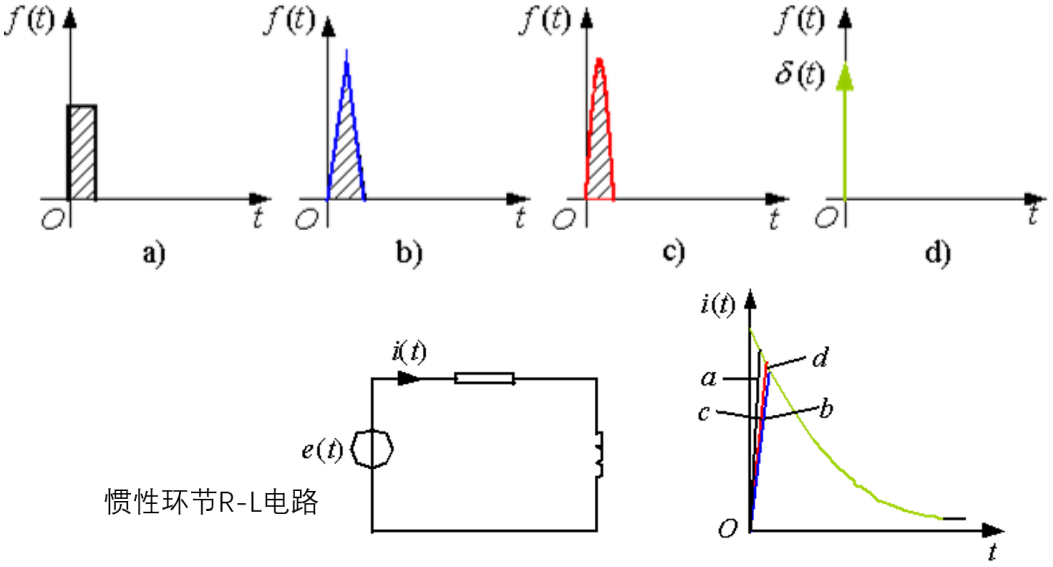


图9-1 面积等效原则

由此，我们可以用一系列宽度不变的脉冲来等效一个直流信号，也可以用用一系列宽度按正弦规律变化的脉冲来代替一个正弦半波。

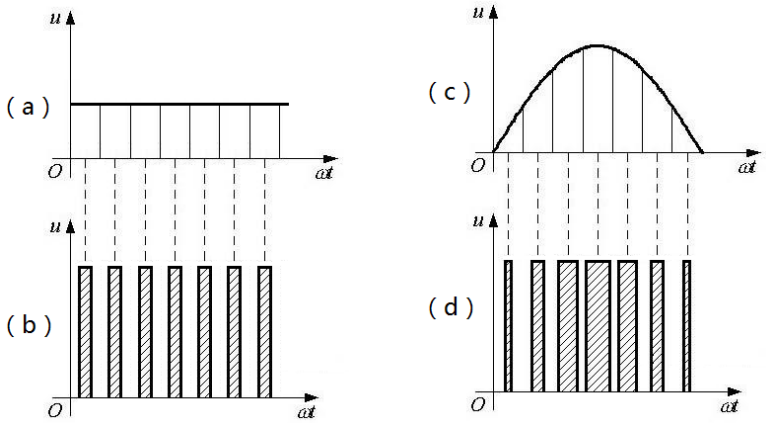


图9-2 PWM信号

**PWM死区**

大功率电机、变频器通常都是由H桥或3相桥来驱动，上、下桥臂不能同时导通。如果没有死区，控制H桥的两路PWM信号会因为开关速度问题，造成1个半桥元件没有完全关断的情况下，另一个半桥元件导通。

死区就是一对互补的PWM信号在电平翻转时插入一个时间间隔，以防止上、下桥臂同时导通。

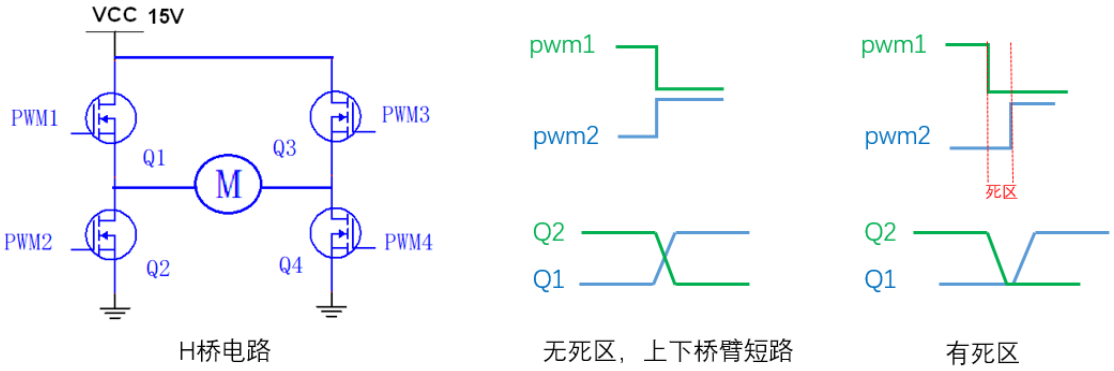


图9-3 PWM死区

## 1.2 功能和特色

TM4C123GH6PM微控制器包含两个PWM模块，每个模块由四个PWM发生器模块和一个控制模块组成，一共可以产生16路PWM输出。

每个PWM发生器模块产生两个PWM信号，这两个信号拥有相同的计数器，我们可以编程对它们单独操作，比如简单电荷泵需要的PWM信号；也可以把它们生成含死区延迟的一对互补信号，比如半H桥驱动需要的PWM信号。三个发生器模块也可以生成完整的六通道PWM信号来控制3相桥式逆变器。

每个PWM发生器模块具有以下特点：

1) 1个故障条件处理输入端，能够快速提供低延迟关闭，防止损坏控制的电机
2) 一个16位计数器，能工作在递减模式或者递增/递减模式，装载值可以同步更新
3) 两个PWM比较器，比较器的值可以同步更新
4) PWM信号发生器，可以产生两个独立的PWMA和PWMB信号，也可以产生一对互相配合的PWM信号
5) 死区发生器，可以产生一对可编程死区延迟的PWM信号，用于驱动半H桥；也可以被禁用，使PWM信号直接输出
6) 可以产生中断和触发ADC采样序列

## 1.3 基本结构

图9-1为PWM模块的结构图,。TM4C123GH6PM控制器包含两个PWM模块，每个模块配有四个PWM发生器（PWM Generator n），可以产生8个独立的PWM信号或者4对带死区延迟插入的PWM信号。每个PWM发生器输出两个内部PWM信号（pwmnA`和pwmnB`）和一个故障信号（pwmnfault），PWM输出控制模块（PWM Output Control Logic）对这些信号处理（输出使能、输出翻转或输出故障设定值），从而输出最终信号PWMn。

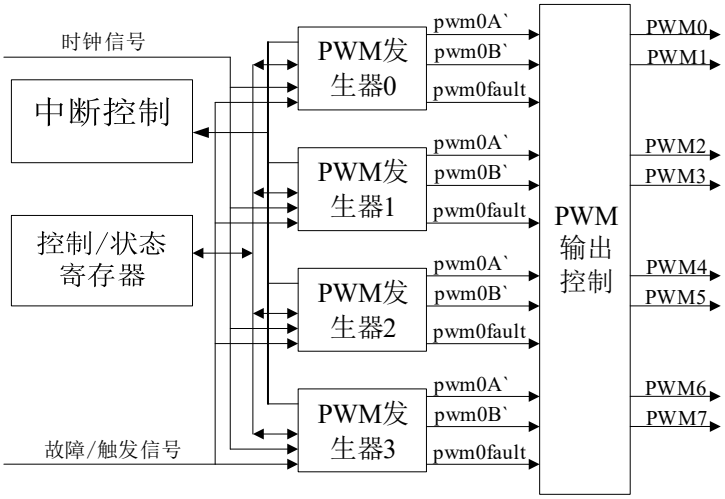


图9-4 PWM模块结构图

在PWM发生器（PWM Generator n）中，定时器（Timer）会产生3个信号：zero（计数值到0）、load（计数值到装载值）和dir（计数方向）。比较器（Comparators）会在定时器的计数值到达比较值A和比较值B时产生cmpA和cmpB信号。中断和触发发生器（Interrupt and Trigger Generator）根据这5个信号产生中断和触发信号。同时信号发生器（Signal Generator）也根据这5个信号产生pwmA和pwmB，最后经过死区发生器（Dead-Band Generator）产生带死区的PWM信号pwmA`和pwmB`。

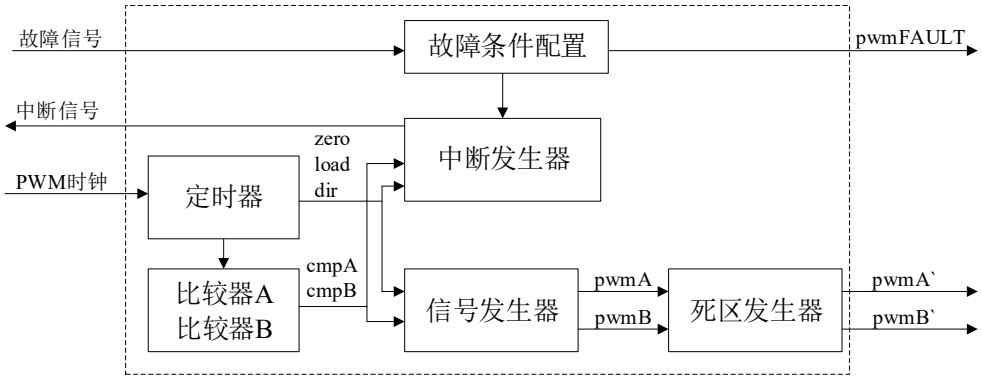


图9-5 PWM发生器（PWM Generator n）结构图

### 1.3.1 信号描述

每个PWM模块有8个PWM输出信号、1个fault输入信号。两个PWM模块的GPIO复用引脚分布如表9-1所示。

9-1 PWM信号

| Pin Name | Pin Number | Pin Mux / Pin Assignment | Pin Type | Buffer Type[a] | Description |
|---|---|---|---|---|---|
| M0FAULT0 | 30<br>53<br>63 | PF2 (4)<br>PD6 (4)<br>PD2 (4) | I | TTL | Motion Control Module 0 PWM Fault 0. |
| M0PWM0 | 1 | PB6 (4) | O | TTL | Motion Control Module 0 PWM 0. This signal is controlled by Module 0 PWM Generator 0. |
| M0PWM1 | 4 | PB7 (4) | O | TTL | Motion Control Module 0 PWM 1. This signal is controlled by Module 0 PWM Generator 0. |
| M0PWM2 | 58 | PB4 (4) | O | TTL | Motion Control Module 0 PWM 2. This signal is controlled by Module 0 PWM Generator 1. |
| M0PWM3 | 57 | PB5 (4) | O | TTL | Motion Control Module 0 PWM 3. This signal is controlled by Module 0 PWM Generator 1. |
| M0PWM4 | 59 | PE4 (4) | O | TTL | Motion Control Module 0 PWM 4. This signal is controlled by Module 0 PWM Generator 2. |
| M0PWM5 | 60 | PE5 (4) | O | TTL | Motion Control Module 0 PWM 5. This signal is controlled by Module 0 PWM Generator 2. |
| M0PWM6 | 16<br>61 | PC4 (4)<br>PD0 (4) | O | TTL | Motion Control Module 0 PWM 6. This signal is controlled by Module 0 PWM Generator 3. |
| M0PWM7 | 15<br>62 | PC5 (4)<br>PD1 (4) | O | TTL | Motion Control Module 0 PWM 7. This signal is controlled by Module 0 PWM Generator 3. |
| M1FAULT0 | 5 | PF4 (5) | I | TTL | Motion Control Module 1 PWM Fault 0. |
| M1PWM0 | 61 | PD0 (5) | O | TTL | Motion Control Module 1 PWM 0. This signal is controlled by Module 1 PWM Generator 0. |
| M1PWM1 | 62 | PD1 (5) | O | TTL | Motion Control Module 1 PWM 1. This signal is controlled by Module 1 PWM Generator 0. |
| M1PWM2 | 23<br>59 | PA6 (5)<br>PE4 (5) | O | TTL | Motion Control Module 1 PWM 2. This signal is controlled by Module 1 PWM Generator 1. |
| M1PWM3 | 24<br>60 | PA7 (5)<br>PE5 (5) | O | TTL | Motion Control Module 1 PWM 3. This signal is controlled by Module 1 PWM Generator 1. |
| M1PWM4 | 28 | PF0 (5) | O | TTL | Motion Control Module 1 PWM 4. This signal is controlled by Module 1 PWM Generator 2. |
| M1PWM5 | 29 | PF1 (5) | O | TTL | Motion Control Module 1 PWM 5. This signal is controlled by Module 1 PWM Generator 2. |
| M1PWM6 | 30 | PF2 (5) | O | TTL | Motion Control Module 1 PWM 6. This signal is controlled by Module 1 PWM Generator 3. |
| M1PWM7 | 31 | PF3 (5) | O | TTL | Motion Control Module 1 PWM 7. This signal is controlled by Module 1 PWM Generator 3. |

## 1.4 工作过程

### 1.4.1 时钟配置

PWM有两个时钟源选择：

1) 系统时钟。
2) 预分频系统时钟，可以2、4、8、16、32和64分频。是否预分频由系统控制(sysctl)模块中的寄存器RCC.USPWMDIV位选择，RCC.PWMDIV位域指定了预分频数。

### 1.4.2 PWM 定时器和比较器

每个PWM发生器中的定时器可以向下计数，也可以向上/向下计数。在向下计数模式中，定时器从装载值计数直到为零，然后回到装载值，并继续向下计数。在向上/向下计数模式中，定时器从零计数至装载值，然后向下计数到零，再向上计数至装载值，以此进行。一般来说，向下计数模式用于产生左对齐或右对齐的PWM信号，而向上/向下计数模式用于产生中间对齐的PWM信号。

定时器输出三个信号，这三个信号会在PWM生成过程中使用：方向信号“dir”，当计数器的值递减时，该信号是低电平；计数器的值递增时，该信号为高电平。所以在向下计数模式中，dir一直为低电平；在向上/向下计数模式中，dir会在低电平和高电平之间轮流变

换。

当计数器计数到零时产生会产生信号“zero”；当计数器计数到装载值时会产生信号“load”。注意，在向下计数模式中，计数到零产生的 “zero”之后会紧接着由装载值产生的“load”。

每个PWM发生器都有两个比较器，用来监视计数器的值。当计数器的值和比较器A相匹配时，会产生“cmpA”信号；当计数器的值和比较器B相匹配时，产生“cmpB”信号。“cmpA”和“cmpB”信号按照计数方向限定，又有了Aup、Adown和Bup、Bdown。

“zero”、 “load”、 “cmpA”和“cmpB”信号都是单时钟周期宽度的高电平脉冲。

图9-3显示了在向下计数模式中，计数器的运行状态以及和这些信号之间的关系。图9-4显示了在向上/向下计数模式中，计数器的运行状态以及和这些信号之间的关系。

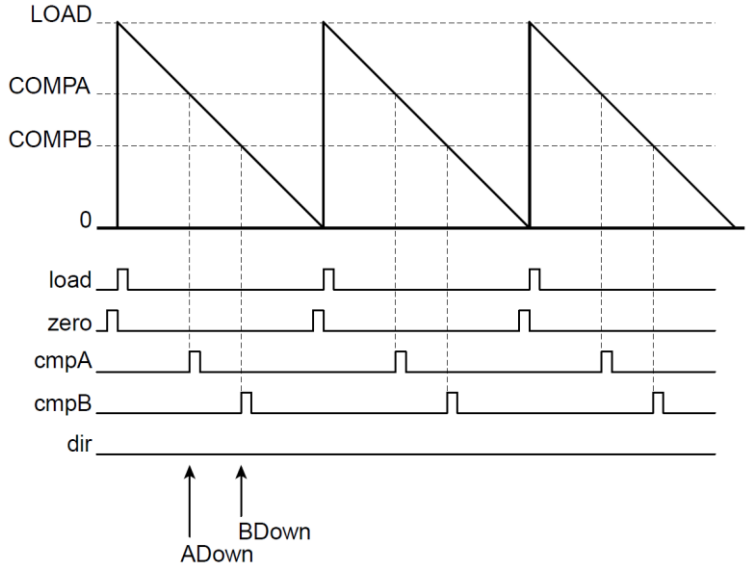


图9-6 PWM向下计数模式

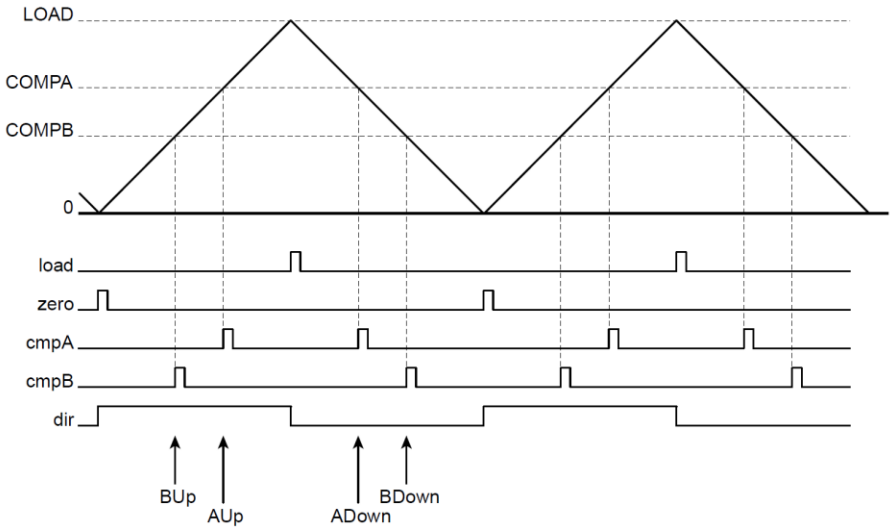


图9-7 PWM向上/向下计数模式

### 1.4.3 PWM 信号发生器

每个PWM发生器接受load，zero，cmpA和cmpB脉冲（由dir信号修饰），用于产生两个内部PWM信号，pwmA和pwmB。

在向下计数模式中，这些脉冲会产生四个事件：zero、load、match A down、match B down。在向上/向下模式中，这些脉冲会产生六个事件：zero，load、match A up、match A

down和match B up、match B down。当match A和match B事件与zero和load事件同时发生时，前者会被忽略。如果match A和match B事件同时发生时，pwmA信号只会基于match A事件产生，而pwmB信号只会基于match B事件产生。

每个事件对PWM信号的影响都是可编程的：可以被忽略（忽略事件），反转信号，驱动信号为低电平，驱动信号为高电平。这些操作可以用来产生不同位置和占空比的一对PWM信号。图9-5显示了使用向上/向下计数模式生成一对中间对齐，有不同占空比的PWM信号。

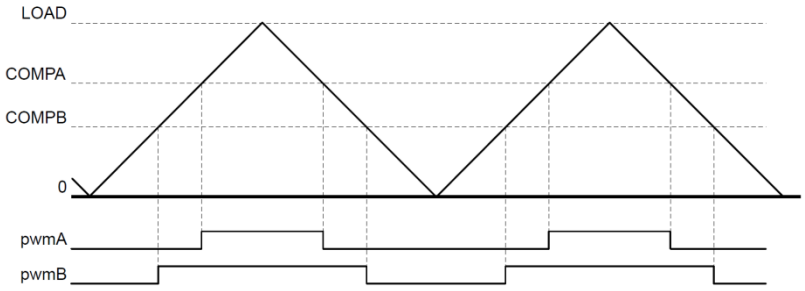


图9-8 向上/向下模式中PWM生成示例

在这个例子中，pwmA发生器设置为在match A up事件时驱动高电平，在match A down事件时驱动低电平，并忽略其他四种事件。pwmB发生器设置为在match B up事件时驱动高电平，在match B down事件时驱动低电平，并忽略其他四种事件。改变比较器A的值会改变pwmA信号的占空比，而改变比较器B的值会改变pwmB信号的占空比。

### 1.4.4 同步方式

每个PWM模块都提供了4个PWM发生器，这些发生器可以独立使用，也可以同步使用。

同步就是PWM发生器配合其他PWM发生器使用统一的时间基准。如果多个PWM发生器配置成相同的计数器装载值，就可以使用同步以保证他们有相同的计数值。有了这个功能，就可以产生出两个以上的MnPWMn信号，这些信号的边沿之间有着已知的关系，因为计数器总是具有相同的值。

通过写PWM时间基准同步寄存器（PWMSYNC）中相应的SYNCn位可以将PWM发生器的计数值复位为0。多路PWM通过一个访问操作对所有相关的的SYNCn位置位，实现一起同步。例如，将PWMSYNC寄存器中的SYNC0和SYNC1位同时置1可以使PWM发生器0和PWM发生器1一起复位。

寄存器被写入后，有时并不会马上起作用，有以下三种情况：

1) 立即。写操作立刻产生作用，硬件立即作出反应。
2) 本地同步。写操作不影响执行逻辑直到本PWM周期结束（计数器的值为零）。在这种情况下，写操作的效果延迟了，但提供了一个有保障的行为，以防止输出过短或过长的PWM脉冲。
3) 全局同步。写操作不影响执行逻辑直到满足下面两个连续事件：
   （1）发生器功能的更新模式在PWMnCTL寄存器中被设置为全局同步，
   （2）计数器值在PWM周期结束时为零。在这种情况下，写操作的影响会被推迟直到所有PWM周期结束。

这种模式可以同时对多个PWM发生器中的多个寄存器进行更新，而不会在更新过程中出现意外的情况；每个PWM发生器都一直在旧值运行，直到在某一时刻它们能够同时从新值开始运行。

以下寄存器默认为本地同步，也可设置为全局同步：

- 发生器寄存器：PWMnLOAD，PWMnCMPA，PWMnCMPB

以下寄存器默认为立即更新，但也可选择为本地同步或全局同步：

- 模块级寄存器：PWMENABLE
- 操作寄存器：PWMnGENA，PWMnGENB，PWMnDBCTL，PWMnDBRISE，PWMnDBFALL。

### 1.4.5 死区发生器

每个PWM发生器产生的pwmA和pwmB信号都会传递到死区发生器。如果死区发生器被禁用，那么PWM信号会直接通过死区发生器，毫无变化的变成pwmA`和pwmB`信号。如果死区发生器被启用，那么会丢弃pwmB信号并基于pwmA信号产生两个PWM信号。pwmA`是由pwmA信号的上升沿上增加一个延迟产生的；而pwmB`是由pwmA信号反转，并在pwmA信号的下降沿增加一个延迟产生。图9-6显示了pwmA信号对pwmA`和pwmB`信号的影响。

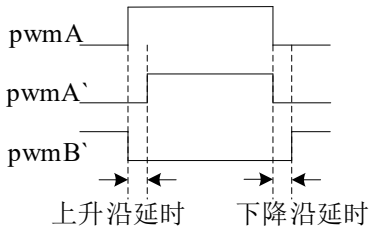


图9-9 PWM死区发生器

pwmA`和pwmB`信号是一对基本互补的信号，除了在翻转时的一段过渡时间内，两个信号都是低电平。因此，这些信号适合驱动半H桥，死区延时可以防止直通电流损坏电子设备。

### 1.4.6 故障状态

故障状态是指控制器必须关闭PWM输出，并将输出信号保持为一个安全值的状态。许多电机（电源）驱动芯片都带有fault引脚，以通知MCU电机（电源）的过流、欠压等故障状态。

每个PWM发生器都可以用下面的输入产生一个故障状态，包括：

- 外部故障信号输入（MnFAULTn引脚）
- ADC模块的数字比较器触发
- 调试器（debug）引发的控制器中止

**1) 故障状态的产生**

寄存器**PWMnCTL**指定了故障状态由外部故障信号或者由数字比较器触发。外部故障信号在何种电平（1还是0）下产生故障状态由**PWMnFLTSEN**寄存器配置。

故障状态能使能一个计数器，这个计数器可以用来延长外部信号产生的故障状态的时间，以保证它的持续时间是最小长度。最小故障时间数在**PWMnMINFLTPER**寄存器中指定。

**2) 故障状态下的输出值**

在故障状态期间，PWM输出信号MnPWMn必须输出安全值才能保证外部设备受到安全控制。**PWMFAULT**寄存器指定了在故障期间，所产生的信号是保持不变还是变成**PWMFAULTVAL**寄存器中指定的值。

### 1.4.7 PWM 输出控制

PWM输出控制模块可以关闭PWM信号输出，也可以使PWM信号反转输出。

PWM输出使能寄存器（**PWNENABLE**）寄存器决定是否将pwmA`和pwmB`信号连接到输出引脚MnPWMn上，如果不使能，则引脚输出为0。例如：用对**PWNENABLE的**写操作来执行无刷直流电机的换流，这样就不需要修改PWM发生器，而PWM发生器可以一直由反馈回路控制。

PWM输出反转寄存器（**PWMINVERT**）可以设置任一MnPWMn信号进行反转，使其成为低电平有效而不是默认的高电平有效。反转对故障状态下的**PWMFAULTVAL**寄存器也有效。换句话说，如果在故障状态下，**PWMFAULTVAL**中置1，而**PWMINVERT**寄存器也设置了，那么MnPWMn信号上输出是0而不是**PWMFAULTVAL**寄存器中指定的1。

### 1.4.8 中断/ADC 触发

每个PWM发生器都接受4个（或6个）计数器事件，来产生一个中断或ADC触发。这些事件的任一事件或一组事件都可以作为中断源。PWM故障状态也可以触发中断。

注意，中断和ADC触发器基于原始事件，不考虑死区发生器造成的PWM信号边沿上的延迟。

## 1.5 寄存器描述

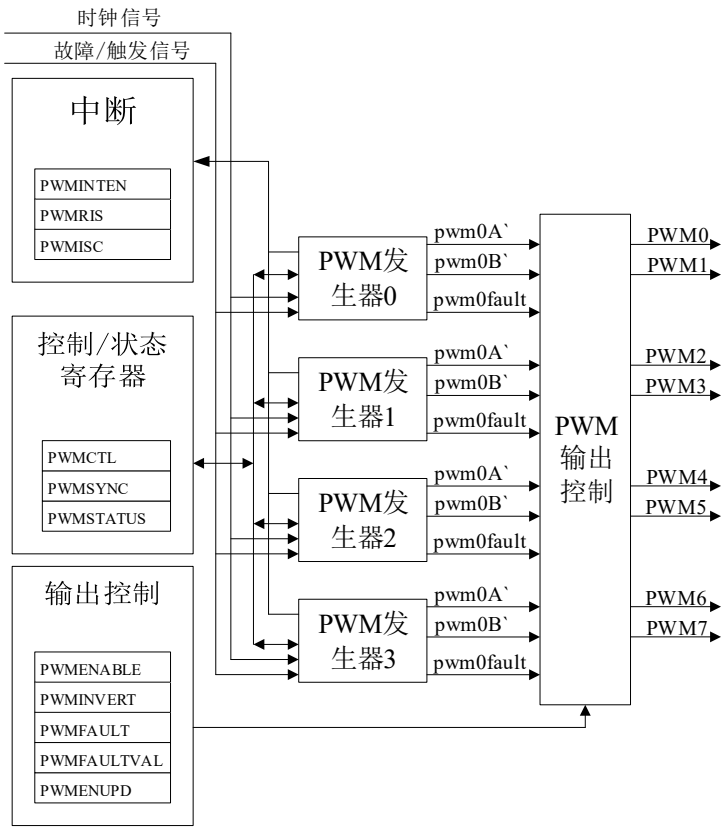


图9-10 PWM模块寄存器

### 1.5.1 WM 模块控制/状态寄存器

PWMCTL：全局同步更新
PWMSYNC：各发生器同步
PWMSTATUS：故障状态

### 1.5.2 PWM 输出控制

PWMENABLE：PWM引脚输出使能
PWMINVERT：输出信号反转
PWMFAULT：故障时信号输出
PWMFAULTVAL：故障时信号输出值
PWMENUPD：PWMENABLE寄存器更新方式

### 1.5.3 PWM 模块中断控制

### 1.5.4 PWM 发生器

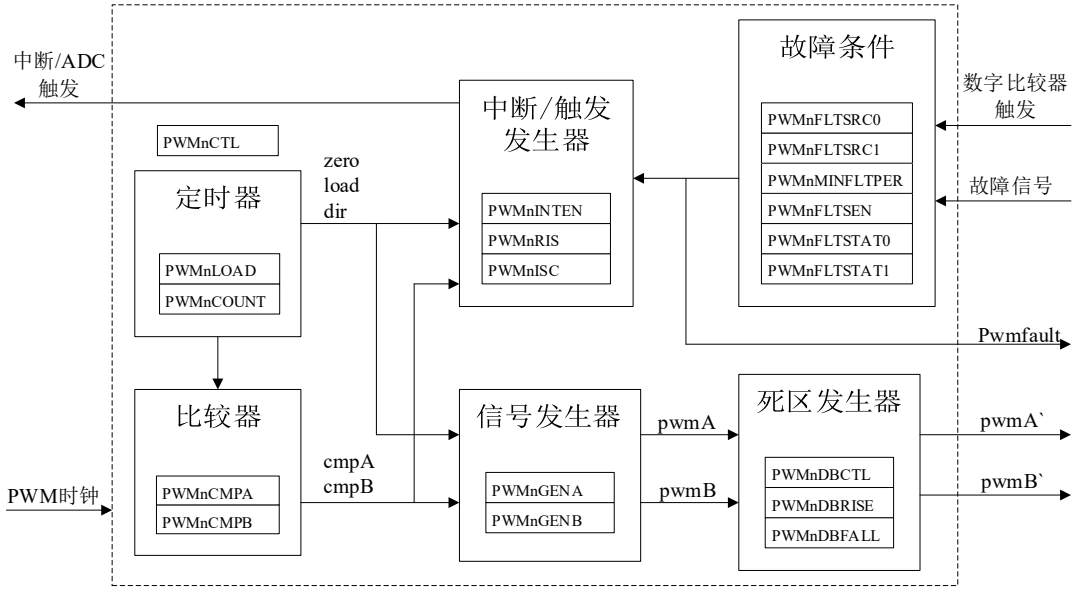


图9-11 PWM发生器寄存器

PWMnCTL：PWM发生器使能、向下/上下计数、设置各寄存器的更新方式

定时器和比较器（16 位）
PWMnLOAD：定时器重载值
PWMnCOUNT：定时器计数值
PWMnCMPA：比较器A
PWMnCMPA：比较器B

信号发生器
PWMnGENA：配置6个事件对PWMA信号的影响
PWMnGENB：配置6个事件对PWMB信号的影响

死区发生器
PWMnDBCTL: 死区使能
PWMnDBRISE：死区上升沿延时时钟个数
PWMnDBFALL：死区下降沿延时时钟个数

PWM发生器中断控制
PWMINTEN：ADC触发使能和中断屏蔽
PWMRIS：原始状态
PWMISC：中断屏蔽状态

### 1.5.5 初始化配置

下面的例子演示了如何初始化设置PWM发生器0，频率为25KHz，MnPWM0引脚信号占空比为25%，MnPWM1引脚信号占空比为75%。此例假定系统时钟为20MHz。

1) 通过写0x00100000到系统控制模块的RCGC0寄存器使能PWM时钟。
2) 通过系统控制模块中RCGC2寄存器使能相应GPIO模块的时钟。
3) 在GPIO模块中，使用GPIOAFSEL寄存器使能相应引脚的复用功能。
4) 配置GPIOPCTL寄存器中的PWCn域将PWM信号分配到合适的引脚上。
5) 配置系统控制模块的运行模式时钟配置（**RCC**）寄存器，使用PWM分频（**USEPWMDIV**）并设置分频器（**PWMDIV**）的除数因子为2（000）。
6) 配置PWM发生器为向下计数，参数立即更新模式。
   - 写0x00000000到**PWM0CTL**寄存器。
   - 写0x0000008C到**PWM0GENA**寄存器。
   - 写0x0000080C到**PWM0GENB**寄存器。
7) 设置周期时间。对于一个25KHz的频率，周期=1/25000，或40微秒。PWM的时钟源为10MHz；为系统时钟频率的一半。因此，每PWM周期有400个时钟周期。使用这个值设置**PWM0LOAD**寄存器。在向下计数模式，设置**PWM0LOAD**寄存器中**LOAD**域为需要的周期长度减1。
   - 写0x0000018F到**PWM0LOAD**寄存器。
8) 设置MnPWM0引脚的脉冲宽度为25％占空比。
   - 写0x0000012B到**PWM0CMPA**寄存器。
9) 设置MnPWM1引脚的脉冲宽度为75％占空比。
   - 写0x00000063到**PWM0CMPB**寄存器。
10) 启动PWM发生器0的定时器。
   - 写0x00000001到**PWM0CTL**寄存器。
11) 使能PWM输出。
   - 写0x00000003到**PWMENABLE**寄存器。

## 1.6 主要的库函数

### 1.6.1 函数 SysCtlPWMClockSet

原形：void SysCtlPWMClockSet(uint32_t ui32Config)
描述：该函数用于设置PWM时钟的预分频数；
参数：**ui32Config**是PWM时钟的预分频数，可以为1、2、4、8、16、32、64分频，例如**SYSCTL_PWMDIV_64**。

### 1.6.2 函数 PWMGenConfigure

原形：void PWMGenConfigure(uint32_t ui32Base, uint32_t ui32Gen, uint32_t ui32Config)
描述：该函数用于设置PWM发生器的运行模式。计数模式(向下或递增/递减)、同步模式和调试状态都在此函数中配置，配置后发生器处于禁用状态。

此函数还设定了pwm信号的产生方式：

如果配置为向下计数，则pwmA发生器设置为在load事件时驱动高电平，在match A

down事件时驱动为低电平。pwmB信号在load事件时驱动为高电平，在match B down事件时驱动为低电平。

如果配置为递增/递减计数，则pwmA发生器设置为在match A up事件时驱动高电平，在match A down事件时驱动为低电平。pwmB信号在match B up事件时驱动为高电平，在match B down事件时驱动为低电平。

参数：**ui32Base**为PWM模块基地址；

**ui32Gen**是需要配置的PWM发生器，例如**PWM_GEN_0**；

**ui32Config**是PWM发生器的配置信息。

### 1.6.3 函数 PWMGenPeriodSet

原形：void PWMGenPeriodSet(uint32_t ui32Base, uint32_t ui32Gen, uint32_t ui32Period)

描述：该函数指定PWM输出的脉冲周期；

参数：**ui32Base**为PWM模块的基地址；

**ui32Gen**为需要修改的PWM发生器。例如**PWM_GEN_0**；

**ui32Period**指定PWM发生器的输出周期，由PWM clock ticks的数目确定。

### 1.6.4 函数 PWMPulseWidthSet

原形：void PWMPulseWidthSet(uint32_tui32Base, uint32_t ui32PWMOut, uint32_t ui32Width)

描述：该函数指定PWM输出的脉冲宽度；

参数：**ui32Base**为PWM模块的基地址

**ui32PWMOut**为需要修改的PWM输出，为从**PWM_OUT_0**到**PWM_OUT_7**的8个输出中的一个；

**ui32Width**指定了脉冲正极性部分的宽度，该宽度由PWM clock ticks的数目确定。

### 1.6.5 函数 PWMDeadBandEnable

原形：void PWMDeadBandEnable(uint32_t ui32Base, uint32_t ui32Gen, uint16_t ui16Rise, uint16_t ui16Fall)

描述：该函数使能死区输出，并配置死区延时；

参数：**ui32Base**为PWM模块的基地址

**ui32Gen**为需要修改的PWM发生器。例如**PWM_GEN_0**；

**ui16Rise**指定了死区上升沿延时，该延时由PWM clock ticks的数目确定；

**ui16Fall**指定了死区下降沿延时，该延时由PWM clock ticks的数目确定。

### 1.6.6 函数 PWMOutputState

原形：void PWMOutputState(uint32_t ui32Base, uint32_t ui32PWMOutBits, bool bEnable)

描述：此功能启用或禁用选定的PWM输出；

参数：**ui32Base**为PWM模块的基地址；

**ui32PWMOutBits**为需要修改的PWM输出，该参数为从**PWM_OUT_0**到**PWM_OUT_7**中的8个输出的逻辑与的任意组合；

**bEnable**指定了该信号是否启用，如果**bEnable**为true，则启用所选的PWM输出。

### 1.6.7 函数 PWMGenEnable

原形：voidPWMGenEnable(uint32_t ui32Base,uint32_t ui32Gen)

描述：该函数允许PWM时钟驱动指定的发生器模块的定时器/计数器

参数：**ui32Base**为PWM模块的基地址；

**ui32Gen**为需要修改的PWM发生器。例如**PWM_GEN_0**；

### 1.6.8 函数 PWMOutputInvert

原形：voidPWMOutputInvert(uint32_t ui32Base, uint32_t ui32 PWMOutBits, bool bInvert)

描述:该函数用于选择PWM输出的反转模式；

参数：**ui32Base**为PWM模块的基地址；

**ui32PWMOutBits**为需要修改的PWM输出，该参数为从**PWM_OUT_0**到**PWM_OUT_7**中的8个输出的逻辑与的任意组合；

**bInvert**确定信号反转还是直接通过，如果bInvert为true，指定的PWM输出信号进行反转或变成低电平有效。

## 1.7 例程

```
#include <stdbool.h>
#include <stdint.h>
#include "inc/hw_memmap.h"
#include "driverlib/gpio.h"
#include "driverlib/interrupt.h"
#include "driverlib/pin_map.h"
#include "driverlib/pwm.h"
#include "driverlib/sysctl.h"

int main(void)
{
    SysCtlClockSet(SYSCTL_SYSDIV_1 | SYSCTL_USE_OSC | SYSCTL_OSC_MAIN |
                   SYSCTL_XTAL_16MHZ);
// 配置PWM时钟为64分频，使能PWM0模块
    SysCtlPWMClockSet(SYSCTL_PWMDIV_64);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM0);
// M0PWM7引脚复用设置
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC);
    GPIOPinConfigure(GPIO_PC5_M0PWM7);
    GPIOPinTypePWM(GPIO_PORTC_BASE, GPIO_PIN_5);
// 配置PWM0发生器3为向上/向下计数，不同步
    PWMGenConfigure(PWM0_BASE, PWM_GEN_3, PWM_GEN_MODE_UP_DOWN |
PWM_GEN_MODE_NO_SYNC);
// 配置PWM周期和宽度，使能输出
    PWMGenPeriodSet(PWM0_BASE, PWM_GEN_3, SysCtlClockGet()/400);
    PWMPulseWidthSet(PWM0_BASE, PWM_OUT_7, SysCtlClockGet()/800);
    PWMOutputState(PWM0_BASE, PWM_OUT_7_BIT, true);
// 使能PWM0发生器3
    PWMGenEnable(PWM0_BASE, PWM_GEN_3);

    while(1)
    {
```

```
    }
}
```

# 2　正交编码器 QEI

## 2.1　正交编码器简介

正交编码器又叫增量编码器，可以用来测量旋转系统的旋转运动，广泛应用于各种电机的转速和位置测量。图9-5显示了一种编码器的基本组成，包括一个发光二极管（LED）、一个码盘，以及码盘背面的一个光传感器。这个码盘安置在旋转轴上，上面按一定编码形式排列着不透光和透光的扇形区域。当码盘转动时，不透光扇区能够遮挡光线，而透光扇区则允许光线透过。这样就产生了方波脉冲。发光二极管和光传感器也可以安装在同一面，利用光的反射来产生脉冲信号。扇区的数量决定了测量的精度，假如一个编码器有100个扇区，则其可以提供3.6度的精度。

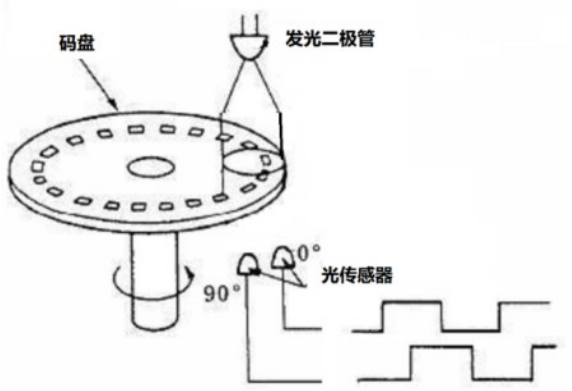


图9-12　正交编码器A和B的输出信号

仅有一路脉冲输出的编码器可以测量旋转的速度，但不能确定旋转的方向。如果使用两路脉冲输出，使它们的相位差为90°（如图9-6所示），那么通过两个脉冲的相位关系就可以确定旋转的方向。如图9-7，如果通道A的信号相位超前90°，码盘就以顺时针旋转。如果通道B的信号相位超前90°，那么码盘就是以逆时针旋转。因此，通过监控脉冲的数目和信号A、B之间的相对相位信息，就可以同时获得旋转的位置和方向信息。

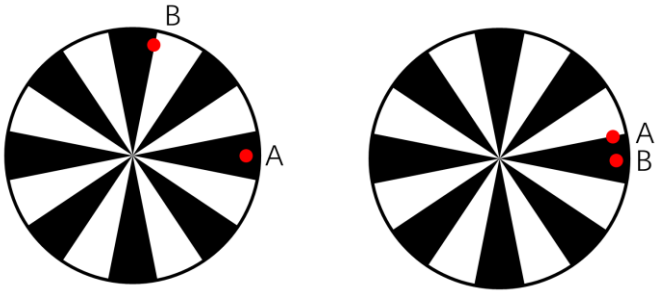


图9-13　正交编码器A和B的安装


（a）顺时针旋转信号　　（b）逆时针旋转信号

图9-14　正交编码器A和B的输出信号

除此之外，有些正交编码器还包含被称为索引（index）信号的第三个输出通道。这个通道每旋转一圈输出一个单脉冲，可以使用这个单脉冲作为参考位置信号，来精确测量转盘的当前位置。

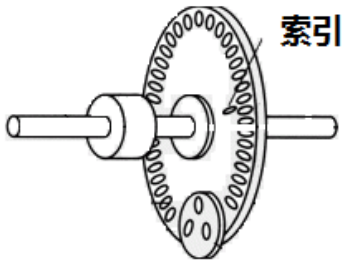


图9-15 正交编码器的索引信号

tm4c123gh6pm微控制器包括两个QEI模块，每个模块包括3个引脚：PhAn、PhBn和索引信号IDX。QEI模块具有以下特点：

■位置积分器（Position Integrator）跟踪编码器的位置

■可以编程设置滤波器去除输入中的噪声

■可以通过内置的定时器计算转速

■QEI输入信号的频率可高达1/4的处理器频率（例如：12.5MHz，50MHz系统时钟）

## 2.2 结构图

TM4C123GH6PM的一个QEI模块的内部框图如图14-8所示，其由正交编码器、速度预分频、位置积分器、速度测量（速度定时器和速度累加器）组成。两个输入的正交信号PhA和PhB经过正交编码，生成脉冲clk信号和方向dir信号；clk、dir信号和IDX信号送到位置积分器测量位置；clk信号经过速度预分频，生成clkdiv信号，送到速度累加器，和速度定时器一起，测量转速。

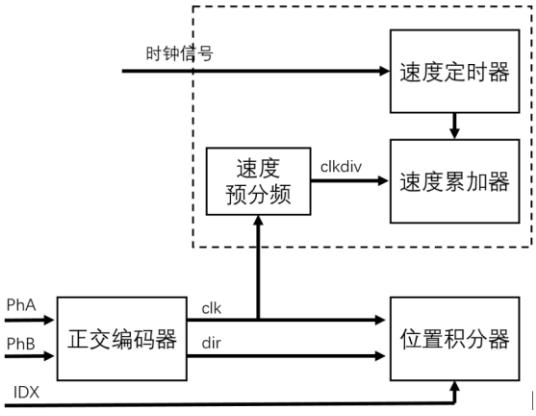


图9-16 QEI框图

## 2.3 信号描述

下表列出QEI模块的外部信号，2个QEI模块共6个输入信号，并列出了其复用引脚。

表9-2 QEI信号（64LQFP）

| Pin Name | Pin Number | Pin Mux / Pin Assignment | Pin Type | Buffer Type[a] | Description |
|---|---|---|---|---|---|
| IDX0 | 5<br>64 | PF4 (6)<br>PD3 (6) | I | TTL | QEI module 0 index. |
| IDX1 | 16 | PC4 (6) | I | TTL | QEI module 1 index. |
| PhA0 | 28<br>53 | PF0 (6)<br>PD6 (6) | I | TTL | QEI module 0 phase A. |
| PhA1 | 15 | PC5 (6) | I | TTL | QEI module 1 phase A. |
| PhB0 | 10<br>29 | PD7 (6)<br>PF1 (6) | I | TTL | QEI module 0 phase B. |
| PhB1 | 14 | PC6 (6) | I | TTL | QEI module 1 phase B. |

## 2.4 功能描述

### 2.4.1 QEI 输入信号逻辑

外部输入脉冲信号PhAn和PhBn先通过反相和交换逻辑（如图9-10）后得到的内部信号PhA、PhB，即可以将PhAn和PhBn信号取反，或者将PhAn和PhBn信号互相交换来生成PhA和PhB。反相和交换逻辑可以用来纠正硬件电路中的错误接线。

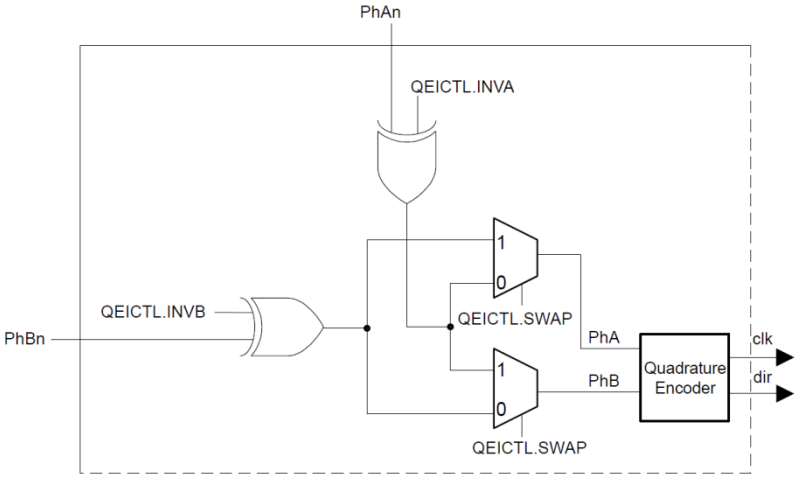


图9-17 QEI输入信号逻辑

### 2.4.2 正交编码

正交编码器会根据输入的PhA和PhB信号输出计数信号clk和旋转方向信号dir（如图9-11）。

计数信号clk会在PhA和PhB信号的边沿产生，每个边沿（上升沿或下降沿）会产生一个clk脉冲。QEI可以在PhA和PhB两个信号的边沿都产生clk信号，也可以只在PhA的边沿产生clk信号。

方向信号dir由PhA和PhB信号的相位关系决定：当PhA超前PhB 90°相位时，dir为低电平，表示一个旋转方向；当PhA滞后 PhB 90°相位时，dir为高电平，表示另一个旋转方向。

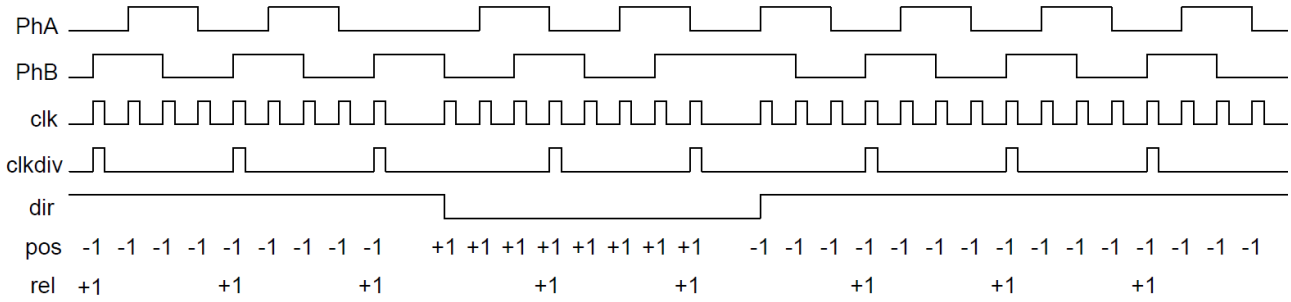


图9-18 QEI的操作过程

如果外部输入的信号是已经做过正交编码处理的，即输入的就是clk和dir信号，QEI也可以将PhA和PhB直接连接到clk和dir上，这样正交编码器实际上就没有工作了。

### 2.4.3 位置测量

位置的计算是通过对计数信号clk进行积分得到的。如图9-11中的pos信号，当dir信号为低电平时，每来1个clk信号，pos加1，当dir信号为高电平时，每来1个clk信号，pos减1。pos的值保存在QEIPOS寄存器中。

这样计算的结果只是转盘总的位移，要想得到转盘当前在圆周中的位置还需要有一个起始点作为参考。QEI模块提供了两种方法。

第一种方法是使用QEIMAXPOS寄存器，当QEIPOS的值到达QEIMAXPOS中设定的值时，QEIPOS复位。我们可以将QEIMAXPOS的值设为码盘每旋转一圈产生的clk值减1，这样最后得到的QEIPOS值就是相对于初始点在一圈内的位置。

另一种方法是使用索引（IDX）信号。IDX信号码盘每圈产生一次，检测到IDX信号时，QEIPOS复位。即每圈QEIPOS**都**会在码盘转到IDX信号的位置时归零，重新计数，这样重新计数的值就可以表明码盘当前在一圈内的位置。

### 2.4.4 速度测量

#### 2.4.4.1 速度预分频

速度预分频器（Velocity Predivider）可以将clk信号做一个预分频处理后再交给后面的速度计算单元。例如图9-11中，每4个clk信号转换为1个clkdiv，即做了一个4分频。预分频器使速度计算单元可以处理的clk信号频率更高。

#### 2.4.4.2 速度定时器和速度累加器

测量速度需要计算单位时间内的脉冲个数。模块带有1个速度定时器（Velocity Timer），用来确定clkdiv信号计数的时间。QEICOUNT寄存器对clkdiv信号计数，速度定时器会周期性定时，每次定时时间到达时都会将QEICOUNT里的值保存到QEISPEED寄存器中，然后QEICOUNT归零，重新计数。所以，QEICOUNT寄存器里保存的是clkdiv计数的瞬时值，QEISPEED寄存器里保存的是上一次定时时间内clkdiv信号的总个数。

$clock$是速度定时器的时钟频率，$LOAD$是速度定时器的重载值，则定时时长为：

$$T = \frac{LOAD}{clock}$$

$ppr$是每转物理编码器的脉冲数；$edges$是2或4（基于clk信号对PhA和PhB都产生，还是只对其中1个产生），$2^{VELDIV}$是预分频数，则每转的clkdiv数为：

$$N = \frac{ppr * edges}{2^{VELDIV}}$$

*SPEED*是定时器每次更新时QEISPEED寄存器中记录的clkdiv个数，转速的单位是rpm（转/分钟），则转速rpm为：

$$\text{rpm} = 60 * SPEED \div N \div T$$
$$= (clock * (2^{VELDIV}) * SPEED * 60) \div (LOAD * ppr * edges)$$

例如，使用一个1024脉冲每转的正交编码器连接到电机。当速度预分频器没有预分频（$VELDIV$=0）和对PhA和PhB的边缘都产生clk时，则每转会产生4096个clkdiv。如果电机转速为600rpm（10转/秒），则每秒产生40960个clkdiv。定时器频率为1MHz，重载值为250000（¼秒），则定时器每次更新将计数10240个clkdiv，即*SPEED=10240*，使用上面的公式：

$$\text{rpm} = (1000000 * 1 * SPEED * 60) \div (250000 * 1024 * 4) = 600rpm$$

要注意的是：在编写程序时，上面公式中前面的乘法部分很容易造成变量溢出，例如代码：

```
uint32_t speed, rpm;
speed=10240;
rpm=(1000000*1*speed*60)/(250000*1024*4);
```

得到的结果rpm为0，这是因为计算过程中的中间结果为：

$$1000000 * 1 * 10240 * 60 = 614400000000$$

很明显这个数超出了32位变量的大小范围，会出现溢出，最终导致计算结果错误。

解决的办法是先将分子分母中的参数相约，再输入等式。例如：

```
rpm=(4*1*speed*60)/(1024*4);
```

此时运算结果为rpm=600，结果正确。

### 2.4.5 数字噪声滤波

QEI的数字滤波器被用来滤除两路正交信号和索引信号中的噪声，特别是电机系统中常见的持续时间很短的脉冲尖峰。

开启数字滤波器后，只有输入信号电平在3个滤波器时钟边沿都保持不变，滤波后的输出信号才会改变为相应的电平。这样，持续时间少于两个周期的脉冲将会被滤除。

滤波器时钟由系统时钟分频得到，在QEICTL.FILTCNT中设置分频数，滤波器时钟频率越低，滤波器能通过的信号频率越低。

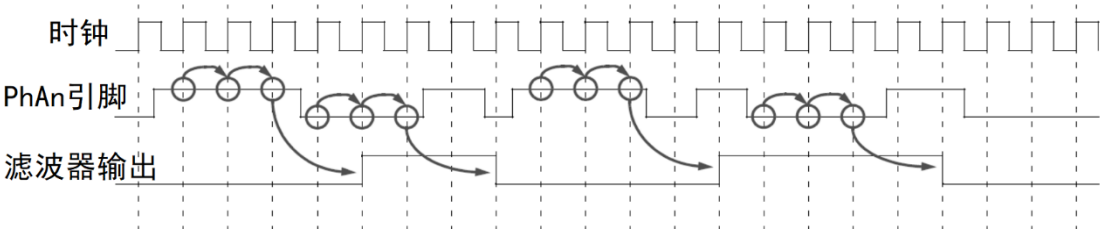


图9-19 数字噪声滤波

### 2.4.6 初始化配置

下面的示例演示如何配置正交编码器模块以读取绝对位置：

1) 外设使能

配置系统控制模块的 RCGCQEI 寄存器使能 QEI 时钟。

2) 引脚复用分配

通过系统控制模块中 RCGCGPIO 寄存器使能相应 GPIO 模块的时钟。

在 GPIO 模块中，使用 GPIOAFSEL 寄存器使能相应引脚的复用功能。

配置 GPIOPCTL 寄存器中的 PWCn 域将 QEI 信号分配到合适的引脚上。

3) 配置工作方式

此例中，配置一个双边沿捕获，测量一圆周内的绝对位置。假设每圈 PhA 的脉冲数为 1000，则双边沿捕捉每转会有 4000 个 clk 脉冲。

- 写 0x00000018 到 QEICTL 寄存器；
- 写 0x00000f9f（4000-1）到 QEIMAXPOS 寄存器；

4) 读取位置

置位 QEICTL 寄存器的第 0 位使能正交编码器；

延时等待，直到获得绝对位置；

读取位置寄存器（QEIPOS)的值获得编码器的位置。

## 2.5 例程

```
#include <stdbool.h>
#include <stdint.h>
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "inc/hw_gpio.h"
#include "driverlib/gpio.h"
#include "driverlib/pin_map.h"
#include "driverlib/qei.h"
#include "driverlib/sysctl.h"

int main(void)
{
    uint32_t NUM;
    SysCtlClockSet(SYSCTL_SYSDIV_5 | SYSCTL_USE_PLL | SYSCTL_XTAL_16MHZ |
SYSCTL_OSC_MAIN);
// 使能QEI0外设,GPIOD外设
    SysCtlPeripheralEnable(SYSCTL_PERIPH_QEI0);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);
// PD7解锁
    HWREG(GPIO_PORTD_BASE + GPIO_O_LOCK) = GPIO_LOCK_KEY;
    HWREG(GPIO_PORTD_BASE + GPIO_O_CR) |= 0x80;
    HWREG(GPIO_PORTD_BASE + GPIO_O_LOCK) = 0;
// 引脚复用设置
    GPIOPinConfigure(GPIO_PD6_PHA0);
    GPIOPinConfigure(GPIO_PD7_PHB0);
    GPIOPinTypeQEI(GPIO_PORTD_BASE,  GPIO_PIN_6 | GPIO_PIN_7);
// 配置QEI,双边沿捕获,配置IDX复位,正交信号输入,最大位置计数为3999
    QEIConfigure(QEI0_BASE, (QEI_CONFIG_CAPTURE_A_B | QEI_CONFIG_NO_RESET |
QEI_CONFIG_QUADRATURE ), 3999);
// 使能QEI0
    QEIEnable(QEI0_BASE);
```

```
// 延时
    SysCtlDelay(SysCtlClockGet()/12);

// 读取位置
    NUM = QEIPositionGet(QEI0_BASE);
    while(1)
    {
    }
}
```

当输入两个频率为 500Hz 的正交脉冲时，0.25 秒后读到的脉冲数为 500。

# 3 习题

## 3.1 预习

1) 描述定时器 PWM 的基本结构和原理。
2) QEI 的主要用途是什么？Tiva QEI 可以达到的最高工作频率是多少？
3) 完成德研 TIVA-demo\PWM\SPWM 例程，将产生的 PWM 波形和滤波后的 sin 波形（德研板上已经滤波，用示波器在德研板上的 SPWM 探针观测）拍照后贴到报告中。

## 3.2 课堂 1

1) 直流电机的控制与测速。在艾研板上用 PWM 控制 1 个直流空心杯电机的转速（开环），并测量转速（用 QEI）（可在 debug 中观察转速结果）。发挥：转速的闭环控制。
   注意：
   a) 母板上 J1 和 J2 的连接（见《2015 升级版 5529MSEK 实验手册》第一章）
   b) 电机板上 S2、S4 开关的设置
   c) PB4 输出 PWM 波、PB7 和 PB0 要输出高电平
   d) PB2 要短接到某个 QEI 脚上（见《AY-MSEKIT_电机与电源控制》第 8 章））

## 3.3 课后

1) 在 TivaLaunchPad 上，使用 PWM 分别输出 1KHz、2KHz、3KHz 频率的方波，听蜂鸣器的发声情况，将代码贴到报告中,并解释原因。
2) 各小组提交本组 PWM 相关任务题目 1。

## 3.4 课堂 2

1) 结合各自的综合任务，完成一种 PWM 相关任务。