Application Notes

ZHCAAL4 – July 2021

电机控制系统绝对值编码器正交分频输出 Abs2QEP PTO(Pulse Train Output)基于 F28004x 的软件实现

Ricky Zhang *C2000 FAE / China*

摘要

绝对值编码器正交分频输出 Abs2QEP PTO(Pulse Train Output)功能是电机控制系统里面非常常见和必要的一个基本功能。它将通过绝对值编码器采集到的电机位置信息转换成任意整数或小数分频的与电机转速成比例的正交 QEP 格式的 A/B/Z 数字信号进行输出，反馈给上位机控制器 PLC 进行上报，或者给该电机驱动器的后一级电机驱动器实现实时的同步控制。本文提供了基于 F28004x MCU 的软件实现方法和参考代码 (https://www.ti.com.cn/cn/lit/zip/ZHCAAL4)，使得单芯片 F28004x 替换 CPLD 或者 FPGA 实现绝对值编码器解码和分频输出功能，或者单芯片 F2837x/38x 实现绝对值编码器解码和分频输出功能以及电机控制成为可能。

Contents

Figures

1 绝对值编码器正交分频输出 Abs2QEP PTO(Pulse Train Output)基本介绍

绝对值编码器正交分频输出 Abs2QEP PTO(Pulse Train Output)功能是电机控制系统里面非常常见和必要的一个基本功能。它将通过绝对值编码器采集到的电机位置信息转换成任意整数或小数分频的与电机转速成比例的正交 QEP 格式的 A/B/Z 数字信号进行输出，反馈给上位机控制器 PLC 进行上报，或者给该电机驱动器的后一级电机驱动器实现实时的同步控制。

在常见的交流通用伺服和专用伺服以及一些高端的交流变频器系统里，由于控制精度的要求，基本上都是使用绝对值编码器对电机位置进行采集。这些编码器包括支持各种通信协议比如 Endat2.2、Biss-C、Tamagawa 和 Nikon 的数字式绝对值编码器，和旋转变压器等模拟量绝对值编码器。电机的位置信息除了用于电流环路的控制以外，还需要进行分频输出。考虑传统电机控制器和驱动器的使用习惯，以及抗干扰的要求，通常都是正交 QEP 格式的 A/B/Z 数字信号(EtherCAT 的出现使得这个功能不再成为必须，但考虑兼容性仍非常必要)，而因为输出频率限制和同步控制比如不同的齿轮匝数比的要求，正交输出通常还需要进行任意整数或小数的分频。由于编码器解码的要求以及复杂的正交分频输出逻辑和时序，传统的实现方法是将两者合二为一，在 CPLD 或者 FPGA 上完成。

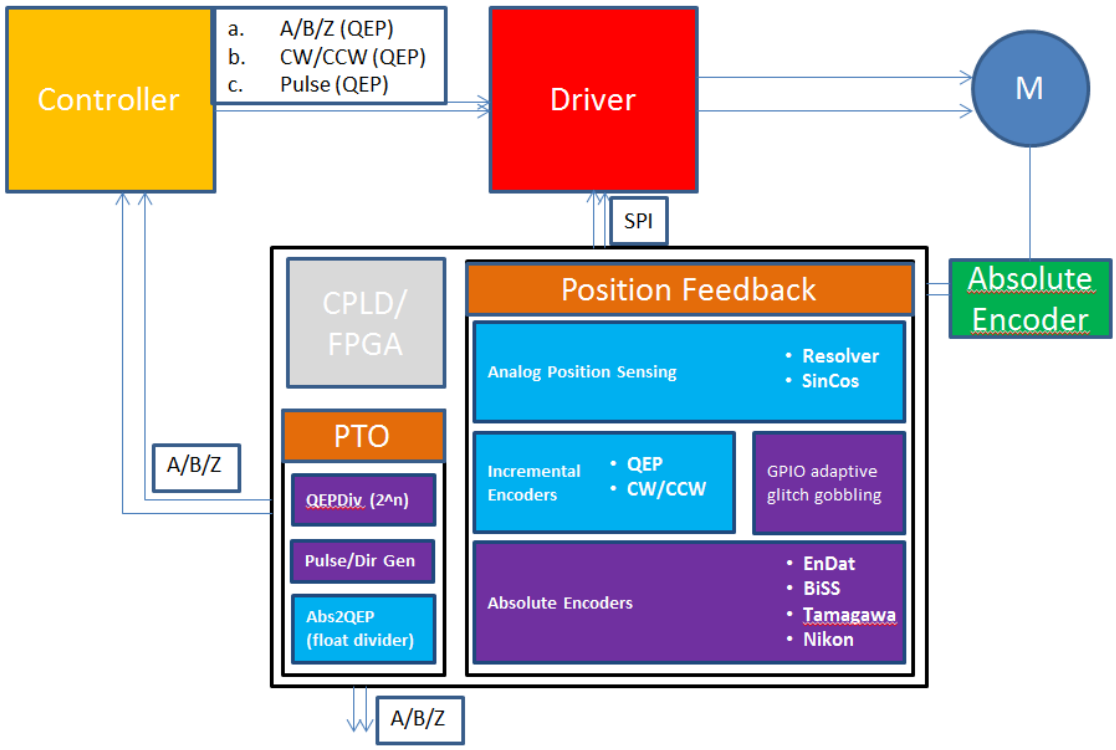

Figure 1.　绝对值编码器解码和正交分频输出

2 C2000 F2837x/38x 和 F28004x 的软件实现及意义

第三代 C2000 MCU 从 F2837x/38x 和 F28004x 开始，由于强大的电机控制功能的集成，使得单芯片实现绝对值编码器解码和正交分频输出成为可能。

2.1 F2837x/38x 和 F28004x 的软件实现

F2837x/38x 和 F28004x 片上提供了更强大的 C28x CPU 和 CLA 协处理器，Type 4 EPWM 模块以及可配置的逻辑单元 CLB，其中 CLB 可以支持各种编码器的解码，包括 Endat2.2、Biss-C、Tamagawa 和 Nikon 的数字式绝对值编码器，和旋转变压器等模拟量绝对值编码器，目前已经发布 CLB Tools 以更好地支持 PositionManager 功能，而 Type 4 EPWM 模块配合 C28x CPU 或 CLA 可以用于模拟绝对值编码器的分频输出功能，以有限的 CPU 带宽和片上资源进行软件实现，并达到与 CPLD 或者 FPGA 一样的高性能。

2.2 F2837x/38x 和 F28004x 的软件实现的意义

通过 F2837x/38x 和 F28004x 使用软件实现绝对值编码器的分频输出功能后，由于 PositionManager 可以同时实现各种绝对值编码器的解码，使得以下两种方案变得可能：

1. 用于实现绝对值编码器解码和分频输出功能的 CPLD 或者 FPGA 直接使用低成本的 C2000 系列芯片如 F28004x 替换，降低系统成本，简化生产维护，如下图 2。

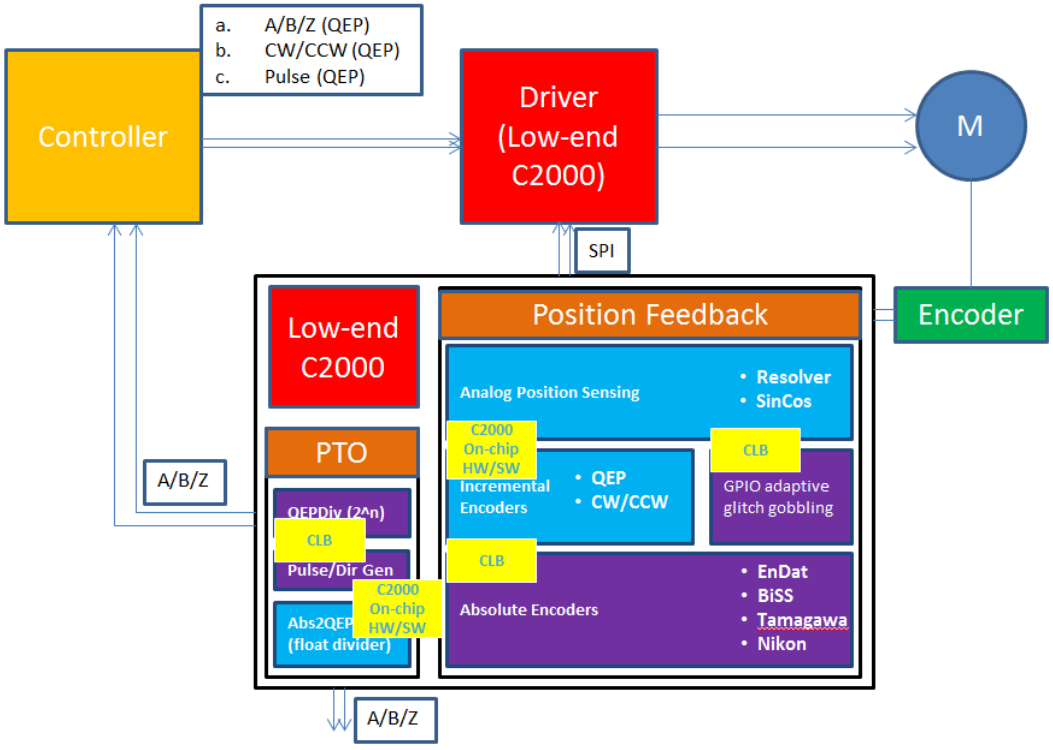

Figure 2. 低成本 F28004x 替换 CPLD 或者 FPGA

2. 将绝对值编码器解码和分频输出功能整合到驱动器控制芯片里比如 F2837x/38x，从而实现真正的单芯片控制，降低系统成本，简化系统控制，如下图 3。

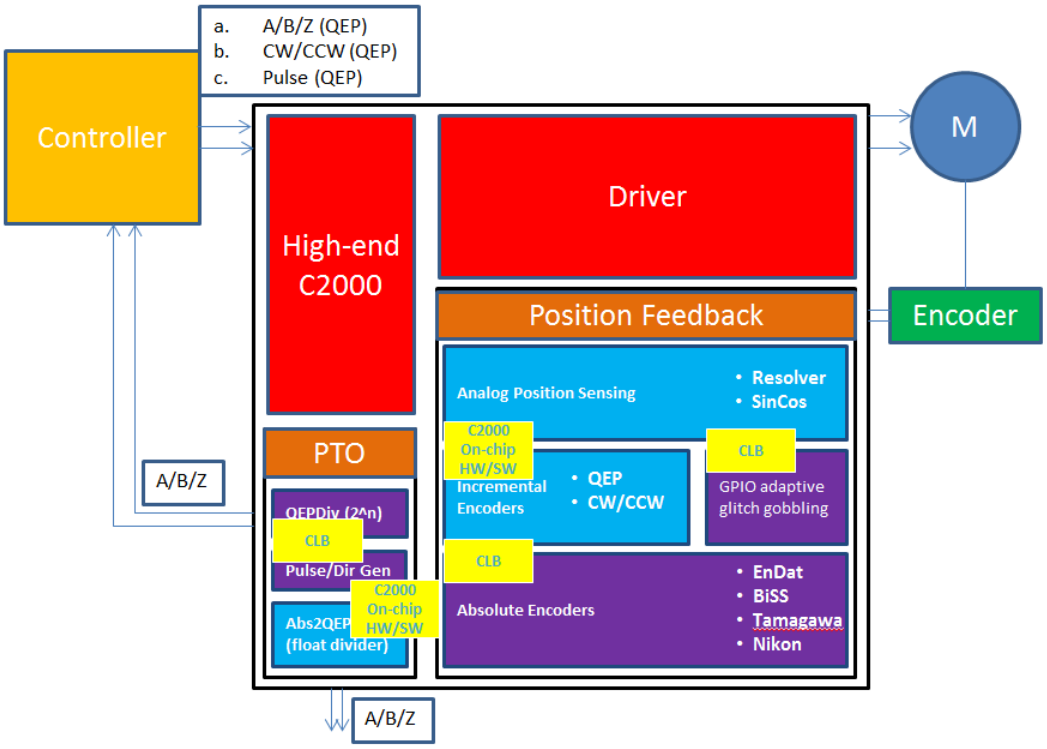

Figure 3.　单芯片 F2837x/38x 集成 CPLD 或者 FPGA 实现的功能

3　软件实现的基本思路

软件实现主要包含以下三方面的内容：

- 通过绝对值编码器获取电机位置，并计算采样时间间隔的位置差，从而得到电机转速；
- 根据电机转速和编码器线数，以及设定的分频系数，计算出脉冲个数；
- 根据电机转动方向，输出 50%占空比特定频率的 EPWM 信号模拟 QEP 正交输出并适时调整；

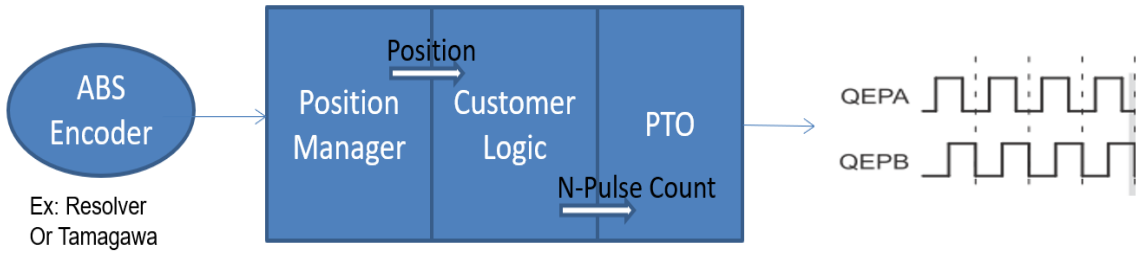

Figure 4.　绝对值编码器正交分频输出 Abs2QEP PTO

3.1 电机转速

在得到电机位置差的前提下，对于磁场极对数为 1 的电机来说，转速决定于采样时间间隔，即 n = (p2 – p1) / t，其中，p2 或 p1 代表两次采集的电机位置，t 是采样时间间隔，转速单位为 rpm，因此需要进行时间单位换算。

假定我们以 10kHz 的频率进行采样，则采样时间时隔为 100us，需要乘以 60 和10^6，电机位置如果用角度表示，则位置差需要除以 360°以得到圈数，比如两个位置 0°和 1.2°对应的就是 2000rpm，如果用 pu(per unit)表示，则位置差就是圈数。

3.2 脉冲个数

编码器线数因应用场景不同而有差异，比如交流伺服通常使用 14 线，而变频器使用 10 线，对应电机转动一圈的脉冲个数也会不同，为2^n，即 14 线对应 16384 个脉冲，10 线对应 1024 个脉冲。4 个脉冲组成一个完整的周期，因此计算时还需要除以 4，因此上述例子中，两个位置 0°和 1.2°在 100us 内对应的脉冲个数在 10 线时应为 0.85 个。

如果考虑分频输出，脉冲个数还需要除以相应的分频系数。同时考虑电机转动方向，脉冲个数使用有符号数表示，可以传入到 QEP 正交信号产生输出函数中作为参数。

3.3 QEP 正交输出

根据电机转动方向，可以设定 A/B 两路信号的输出相位领先或落后 90°的情况(比如正转时 A 领先，反转时落后)，而输出占空比固定为 50%，此时接收脉冲个数参数，可以根据之前一周期的实际输出情况和相位情况进行方向检测，输出频率和比较值寄存器计算，并实时修改和调整 EPWM 模块的相关配置，进行 EPWM 波形的输出，模拟 QEP 正交信号。

模拟输出过程中需要考虑三个实际问题。

一是脉冲个数的处理。由于实际采样时间间隔不同和编码器线数不同，得到的脉冲个数通常不会是整数，且分频计算也会导致小数的产生，因此实际模拟过程并不计算和直接提供输出频率，而是让脉冲产生和输出函数根据脉冲个数参数进行 EPWM 模块的周期值计算，且计算时直接使用浮点数，通过四舍五入进行本周期的输出，但累积的误差会在下一周期代入。

二是高速和低速的兼容。电机高速转动时的脉冲个数及变化都较大，通常不会存在问题，但低速时会出现一个采样时间间隔内可能只有 4 个或以下个脉冲，即不需要产生一个完整的 EPWM 波形，此时只需要知道脉冲个数，而不必调整 EPWM 的输出频率。

三是方向的变化和对应输出的处理。方向的变化可以很容易通过传入参数有符号数脉冲个数的正负值进行判断，此时可以方便地修改输出相位差变化的配置，即 EPWM 的 AQCTL 寄存器，但同时用于触发 CMPCTL 和 AQCTL 生效的同步事件对应的相位寄存器和计数方向寄存器也需要同步修改，以确保新的输出能生效并正常连接方向变化前的最后一个脉冲和方向变化后的第一个脉冲。

4 具体的软件实现

软件实现的重点和难点在于第三部分 QEP 正交输出的设置和处理，因此本文我们重点研究该部分的实现。以 F28004x 为例，假定用户程序已经通过 PositionManager 获取电机位置并计算出分频后需要的脉冲个数，且传入到 QEP 正交信号产生输出函数中作为参数，接下来将要通过配置 EPWM 生成正交输出，并根据每一次的传入参数实时更新 EPWM 的输出频率或脉冲个数，以及对应的 EPWM 设置以满足电机方向变化和高低速兼容的要求。

4.1 EPWM1 和 EPWM2 的基本配置

按照之前的例子，我们假定程序以 10kHz 的频率进行电机位置的采集，则需要一个相同频率的中断应用程序对 EPWM 波形输出进行相应的配置，此处我们选择 EPWM1 模块来实现。

EPWM1 采用对称计数模式进行 EPWM 模块的 TBCTR 计数，频率为 10kHz，在每次计数过零点产生中断用于执行 QEP 正交信号产生和输出，同时输出 EPWM1A 信号来表征它的计数过零点，输出 EPWM1B 用于 EPWM2 模块的同步，它是一个 2.5us 的小脉冲，产生在计数过零点的 20us 后，确保相关计算已经完成。

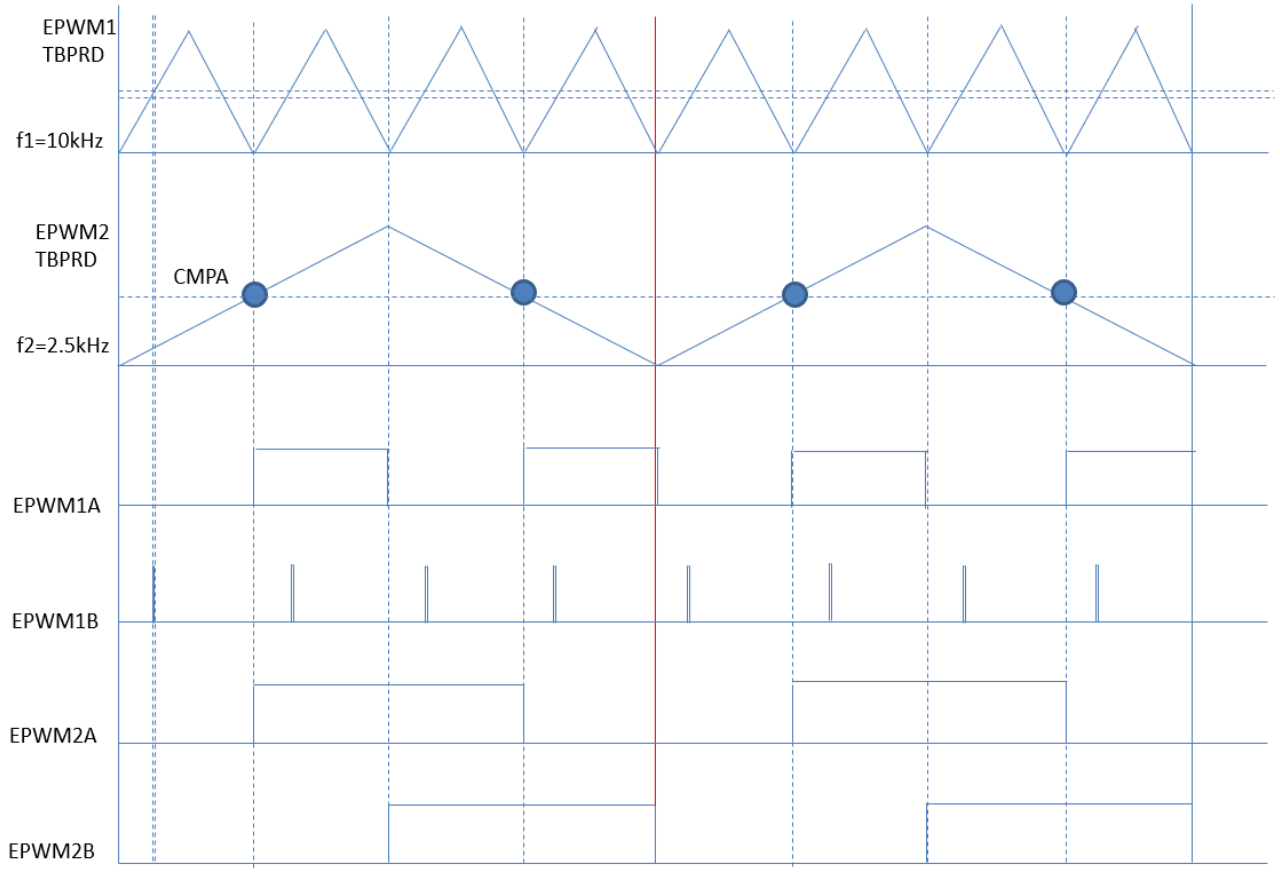

Figure 5. EPWM1 和 EPWM2 的基本配置及输出波形

EPWM2 用于模拟 QEP 正交信号的输出，也采用对称计数模式，设定基频为 2.5kHz，输出 EPWM2A 和 EPWM2B 对应脉冲输出 QEPA 和 QEPB，均为 50%占空比，但存在相位领先或落后 90°的差别。

定义电机正转时，对应的正交分频输出 QEPA 脉冲信号领先 QEPB 脉冲信号 90°，反之电机反转时，对应的正交分频输出 QEPA 脉冲信号落后 QEPB 脉冲信号 90°。

参考代码如下：

```
EPwm1Regs.TBPRD = EPWM1_TIMER_TBPRD;       // Set timer period
EPwm1Regs.TBPHS.bit.TBPHS = 0x0000;        // Phase is 0
EPwm1Regs.TBCTR = 0x0000;                  // Clear counter
// Set Compare values
EPwm1Regs.CMPA.bit.CMPA = EPWM1_CMPA;      // Set compare A value
EPwm1Regs.CMPB.bit.CMPB = EPWM1_CMPB;      // Set compare B value
// Setup counter mode
EPwm1Regs.TBCTL.bit.CTRMODE = TB_FREEZE; // Count up and down
EPwm1Regs.TBCTL.bit.PHSEN = TB_DISABLE;        // Disable phase loading
EPwm1Regs.TBCTL.bit.SYNCOSEL = TB_SYNC_DISABLE;// Sync out disabled (defined by
TBCTL2[SYNCOSELX], default to be disabled)
EPwm1Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;       // Clock ratio to SYSCLKOUT
EPwm1Regs.TBCTL.bit.CLKDIV = TB_DIV1;
// Set actions
EPwm1Regs.AQCTLA.bit.ZRO = AQ_TOGGLE;          // Toggle PWM1A on ZRO
EPwm1Regs.AQCTLB.bit.CAU = AQ_SET;             // Set PWM1B on CAU
EPwm1Regs.AQCTLB.bit.CBU = AQ_CLEAR;           // Clear PWM1B on CBU
// Interrupt where we will change the Compare Values
EPwm1Regs.ETSEL.bit.INTSEL = ET_CTR_ZERO;     // Select INT on Zero event
EPwm1Regs.ETSEL.bit.INTEN = 1;                 // Enable INT
EPwm1Regs.ETPS.bit.INTPRD = ET_1ST;            // Generate INT on 1st event


EPwm2Regs.TBPRD = EPWM2_TIMER_TBPRD;         // Set timer period
EPwm2Regs.TBPHS.bit.TBPHS = 0x0000;          // Phase is 0
EPwm2Regs.TBCTR = 0x0000;                    // Clear counter
// Set Compare values
EPwm2Regs.CMPA.bit.CMPA = EPWM2_CMPA;        // Set compare A value
```

```
    // Setup counter mode
    EPwm2Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN;      // Freeze count and will work in
count up and down mode
    EPwm2Regs.TBCTL.bit.PHSEN = TB_ENABLE;        // Enable phase loading
    EPwm2Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;      // Clock ratio to SYSCLKOUT
    EPwm2Regs.TBCTL.bit.CLKDIV = TB_DIV1;
    EPwm2Regs.TBCTL2.bit.PRDLDSYNC = TB_SYNC;
    // Setup shadowing for CMPA, AQCTLA and AQCTLB
    EPwm2Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;        // CMPA load in shadow mode
    EPwm2Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO;      // Load when TBCTR=ZRO
    EPwm2Regs.CMPCTL.bit.LOADASYNC = CC_SYNC;          // Load when SYNC is received
    EPwm2Regs.AQCTL.bit.SHDWAQAMODE = AQ_SHADOW;       // AQCTLA load in shadow
mode
    EPwm2Regs.AQCTL.bit.SHDWAQBMODE = AQ_SHADOW;       // AQCTLB load in shadow
mode
    EPwm2Regs.AQCTL.bit.LDAQAMODE = AQ_CTR_ZERO;       // Load when TBCTR=ZRO
    EPwm2Regs.AQCTL.bit.LDAQBMODE = AQ_CTR_ZERO;       // Load when TBCTR=ZRO
    EPwm2Regs.AQCTL.bit.LDAQASYNC = AQ_SYNC;          // Load when SYNC is received
    EPwm2Regs.AQCTL.bit.LDAQBSYNC = AQ_SYNC;          // Load when SYNC is received
```

4.2 电机转动方向的判定及四象限的定义

电机转动方向的判定由传入参数有符号数的脉冲个数完成，当它为正数时，表示正转，反之反转。

在 EPWM2 的基本配置里，设定 EPWM2A 输出动作控制寄存器 AQCTLA 的配置始终保持不变，在计数上升到比较值寄存器即满足 CAU 时置高电平输出，在计数下降到比较值寄存器即满足 CAD 时拉低电平输出，而 EPWM2B 输出动作控制寄存器 AQCTLB 则在每个采样和计算周期都根据电机转动方向进行调整，它的初始配置为正转，在计数上升到周期值即满足 PRD 时置高电平输出，在计数下降到计数零点即满足 ZRO 时拉低电平输出，反转时则逻辑相反。

参考代码如下：

```
// CW (clockwise)
    if (QEPPulseNumberAbs > 0)
    {
       Direction = 0;
```

```
    EPwm2Regs.AQCTLA.bit.CAU = AQ_SET;            // Set PWM2A on CAU
    EPwm2Regs.AQCTLA.bit.CAD = AQ_CLEAR;          // Clear PWM2A on CAD
    EPwm2Regs.AQCTLA.bit.ZRO = AQ_CLEAR;          // Clear PWM2A on ZRO
    EPwm2Regs.AQCTLA.bit.PRD = AQ_SET;            // Set PWM2A on PRD
    EPwm2Regs.AQCTLB.bit.CAU = AQ_CLEAR;          // Clear PWM2B on CAU
    EPwm2Regs.AQCTLB.bit.CAD = AQ_SET;            // Set PWM2B on CAD
    EPwm2Regs.AQCTLB.bit.ZRO = AQ_CLEAR;          // Clear PWM2B on ZRO
    EPwm2Regs.AQCTLB.bit.PRD = AQ_SET;            // Set PWM2B on PRD
}
// CCW (counter clockwise)
else if (QEPPulseNumberAbs < 0)
{
    Direction = 1;
    EPwm2Regs.AQCTLA.bit.CAU = AQ_SET;            // Set PWM2A on CAU
    EPwm2Regs.AQCTLA.bit.CAD = AQ_CLEAR;          // Clear PWM2A on CAD
    EPwm2Regs.AQCTLA.bit.ZRO = AQ_CLEAR;          // Clear PWM2A on ZRO
    EPwm2Regs.AQCTLA.bit.PRD = AQ_SET;            // Set PWM2A on PRD
    EPwm2Regs.AQCTLB.bit.CAU = AQ_SET;            // Set PWM2B on CAU
    EPwm2Regs.AQCTLB.bit.CAD = AQ_CLEAR;          // Clear PWM2B on CAD
    EPwm2Regs.AQCTLB.bit.ZRO = AQ_SET;            // Set PWM2B on ZRO
    EPwm2Regs.AQCTLB.bit.PRD = AQ_CLEAR;          // Clear PWM2B on PRD
}
else if (QEPPulseNumberAbs == 0)
{
    EPwm2Regs.AQCTLA.bit.CAU = AQ_NO_ACTION;      // Do nothing
    EPwm2Regs.AQCTLA.bit.CAD = AQ_NO_ACTION;      // Do nothing
    EPwm2Regs.AQCTLA.bit.ZRO = AQ_NO_ACTION;      // Do nothing
    EPwm2Regs.AQCTLA.bit.PRD = AQ_NO_ACTION;      // Do nothing
    EPwm2Regs.AQCTLB.bit.CAU = AQ_NO_ACTION;      // Do nothing
```

```
    EPwm2Regs.AQCTLB.bit.CAD = AQ_NO_ACTION;        // Do nothing
    EPwm2Regs.AQCTLB.bit.ZRO = AQ_NO_ACTION;        // Do nothing
    EPwm2Regs.AQCTLB.bit.PRD = AQ_NO_ACTION;        // Do nothing
}
```

对于每一组 QEPA 和 QEPB 脉冲信号，无论正转或反转，有且仅有四组关系，我们定义它为四象限，分别对应两个脉冲信号不同的高低状态，即象限 1 对应 QEPA 为低和 QEPB 为低，象限 2 对应 QEPA 为高和 QEPB 为低，象限 3 对应 QEPA 为高和 QEPB 为高，和象限 4 对应 QEPA 为低和 QEPB 为高。程序第一次运行初始化时，通过判断对应 GPIO2 和 GPIO3 的电平状态，记录该象限值，此后将在更新 EPWM2 相位寄存器和计数方向寄存器时使用，用于实现 EPWM2 被 EPWM1 同步以后的重新计数。

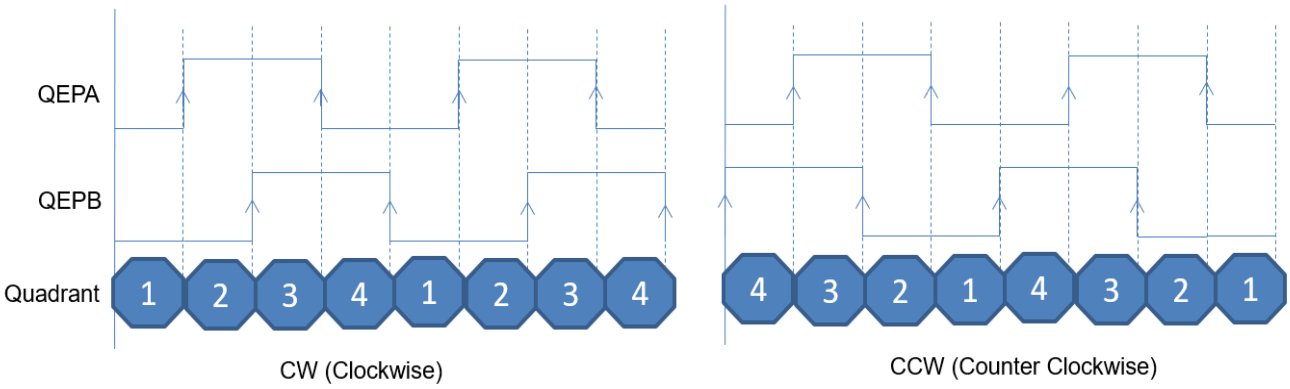

Figure 6.　正反转时的正交分频输出 QEPA 和 QEPB 脉冲信号关系及四象限

参考代码如下：

```
QEPPulseAStatus = GpioDataRegs.GPADAT.bit.GPIO2;
QEPPulseBStatus = GpioDataRegs.GPADAT.bit.GPIO3;
if ((QEPPulseAStatus == 0) && (QEPPulseBStatus == 0))
{
    QEPPulseQuadrant = FIRST_QUADRANT;
}
else if ((QEPPulseAStatus == 1) && (QEPPulseBStatus == 0))
{
    QEPPulseQuadrant = SECOND_QUADRANT;
}
else if ((QEPPulseAStatus == 1) && (QEPPulseBStatus == 1))
```

```
{
    QEPPulseQuadrant = THIRD_QUADRANT;
}
else if ((QEPPulseAStatus == 0) && (QEPPulseBStatus == 1))
{
    QEPPulseQuadrant = FOURTH_QUADRANT;
}
```

4.3 EPWM2 周期的计算和更新

EPWM2 周期的计算取决于传入参数脉冲个数，当它为零值时，表示 2.5kHz 基频可以满足要求，直接保持周期值为默认的 2.5kHz 基频对应的周期值，当它为非零值时，它应该是基频的倍数，因而周期寄存器的值应该被赋予默认的 2.5kHz 基频对应的周期值除以脉冲个数得到的值，而比较值寄存器 CMPA 应该为计算周期值的一半，以满足 50%占空比要求。

需要注意的是，为了保证四舍五入，此处通过浮点型变量进行计算，在每一周期的计算完成后，取整赋给对应的寄存器，并立即触发 EPWM1 的同步信号输出，进行 EPWM2 的同步，重新开启新周期进行 EPWM 的输出。

参考代码如下：

```
if (QEPPulseNumberAbs == 0)
{
    Epwm2Period = (float)(EPWM2_TIMER_TBPRD);
    Epwm2Cmpa = Epwm2Period/2;
}
else
{
    Epwm2Period = (float)(EPWM2_TIMER_TBPRD/QEPPulseNumberAbs);
    Epwm2Cmpa = Epwm2Period/2;
}
```

4.4 EPWM2 的同步

EPWM2 的正交分频输出按照计算完成立即更新的形式，但不是通过使能 EPWM2 的立即模式，因为同时涉及到 CMPA 寄存器和 AQCTL 寄存器。我们采用 EPWM1 在计算完成后产生一个脉冲长度为 2.5us 的 EPWM1B 信号送到 EPWM2 模块进行同步，同时设置 EPWM2 的 CMPCTL 和 AQCTL 寄存器使得 CMPA 和 AQCTL 寄存器可以在接收到同步信号后完成 shadow 寄存器到 active 寄存器的加载，从而立即执行新的 CMPA 值和 AQCTL 触发动作，并且此时的 TBCTR 也按照同步以后的新的相位值和新的计数方向进行计数，实现无缝对接前一周期的正交分频脉冲输出。

同步以后的新的相位值和新的计数方向根据四象限及当前方向决定，基本原则是根据是否有电机转动方向的改变确定是否有相位领先或落后的变化，以及更新 TBCTR 寄存器值为新的相位值，从而调整 EPWM2 模块的 TBCTR 计数方向和计数值达到重新计数开启新周期的目的。完成以后，需要同时计算和保存新脉冲对应的的象限值，用于下一次计算。

参考代码如下：

```
if (!Direction)
{
   if (QEPPulseQuadrant == SECOND_QUADRANT)
   {
      Epwm2Phase = Epwm2Period/2 + EPWM2_TBPHS_DELAY;
      EPwm2Regs.TBCTL.bit.PHSDIR = TB_UP;
   }
   else if (QEPPulseQuadrant == THIRD_QUADRANT)
   {
      Epwm2Phase = Epwm2Period - EPWM2_TBPHS_DELAY;
      EPwm2Regs.TBCTL.bit.PHSDIR = TB_DOWN;
   }
   else if (QEPPulseQuadrant == FOURTH_QUADRANT)
   {
      Epwm2Phase = Epwm2Period/2 - EPWM2_TBPHS_DELAY;
      EPwm2Regs.TBCTL.bit.PHSDIR = TB_DOWN;
   }
   else if (QEPPulseQuadrant == FIRST_QUADRANT)
   {
      Epwm2Phase = EPWM2_TBPHS_DELAY;
```

```
            EPwm2Regs.TBCTL.bit.PHSDIR = TB_UP;
        }
    }
    else
    {
        if (QEPPulseQuadrant == FIRST_QUADRANT)
        {
            Epwm2Phase = EPWM2_TBPHS_DELAY;
            EPwm2Regs.TBCTL.bit.PHSDIR = TB_DOWN;   //TB_UP;
        }
        else if (QEPPulseQuadrant == SECOND_QUADRANT)
        {
            Epwm2Phase = Epwm2Period/2 + EPWM2_TBPHS_DELAY;
            EPwm2Regs.TBCTL.bit.PHSDIR = TB_DOWN;
        }
        else if (QEPPulseQuadrant == THIRD_QUADRANT)
        {
            Epwm2Phase = Epwm2Period - EPWM2_TBPHS_DELAY;
            EPwm2Regs.TBCTL.bit.PHSDIR = TB_UP;   //TB_DOWN;
        }
        else if (QEPPulseQuadrant == FOURTH_QUADRANT)
        {
            Epwm2Phase = Epwm2Period/2 - EPWM2_TBPHS_DELAY;
            EPwm2Regs.TBCTL.bit.PHSDIR = TB_UP;
        }
    }
    if (!Direction)
    {
```

```
        QEPPulseQuadrant = (QEPPulseQuadrant +
QEPPulseNumberDelta)%FOURTH_QUADRANT;

    }

    else

    {

        QEPPulseQuadrant1 = (QEPPulseNumberDeltaAbs/FOURTH_QUADRANT + 1)<<2;

        QEPPulseQuadrant = (QEPPulseQuadrant + QEPPulseQuadrant1 +
QEPPulseNumberDelta)%FOURTH_QUADRANT;

    }

    if (QEPPulseQuadrant == 0)

    {

       QEPPulseQuadrant = FOURTH_QUADRANT;

    }
```

EPWM1B 同步信号的配置通过 INPUT X-BAR 1 接收来自 GPIO1 的 EPWM1B 信号，送至 EPWM X-BAR 4 生成 TRIP4 信号，然后在 EPWM2 中通过数字比较子模块(Digital Compare Sub-module)接收 TRIP4 信号用于产生 DCAEVT1 事件，触发同步动作。

参考代码如下：

```
   InputXbarRegs.INPUT1SELECT = 1;                  // INPUT X-BAR 1 = GPIO1 = EPWM1B

   EPwmXbarRegs.TRIP4MUX0TO15CFG.bit.MUX1 = 1;        // EPWM X-BAR TRIP 4 = INPUT X-
BAR 1

   EPwmXbarRegs.TRIP4MUXENABLE.bit.MUX1 = 1;

    EPwm2Regs.DCTRIPSEL.bit.DCAHCOMPSEL = DC_TRIPIN4;    // DCAH = TRIPIN4 = INPUT
X-BAR 1 = GPIO1 = EPWM1B

    EPwm2Regs.TZDCSEL.bit.DCAEVT1 = TZ_DCAH_HI;         // DCAH = High and DCAL = Don't
care, Trigger DCAEVT1 when EPWM1B goes high

    EPwm2Regs.TZCTL.bit.DCAEVT1 = TZ_NO_CHANGE;

    EPwm2Regs.TZCTL.bit.DCAEVT2 = TZ_NO_CHANGE;

    EPwm2Regs.TZCTL.bit.DCBEVT1 = TZ_NO_CHANGE;

    EPwm2Regs.TZCTL.bit.DCBEVT2 = TZ_NO_CHANGE;

    EPwm2Regs.TZCTL.bit.TZA = TZ_NO_CHANGE;

    EPwm2Regs.TZCTL.bit.TZB = TZ_NO_CHANGE;

    EPwm2Regs.DCACTL.bit.EVT1SYNCE = DC_SYNC_ENABLE;
```

4.5 软件实现流程图

完整的软件实现流程图如下图 7：

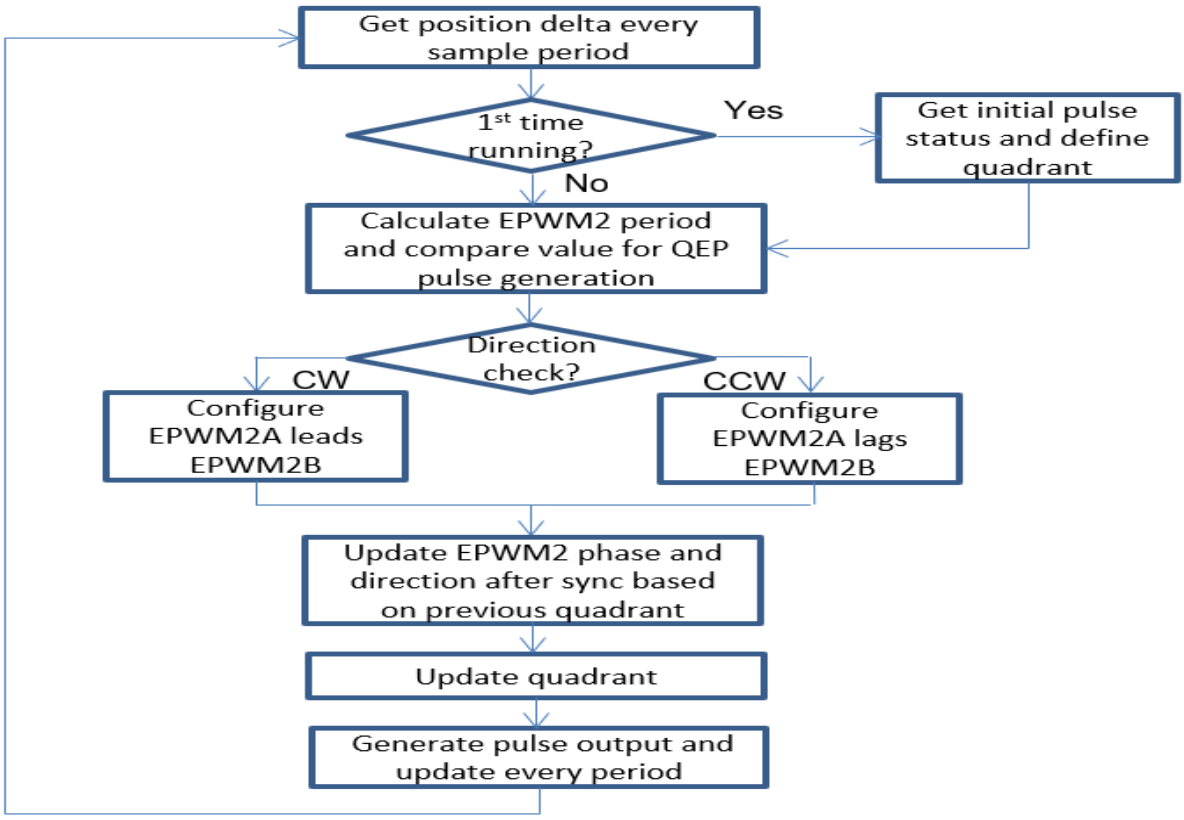

Figure 7.　软件实现流程图

5　小结

本文通过软件配置使用 Type 4 EPWM 模块模拟实现了绝对值编码器正交分频输出脉冲信号的 Abs2QEP PTO 功能，使得单芯片 F28004x 替换 CPLD 或者 FPGA 实现绝对值编码器解码和分频输出功能，或者单芯片 F2837x/38x 实现绝对值编码器解码和分频输出功能以及电机控制成为可能。

在 100MHz 主频的 F28004x 芯片上，使用 10kHz 频率进行电机位置采集和计算并实现正交分频输出脉冲信号的代码执行时间大约为 3us，只占用 3%的 CPU 带宽，可以满足绝大多数应用场景。如果使用 CLA 进行运算，则不占用 CPU 带宽。实际的脉冲输出频率经测试最高可达 350kHz，对应于 1024 线编码器的转速为 20500rpm，也可基本满足绝大多数应用场景。

6　参考文献

1. *TMS320F28004x Piccolo™ Microcontrollers (SPRS945)*
2. *TMS320F28004x Piccolo Microcontrollers Technical Reference Manual (SPRUI33)*

重要声明和免责声明

TI 提供技术和可靠性数据（包括数据表）、设计资源（包括参考设计）、应用或其他设计建议、网络工具、安全信息和其他资源，不保证没有瑕疵且不做出任何明示或暗示的担保，包括但不限于对适销性、某特定用途方面的适用性或不侵犯任何第三方知识产权的暗示担保。

这些资源可供使用 TI 产品进行设计的熟练开发人员使用。您将自行承担以下全部责任：(1) 针对您的应用选择合适的 TI 产品，(2) 设计、验证并测试您的应用，(3) 确保您的应用满足相应标准以及任何其他安全、安保或其他要求。这些资源如有变更，恕不另行通知。TI 授权您仅可将这些资源用于研发本资源所述的 TI 产品的应用。严禁对这些资源进行其他复制或展示。您无权使用任何其他 TI 知识产权或任何第三方知识产权。您应全额赔偿因在这些资源的使用中对 TI 及其代表造成的任何索赔、损害、成本、损失和债务，TI 对此概不负责。

TI 提供的产品受 TI 的销售条款 (https:www.ti.com.cn/zh-cn/legal/termsofsale.html) 或 ti.com.cn 上其他适用条款/TI 产品随附的其他适用条款的约束。TI 提供这些资源并不会扩展或以其他方式更改 TI 针对 TI 产品发布的适用的担保或担保免责声明。

邮寄地址：上海市浦东新区世纪大道 1568 号中建大厦 32 楼，邮政编码：200122