AN908

使用 dsPIC30F 实现交流感应电机的矢量控制

作者：	*Dave Ross 和 John Theys* *Diversified Engineering Inc.*
第二作者：	*Steve Bowling* *Microchip Technology Inc.*

引言

该应用笔记介绍了为 dsPIC30F 系列器件编写的矢量控制应用程序。除简要介绍了有关控制理论的知识外，文中所涉及内容需要读者对交流感应电机（AC Induction Motor，ACIM）的特性具有基本的了解。可参阅文中提到的参考文献，获得相关的背景知识。

软件特征

矢量控制软件具有以下特征：

- 软件采用间接磁通控制法实现交流感应电机的矢量控制。
- 控制环周期为 50 µs，软件需要大约 9 MIPS 的 CPU 开销（不到全部 CPU 能力的 1/3）。
- 应用程序需要占用 258 字节的数据存储空间以及 256 字节的常量存储空间。用户界面需要占用大约 8 K 字节的程序存储空间。
- 根据对存储空间的要求，该应用程序可在 dsPIC30F2010 上运行。 dsPIC30F2010 是目前 dsPIC30F 系列中体积最小、成本最低的一款器件。
- 通过使能可选的诊断模式可在示波器上对内部程序变量进行实时观察。借助该特性，可方便地进行控制环参数的调整。

矢量控制理论

背景知识

由于具有结构简单以及耐久性强的特点，因此交流感应电机是工业和民用电机应用中的主要设备。这种电机没有电刷和磁钢，因此具有较高的可靠性和较低的成本。转子采用简单的钢质鼠笼结构。

ACIM 设计为运行于恒定输入电压和频率条件下，但通过改变电机输入电压频率的控制方法也可使 ACIM 有效运行在开环变速的应用中。如果电机不处于过载状态，电机运行速度近似与输入频率成正比。当希望通过减小驱动电压的频率以降低电机转速时，也应同时按比例减小电压的幅值。否则，当输入频率较低时电机会出现过电流。这种控制方法称为“压频比控制”。

实际应用中，用户采用的电压－频率关系曲线应能确保电机在任何速度设定点都能正常工作。该关系曲线可以采用查找表的形式或通过实时计算的方式获得。实际应用中通常使用一个斜率变量来定义任何运行点驱动频率和电压之间的线性关系。可采用压－频比控制法，通过速度和电流传感器，控制电机闭环运行。

压频比控制方法非常适用于风机和泵等负载变化较慢的应用场合。但对于需要快速动态响应的应用场合则效果不佳。特别是在速度或转矩发生快速变化时，由于较高的转差率则会导致较大的瞬态电流。如果同时对电机转矩和磁通进行闭环控制，则可以在不产生大电流的条件下实现快速的动态响应。矢量控制技术可以实现上述目标，通常也被称作磁场定向控制（FOC）。

矢量控制的优点在于可以直接实现较低的能量消耗。这样可实现更高的效率、较低的运行费用以及降低驱动元件成本。

矢量控制

传统的控制方法，比如前面介绍的压频比控制，都是对电机驱动电压的频率和幅值进行控制。与之相比，矢量控制法则同时控制电机驱动电压的频率、幅值以及**相位**。矢量控制的关键在于产生一个三相电压矢量以控制三相定子电流矢量。三相定子电流矢量可以控制转子磁通矢量并最终控制转子电流矢量。

最终，需要对转子电流分量进行控制。因为转子为钢质鼠笼结构且不存在直接电气连接，因此转子电流难以测量。由于转子电流不能直接测量，应用程序中使用可直接测量的参数来间接计算这些参数。

本应用笔记中介绍的控制技术称为间接矢量控制，这是因为没有直接对转子电流进行控制。使用以下数据可实现转子电流的间接矢量控制：

- 瞬时定子相电流，i_a、i_b 和 i_c
- 转子机械速度
- 转子电气时间常数

电机需配置用以检测三相定子电流的传感器以及转子速度反馈装置。

从不同的角度理解矢量控制

理解矢量控制如何工作的关键是要在头脑中设想参考坐标变换过程。当考虑交流感应电机如何工作时，你可能从定子的角度来设想其运行过程。从这一角度，定子绕组上施加了正弦输入电流，该时变信号产生了旋转的磁通。转子的速度将是旋转磁通矢量的函数。从定子静止坐标系的角度来看，定子电流和旋转磁通矢量看似交流量。

现在，不再采用前面的角度，而是设想你所处的平面以相同的速度随着定子电流产生的旋转磁通矢量进行同步旋转。从这一角度来观察电机处于稳态条件下的情况，此时定子电流看似常量而转子磁通矢量则是静止的！最终，你可能需要控制定子电流来获得期望的转子电流（不能直接测量获得）。通过坐标变换，可使用标准的控制环，如同控制直流量一样实现对定子电流的控制。

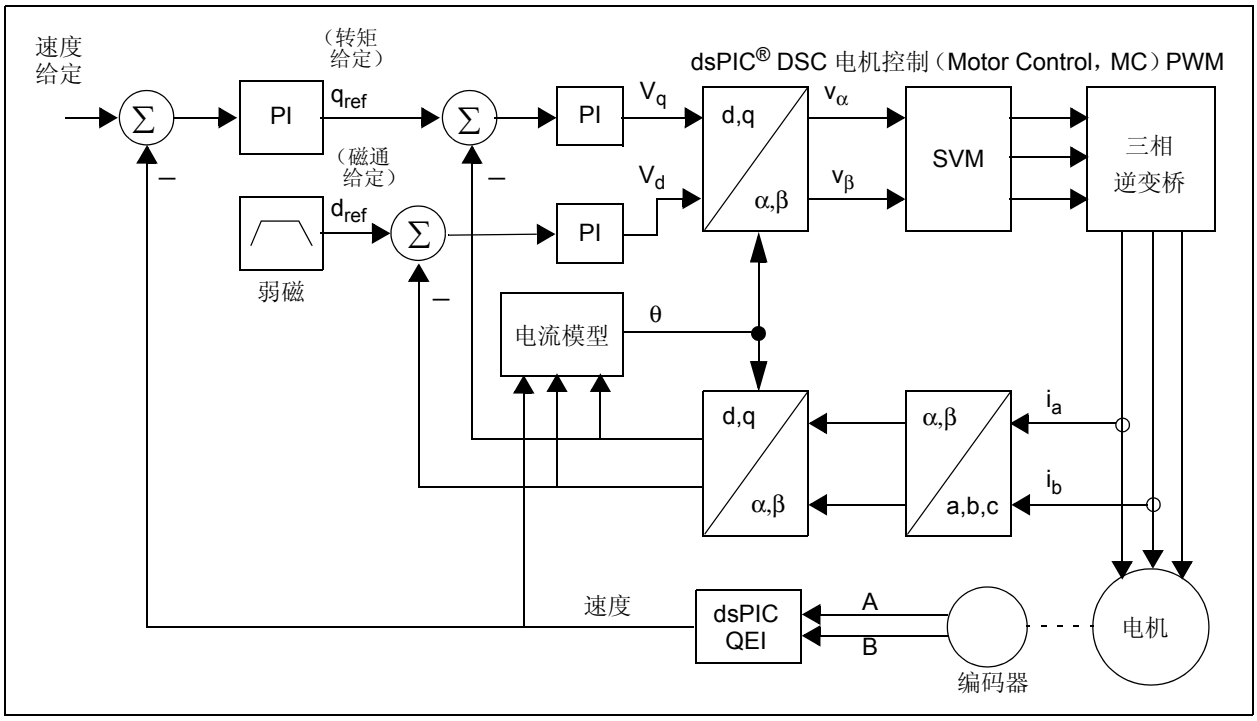
矢量控制综述

以下总结了间接矢量控制的实现步骤：

1. 测量三相定子电流，可分别获得 i_a、i_b 和 i_c。同时测量转子速度。
2. 将三相电流变换至 2 轴系统。该变换将三相相电流测量值 i_a、i_b 和 i_c 变换为变量 i_α 和 i_β。从定子的角度来看，i_α 和 i_β 是互为正交的时变电流值。
3. 通过使用控制环上一次迭代计算出的变换角，旋转 2 轴坐标系使之以转子磁通定向并随之同步旋转。i_α 和 i_β 变量经过该变换可获得 I_d 和 I_q 变量。I_d 和 I_q 变量为变换到旋转坐标系下的正交电流。在静态条件下，I_d 和 I_q 是常量。
4. 通过将实际的 I_d、I_q 与给定值进行比较获得各自的误差信号。I_d 给定值用以控制转子磁通。I_q 给定则用以控制电机的转矩输出。误差信号作为 PI 控制器的输入。控制器的输出为 V_d 和 V_q，这两个变量是施加到电机上的电压矢量在双轴上的两个分量。
5. 计算新的坐标变换角。该计算子程序的输入参数包括电机转速、转子电气时间常数、I_d 和 I_q。新的角度将告知算法下一个电压矢量在何处以获得当前运行条件下所需的转差率。
6. 通过使用新的坐标变换角可将PI控制器的输出变量 V_d 和 V_q 变换至静止参考坐标系。该计算将产生正交电压值 v_α 和 v_β。
7. v_α 和 v_β 值经过反变换得到 三相电压值 v_a、v_b 和 v_c。该三相电压值用来计算新的 PWM 占空比以生成所期望的电压矢量。

图 1 显示了坐标变换、PI 迭代、反变换以及 PWM 发生的整个过程。

图 1：　　矢量控制结构框图



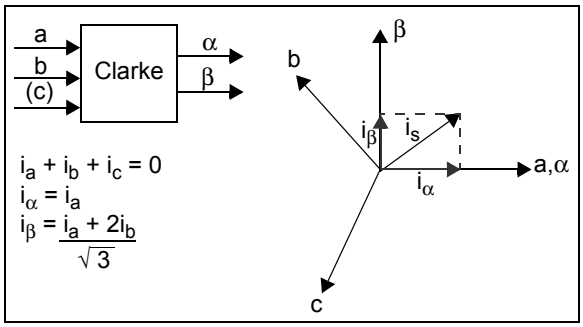
坐标变换

经过一系列的坐标变换，可以间接地确定不随时间变化的转矩和磁通值，并可采用经典的 PI 控制环对其进行控制。控制过程起始于三相相电流的测量。实际上，三相无中线系统的三相电流瞬时值的和总是为零。利用这一约束条件，通过测量两相电流即可知道所有三相电流。由于只需两个电流传感器，因此可以降低硬件成本。

CLARKE 变换

首先是将基于 3 轴、2 维的定子静止坐标系的各物理量变换到 2 轴的定子静止坐标系中。该过程称为 Clarke 变换，如图 2 所示。

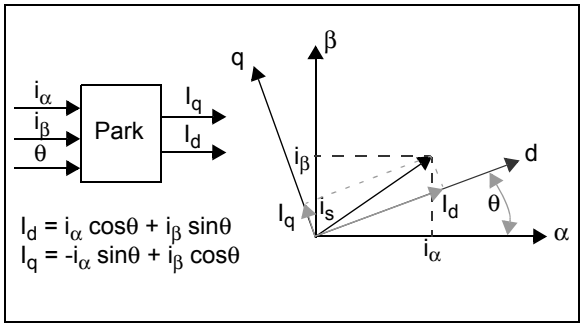
图 2：　　CLARKE 变换



PARK 变换

此刻，已获得基于 α-β 2轴正交坐标系的定子电流矢量。下一步是将其变换至随转子磁通同步旋转的 2 轴系统中。该变换称为 Park 变换，如图 3 所示。该 2 轴旋转坐标系称为 d-q 轴坐标系。

图 3：　　PARK 变换



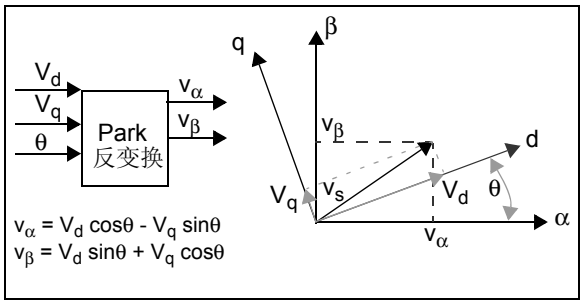
从这一角度来看，基于 d-q 坐标系电流矢量的 d-q 轴分量是不随时间变化的。在稳态条件下，它们是直流量。

定子电流的 d 轴分量与磁通成正比，而其 q 轴分量则与转矩成正比。由于定子电流的 d-q 轴分量皆可用直流量的形式来表示，因此可采用经典 PI 控制环策略对其分别进行控制。

PARK 反变换

经过 PI 迭代后，可获得旋转 d-q 坐标系中电压矢量的两个分量。此时需经过反变换将其重新变换到三相电机电压。首先，需从 2 轴旋转 d-q 坐标系变换至 2 轴静止 α-β 坐标系。该变换为 Park 反变换，如图 4 所示。

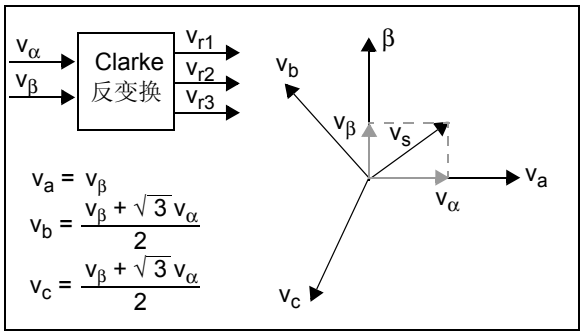
图 4： PARK 反变换



CLARKE 反变换

下一步是将静止 α-β 2 轴坐标系变换到定子静止 3 轴、三相参考坐标系。从数学角度来看，该变换是通过 Clarke 反变换来实现的，如图 5 所示。

图 5： CLARKE 反变换



磁通估计器

在鼠笼式异步电机中，转子机械转速略小于旋转磁场的转速。两者之间角速度的差异称作转差率，并以旋转磁通速度的百分比来表示。例如，如果转子转速和磁通旋转速度相同，则转差率为 0；而当转子转速为 0 时，转差率则为 1。

读者可能已注意到，Park 变换和 Park 反变换需要一个输入角 θ。变量 θ 表征转子磁通矢量的角位置。转子磁通矢量正确的角位置应通过已知值和电机参数来估计。该估计过程中应用了电机等效电路模型。电机运行所需的转差率在磁通估计器公式中得到反映，并包含在角度计算中。

磁通估计器根据定子电流、转子转速以及转子电气时间常数来计算新的磁通位置。磁通估计的实现是基于电机电流模型，特别是以下三个公式：

公式 1： 励磁电流

$$I_{mr} = I_{mr} + \frac{T}{T_r}(I_d - I_{mr})$$

公式 2： 磁通转速

$$f_s = (P_{pr} \cdot n) + \left(\frac{1}{T_r \omega_b} \cdot \frac{I_q}{I_{mr}}\right)$$

公式 3： 磁通角

$$\theta = \theta + \omega_b \cdot f_s \cdot T$$

其中：

I_{mr} = 励磁电流 （通过测量值来计算）

f_s = 磁通转速 （通过测量值来计算）

T = 采样 （循环）时间 （程序中的参数）

n = 转子转速 （通过轴编码器测量获得）

T_r = L_r/R_r = 转子时间常数 （必须通过电机制造商获得）

θ = 转子磁通位置 （该模块的输出变量）

ω_b = 电气标称磁通转速 （从电机铭牌获得）

P_{pr} = 极对数 （从电机铭牌获得）

在稳态条件下，I_d 电流分量用于产生转子磁通。在瞬态变化时，I_d 测量值和转子磁通之间存在一个低通滤波关系。励磁电流 I_{mr} 是 I_d 的分量，用以产生转子磁通。在稳态条件下，I_d 等于 I_{mr}。公式 1 给出了 I_d 和 I_{mr} 之间的关系。该公式的精确性取决于转子电气时间常数是否准确。本质上，公式 1 在瞬态变化过程中对 I_d 的磁通产生分量进行校正。

I_{mr} 的计算值随后被用来计算转差频率，如公式 2 所示。转差频率是转子电气时间常数、I_q、I_{mr} 以及当前转子转速的函数。

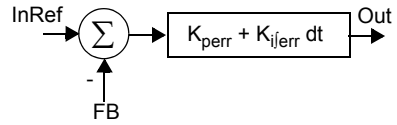
公式 3 是磁通估计器的最后一个公式。它根据公式 2 得出的转差频率以及前次磁通角计算值来计算新的磁通角。

由于公式 1 和 公式 2 给出了转差频率和定子电流的关系，因此电机磁通和转矩就已经是确定的。此外，这两个公式确保定子电流按照转子磁通进行正确地定向。如果可保持定子电流和转子磁通的正确定向，那么就可单独控制磁通和转矩。I_d 电流分量控制转子磁通，而 I_q 电流分量控制电机转矩。这就是间接矢量控制的主要原理。

PI 控制

使用三个 PI 环分别控制相互影响的三个变量。转子转速、转子磁通以及转子转矩皆通过单独的 PI 模块来控制。PI 控制实现采用常规方法，并包含了一个 (Kc*Excess) 项以抑制积分饱和，如图 6 所示。

图 6： PI 控制




```
Err = InRef - FB;
U = Sum + Kp*Err;
If (U > Outmax);
       Out = Outmax;
else if (U < Outmin)
       Out = Outmin;
else
       Out = U;
Excess = U - Out;
Sum = Sum + (Ki*Err)-(Kc*Excess);
```

PID 控制器背景知识

全面论述比例 – 积分 – 微分 （PID）控制器不属于本应用笔记的范围，但本节将对 PID 的基本工作原理进行介绍。

PID 控制器对闭环控制环中的误差信号进行响应，并对控制量进行调节，以获得期望的系统响应。被控参数可为任何可测系统量，比如转速、转矩或磁通。PID 控制器的优点在于，可通过对一个或多个增益值进行调节以及观测系统响应变化的方法，以实验为根据进行调节。

数字 PID 控制器周期性地执行控制操作。假定控制器的执行频率足够高，即可实现对系统的正确控制。误差信号是通过将被控参数的期望设定值减去该参数的实际测量值来获得的。误差的符号表明控制输入所需的变化方向。

控制器的比例 （P）项由误差信号乘以一个 P 增益因子形成，使 PID 控制器产生的控制响应为误差幅值的函数。当误差信号增大时，控制器的 P 项将变大以提供更大的校正量。

随着时间的消逝， P 项有利于减小系统总误差。然而，P 项的影响将随着误差接近于零而减小。在大多数系统中，被控参数的误差会非常接近于零，但并不会收敛。因此总会存在一个微小的静态误差。

PID 控制器的积分项（I）用来消除小的静态误差。I 项对全部误差信号进行连续积分。因此，小的静态误差随时间累计为一个较大的误差值。该累计误差信号乘以一个 I 增益因子即成为 PID 控制器的 I 输出项。

PID 控制器的微分 （D）项用来增强控制器的速度，以及对误差信号变化率的响应速度。D 项输入是通过计算前次误差值与当前误差值的差来获得的。该差值乘以一个 D 增益因子即成为 PID 控制器的 D 输出项 。系统误差变化的越快，控制器的 D 项 将产生更大的控制输出。

并非所有的 PID 控制器都实现 D 或 I 项（不常用）。例如，本文的应用中没有使用 D 项，这是因为电机速度变化的响应时间相对较慢。因此， D 项可能导致 PWM 占空比的过度变化，可能影响算法的运行并产生过电流。

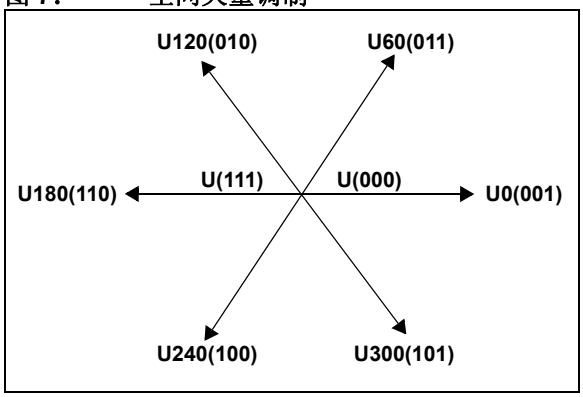
空间矢量调制

矢量控制过程的最后一步是产生三相电机电压的脉宽调制信号。通过使用空间矢量调制（SVM）技术，每相脉冲宽度的产生过程可简化为几个公式。本应用的 SVM 子程序中包含了 Clarke 反变换，进一步简化了计算。

三相逆变器的每相输出可为两种状态之一，即上管导通逆变器输出连接到直流电源正极性端，或下管导通逆变器输出连接到直流电源负极性端，这样使得三相逆变器输出共存在 2^3 = 8 种可能的状态（见表 1）。

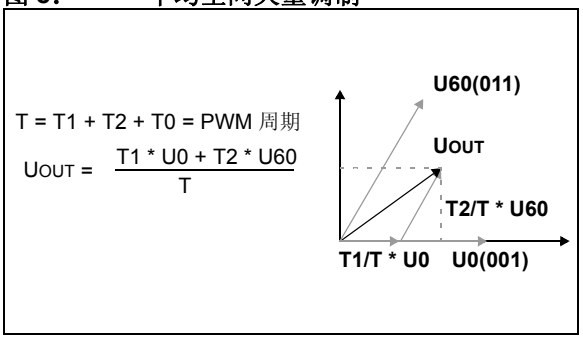
其中，三相输出全部连接到正极性端或负极性端的两种状态被视为零矢量，因为此时电机任一相绕组两端不存在线电压。这两种状态在 SVM 星型图中被绘作原点。剩余的六种状态表征为每一状态间旋转间隔为 60 度电角度的基本矢量，如图 7 所示。

图 7：　　空间矢量调制



在空间矢量调制过程中允许采用相邻的两个基本矢量的组合来表征任意的空间电压矢量。在图 8 中，UOUT 为期望的空间电压矢量。该矢量位于 U60 和 U0 之间的区间内。如果在给定的 PWM 周期 T 内，基本矢量 U0 的输出时间为 T1/T 而 U60 的输出时间为 T2/T，则整个周期内的平均电压矢量将为 UOUT。

图 8：　　平均空间矢量调制



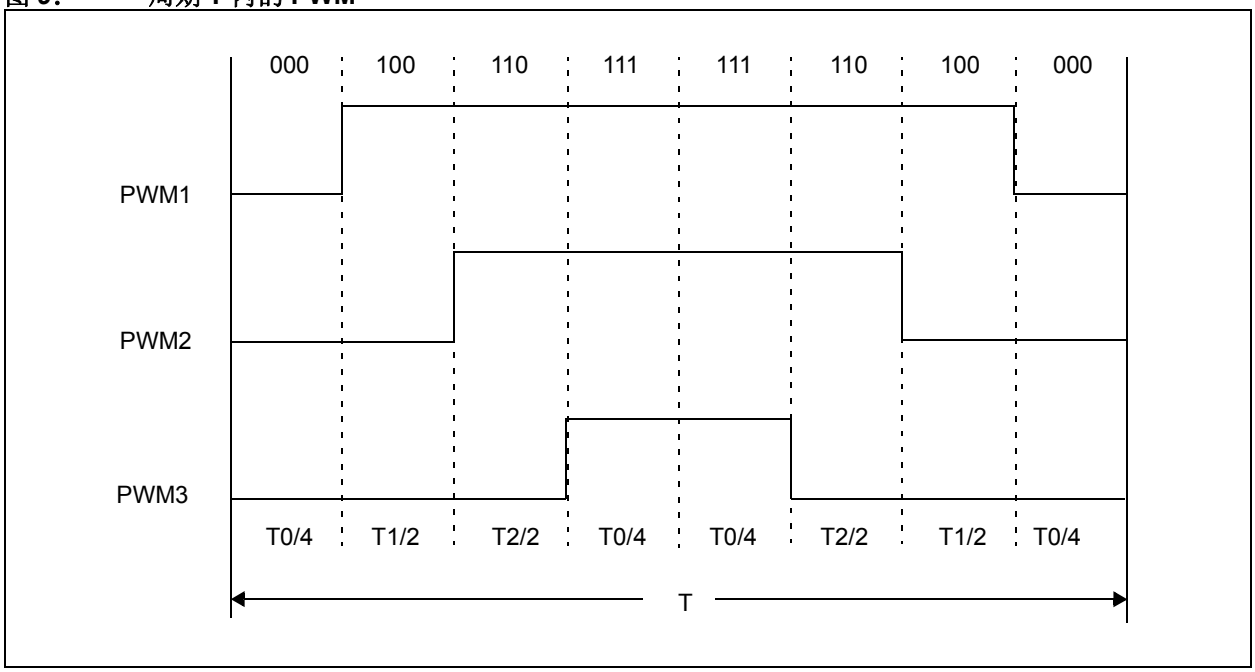
通过使用修正后的 Clarke 反变换，无需多余计算即可获得 T1 和 T2 的具体数值。通过将 v_α 和 v_β 进行颠倒，可以产生一个参考轴，该轴相对于 SVM 星型偏移了 30 度。因此在六个区间的每个区间中，一个轴与该区间正好反向，其他两个轴相互对称作为该区间的边界。沿着这两个边界轴的矢量分量分别等于 T1 和 T2。计算的具体细节请参见 **“附录 B 源代码”**中的 `CalcRef.s` 和 `SVGen.s` 文件。

从图 9 中可见，在 PWM 周期 T 内，矢量 T1 的输出时间为 T1/T，而矢量 T2 的输出时间为 T2/T。在调制周期的剩余时间中则输出零矢量。dsPIC® DSC 器件配置为中心对齐 PWM，使 PWM 以周期的中心对称。该配置将在每一个周期内产生两个线 – 线脉冲。有效开关频率加倍，纹波电流减小，同时并未增加功率器件的开关损耗。

表 1：　　空间矢量调制逆变器状态

C	B	A	V_{ab}	V_{bc}	V_{ca}	V_{ds}	V_{qs}	矢量
0	0	0	0	0	0	0	0	U(000)
0	0	1	VDC	0	-VDC	2/3VDC	0	U_0
0	1	1	0	VDC	-VDC	VDC/3	VDC/3	U_{60}
0	1	0	-VDC	VDC	0	-VDC/3	VDC/3	U_{120}
1	1	0	-VDC	0	VDC	-2VDC/3	0	U_{180}
1	0	0	0	-VDC	VDC	-VDC/3	- VDC/3	U_{240}
1	0	1	VDC	-VDC	0	VDC/3	- VDC/3	U_{300}
1	1	1	0	0	0	0	0	U(111)

图 9：　　周期 T 内的 PWM



代码说明

矢量控制源代码是在 MPLAB® IDE 环境中使用 Microchip MPLAB C30 软件工具套件开发的。主应用程序采用 C 语言编写，而所有主要的矢量控制函数采用汇编语言编写，并优化了执行速度。

约定

函数说明包含在与每一个源文件的头文件中。函数的等效 C 代码也包含在头文件中供参考。在优化的汇编代码中，使用C代码行作为注释，这样有助于理解程序流程。

在每个函数的开头，相关变量被传送到 DSP 和数学指令使用的工作寄存器（W）中。在函数结尾，这些变量被传送回各自的寄存器地址中。大多数变量被归组到相关参数的结构中以使C或汇编代码对这些变量的访问效率更高。

汇编模块中的任何一个W寄存器都已赋予一个说明性的名称以表明在计算过程中寄存器保存的是什么值。对 W 寄存器进行重命名可增强代码的可读性，并能避免寄存器使用冲突。

变量定义和定标

大多数变量采用 1.15 小数格式进行存储，这种方式为 dsPIC DSC 器件固有数学模式的一种。有符号定点整数表示如下：

- MSB 为符号位
- 范围为 -1 至 +.9999
- 0x8000 = -1
- 0000 = 0
- 0x7FFF = .9999

使用标幺制 （PU）对所有值进行归一化处理。

$V_{PU} = V_{ACT}/V_B$

然后定标，这样基值 = .125

因此取值范围可为基值的 8 倍。

V_B = 230V，V_{ACT} =120V，V_{PU} = 120/230 =.5PU，

定标为 $\rightarrow V_B$ = .125 = 0x0FFF (1.15)

120V = .5 * .125 = 0x07FF (1.15)

源文件说明

本节将对包含在每个源文件中包含的函数进行说明。

注：	如果您在阅读该应用笔记的电子版，可通过点击下面的文件名浏览至 **“附录 B 源代码”**中的代码。

UserParms.h

所有用户定义的参数都包含在 UserParms.h 文件中。这些参数包括电机数据和控制环调节参数值。这些参数的更多信息在本文档的**软件调节**一节中介绍。

ACIM.c

ACIM.c 文件为本应用的主要源代码文件。该文件包括主程序循环以及所有中断服务程序。该文件调用所有硬件和变量初始化子程序。

为实现高性能闭环控制，在每个 PWM 周期内必须执行整个矢量控制环。这在 AD 转换器的中断服务程序中进行。PWM 时基用来触发 AD 转换。当 AD 转换结束时，将产生中断。

当不在中断服务程序中时，将运行主程序循环处理用户界面。中断服务程序中利用一个计数变量进行计数，来周期性运行用户界面代码。根据设定，用户界面代码每 50 ms 执行一次。可通过修改 UserParms.h 文件更改该参数。

通过去除 UserParms.h 文件中 #define DIAGNOSTICS 语句的注释符将使能软件诊断模式。诊断模式将使能输出比较通道 OC7 和 OC8 作为 PWM 输出。这些输出可采用简单的 RC 滤波器进行滤波，并用作 D/A 转换器来观测软件变量随时间的变化。诊断输出简化了 PI 控制环的调节。本文档的**软件调节**一节中提供了有关诊断输出的更多信息。

Encoder.c

该文件包含 InitEncoderScaling() 函数，该函数用于计算通过光编码器测量的机械转角和机械转速的定标值。

InitCurModel.c

该文件包含 InitCurModScaling() 函数，该函数被 ACIM.c文件中的 setup函数调用。该函数用来计算定点换算因子，该因子数用于采用浮点数表示的电流模型公式中。电流模型换算因子是转子时间常数、矢量计算循环周期、电机极数以及最大电机转速 （单位是转 / 秒）的函数。

CalcRef.s

该文件包含了 CalcRefVec() 函数，该函数根据 v_α 和 v_β 来计算换算后的三相电压输出矢量 （V_{r1}、V_{r2} 和 V_{r3}）。该函数实现了 Clarke 反变换功能，即将 2 轴静止坐标系中的电压矢量变换到 三相 PWM 可以使用的 3 轴坐标系中。该方法是经过修正的 Clarke 反变换，与常规的 Clarke 反变换相比，将 v_α 和 v_β 进行了交换。必须使用修正方法来确保产生正确的电压矢量相位。

CalcVel.s

该文件包含三个函数，即 InitCalcVel()、CalcVelIrp() 和 CalcVel()。这些函数用来确定电机转速。InitCalcVel() 函数对与转速计算相关的关键变量进行初始化。

CalcVelIrp() 函数在每一个矢量控制中断周期内被调用。中断间隔时间 VelPeriod，必须小于最大转速时 1/2 转所需的最小时间。

该函数对指定中断周期数的变化进行累计，随后将累计值拷贝到 iDeltaCnt 变量，以供 CalcVel() 使用来计算转速。累计值设置为 0，并开始新的累计操作。

CalcVel() 函数仅在获得新的转速信息时才被调用。若采用缺省的程序设定值，CalcVel() 函数每隔 30 个中断周期调用一次。对于中断周期为 50 µs 的情况，则每隔 1.5 ms 获得一次新的转速信息。每次获得新的转速信息后，将运行转速控制环。

ClarkePark.s

该文件包含 ClarkePark() 函数并计算 Clarke 和 Park 变换。该函数使用磁通位置角的正弦和余弦值来计算 I_d 和 I_q 电流值。该函数也适用于整数定标以及 1.15 定标的数据格式。

CurModel.s

该文件包含 CurModel() 和 InitCurModel() 函数。CurModel()函数将计算转子电流模型公式以确定新的转子磁通角，转子磁通角是转子转速和变换后定子电流分量的函数。 InitCurModel() 函数用来对与 CurModel() 子程序相关的变量清零。

图 10：　矢量控制中断服务程序

```
void __attribute__((__interrupt__)) _ADCInterrupt(void)
{
IFS0bits.ADIF = 0;

// 递增控制显示和按钮功能执行的计数变量
iDispLoopCnt++;

// 对上次中断后的编码器计数进行累计
CalcVelIrp();

if( uGF.bit.RunMotor )
      {
      // 置位用于诊断功能的 LED1
      pinLED1 = 1;
      // 根据累计的编码器计数计算转速
      CalcVel();
      // 计算 qIa,qIb
      MeasCompCurr();
      // 根据 qSin,qCos,qIa,qIb 计算 qId,qIq
      ClarkePark();
      // 计算 PI 控制环参数
      DoControl();
      // 根据 qAngle 计算 qSin,qCos
      SinCos();
      // 根据 qSin,qCos,qVd,qVq 计算 qValpha, qVbeta
      InvPark();
      // 根据 qValpha, qVbeta 计算 Vr1,Vr2,Vr3
      CalcRefVec();
      // 根据 Vr1,Vr2,Vr3 计算和设定 PWM 占空比
      CalcSVGen();
      // 清零用于诊断功能的 LED1
      pinLED1 = 0;
      }
}
```

FdWeak.s

FdWeak.s 文件包含用于弱磁控制的函数。本应用笔记中提供的应用代码没有实现弱磁功能。采用弱磁控制可使电机运行速度超过额定转速。当电机转速超过额定转速进行高速运行时，当频率增加时，施加在电机绕组上的电压保持不变。

UserParms.h 文件中定义了一个弱磁控制常数。该值是由电机的 V/Hz 常数得来的。该应用中使用电机的工作电压为 230 VAC 且设计为输入频率 60 Hz。根据这些参数，可确定 V/Hz 常数为 230/60 = 3.83。根据电机的V/Hz常数以及本应用中A/D 反馈值的绝对换算方式，UserParms.h文件中定义经验值3750作为弱磁控制常数。

当电机运行在额定转速和电压范围内时，I_d 控制环中的给定值保持恒定。UserParms.h 文件中定义的弱磁控制常数作为控制环的给定值。在电机正常运行范围内，转子磁通保持恒定。

如果实现弱磁控制，电机的 V/Hz 比不再能保持常数关系，I_d 控制环的给定应被线性减小。例如，假定以 115 VAC 电源来驱动一台230 VAC电机。由于该电机设计为运行在 230 VAC 和 60 Hz，因此若采用 115 VAC 电源进行供电，电机在 30 Hz 时将不再能保持 V/Hz 比为常数。在高于 30 Hz 的情况时，I_d 控制环的给定值应作为频率的函数被线性减小。

通过监视逆变器的直流母线电压，可确定 ACIM 应用中电机 V/Hz 比不再能保持常数关系时的驱动频率。

当工作在需要弱磁运行的区域内，I_d 和 I_q 控制环将出现饱和，这将有效限制电机磁通。采用弱磁控制允许矢量控制算法限制其输出，而不会使控制环出现饱和。这是弱磁控制的主要优点之一。只要保持闭环控制有效，电机运行范围就能被扩展。

在本应用中，用户可通过更改 UserParms.h 文件中定义的给定值来进行弱磁运行实验。通过降低该值，可限制施加在电机绕组上的电压。

InvPark.s

该文件包含 InvPark() 函数，该函数对由内部 PI 电流控制环产生的电压矢量值 V_d 和 V_q 进行处理。InvPark() 函数将同步旋转的电压矢量变换到静止的参考坐标系中。该函数输出 v_α 和 v_β 值。使用前次根据转子电流模型公式计算的新转子磁通角的正弦和余弦值可实现旋转功能。

该函数也适用于整数定标以及 1.15 定标的数据格式。

MeasCur.s

该文件包含两个函数，即 MeasCompCurr() 和 InitMeasCompCurr()。MeasCompCurr()函数负责读取 ADC S/H 通道 CH1 和 CH2 的数据，并使用 qKa、qKb 将其换算为有符号小数值，并将结果保存在 Park-Parm的qIa和qIb中。A/D 偏移量的连续平均值将被保持并在换算前从 ADC 值中减去。

InitMeasCompCurr() 函数用来在启动时对 A/D 偏移值进行初始化。

与这些函数相关的换算和偏移变量存放在 MeasCurrParm 数据结构中，该数据结构在 MeasCur.s 文件中定义。

OpenLoop.s

该文件包含OpenLoop()函数，该函数负责在应用开环运行时计算新的转子磁通角。该函数计算期望运行速度时的转子磁通角变化量。随后将转子磁通角变化量加上旧的磁通角以设定新的电压矢量角。

PI.s

该文件包含 CalcPI() 函数。该函数负责执行 PI 控制器。CalcPI() 函数输入参数为一个指向某个结构的指针，该结构中包含 PI 系数、输入和给定信号、输出极限值以及 PI 控制器输出值。

ReadADC0.s

该文件包含 ReadADC0() 和 ReadSignedADC0() 函数。这些函数负责读取从 ADC 的采样 / 保持通道 0 获取的数据，对其进行换算并存储结果。

ReadSignedADC0() 函数目前用来读取演示板上电位器的速度给定值。如果速度给定通过其他方式获得，则应用中将不需要这些函数。

SVGen.s

该文件包含了 CalcSVGen() 函数。该函数负责计算最终的 PWM 值，该值是三相电压矢量的函数。

Trig.s

该文件包含 SinCos() 函数，该函数在 128 字查找表的基础上利用线性插值法计算指定角度的正弦和余弦值。

为节省数据存储空间，128 字的正弦波查找表存放在程序存储器，并使用 dsPIC DSC 架构的程序空间可视性（Program Space Visibility，PSV）特性对其进行访问。PSV功能允许将程序存储器的一部分映射至数据存储空间，因此可如同在 RAM 中一样对常量数据进行访问。

该子程序也适用于整数定标以及 1.15 定标的数据格式。对于整数定标，角度经过定标后存在对应关系：0 ≤ 角度 < 2Π 对应于 0 ≤ 角度 < 0xFFFF。该子程序返回角度的正弦和余弦计算结果将作为函数的返回值，换算后的值为 -32769 至 +32767 （即 0x8000 至 0x7FFF）。

对于 1.15 定标格式，角度经过换算后存在对应关系：-Π ≤ 角度 < Π 对应于 -1 至 +0.9999（即 0x8000 ≤ 角度 < 0x7FFF）。该子程序返回角度的正弦和余弦计算结果，定标后的值为 -1 至 +0.9999（即 0x8000 至 0x7FFF）。

演示硬件

矢量控制应用程序可在 dsPICDEM™ MC1 电机控制开发系统上运行。需要以下硬件：

- Microchip dsPICDEM MC1 电机控制开发板
- 9 V 直流电源
- Microchip dsPICDEM MC1H 三相高电压功率模块
- 用于功率模块的电源线
- 安装有轴编码器的三相交流感应电机

注：	应使用至少每转 250 线的编码器。上限为每转 32,768 线。

建议使用的电机和编码器

在开发该应用和进行软件调节参数选择时，使用了以下电机和编码器组合。

- Leeson Cat# 102684 电机， 1/3 HP， 3450 RPM
- U.S. Digital 编码器， E3-500-500-IHT 型

用户可从 Microchip 或电机分销商处获得 Leeson 电机。编码器可在 U.S. Digital 网站 www.usdigital.com 上订购。该型号编码器附带一个安装对齐工具以及自粘附编码器底盘。如图 12 所示，编码器可直接安装在电机前端。除 U.S. Digital 生产的编码器外，用户亦可选择其他类似的精度为 500 线的编码器。

图 11： 使用 dsPICDEM 电机控制开发系统的硬件配置

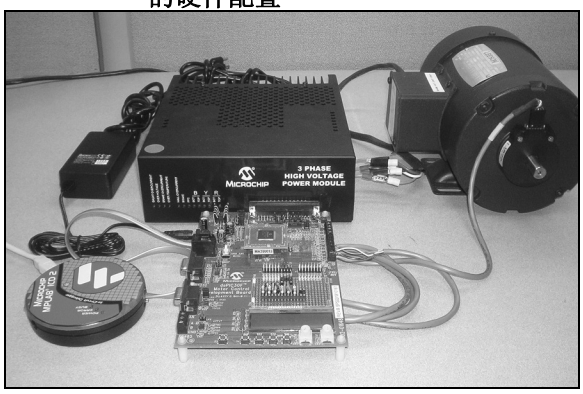
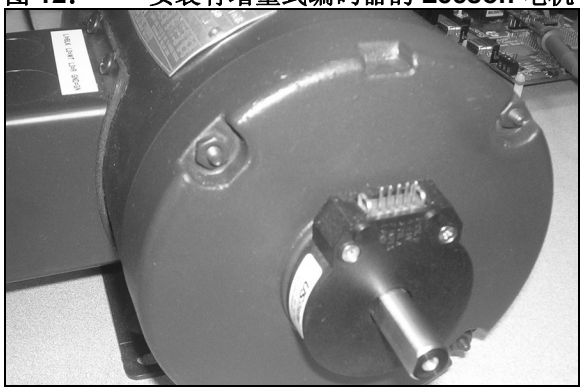


图 12： 安装有增量式编码器的 Leeson 电机



如果选择其他的电机……

如果选用其他电机，用户应通过实验调试选择合适的闭环调节参数来通过控制算法获得良好的响应。至少需要确定转子的电气时间常数 （单位为秒）。用户可从电机制造商处获取该信息。如果转子时间常数选择不正确，尽管应用系统仍可运行，但不能实现理想的瞬态响应。

如果选用上述提及的 Leeson 电机和 500 线编码器，用户无需调整软件调节参数即可使演示系统正确运行。

相电流反馈

矢量控制应用中需要知道电机三相电流的值。本应用设计为采用 dsPICDEM MC1H 功率模块上的隔离霍尔效应电流传感器进行相电流检测。这些传感器为有源器件，可输出 200 kHz 带宽， 0-5V 的反馈信号。本应用中使用霍尔效应器件的原因是因其使用方便和安全。来自霍尔电流传感器的信号可直接连接到 dsPIC DSC 器件的 A/D 转换器。在用户的最终应用中，可选择使用安装在三相逆变器每一桥臂的电阻来进行电流检测。使用电阻的方法提供了一种成本更为低廉的电流检测方案。

电机接线配置

大多数三相 ACIM，包括 Leeson 电机，可采用 208V 或 460V 供电。如果用户使用 dsPICDEM MC1 系统来驱动电机，则应使用 208V 供电。

矢量控制应用中将不对 DC 母线电压进行调节。然而，额定电压为 208V 的电机采用 120V 供电时仍可正常运行，只是转速和转矩输出受到了限制。

跳线配置

三相高电压功率模块上的所有跳线配置可保持为缺省状态。如果用户已打开功率模块的盖进行跳线配置更改，请参考功率模块的用户指南，了解缺省的跳线配置。

对于电机控制开发板，应使用以下跳线配置：

- 隔离的霍尔效应电流传感器用来测量电机相电流。应确保 LK1 和 LK2 （紧邻 5V 稳压器）放置在引脚 1 和 2 上。
- 当运行演示代码时，开关 S2（紧邻 ICD 连接插座）应被设置在 “模拟”位置以将相电流反馈信号连接到 dsPIC DSC 器件的模拟输入引脚 （当进行器件编程时， S2 应置于 “ICD”位置）。
- 所有其他跳线应保持其缺省配置。

外部连接

- 将电机控制开发板直接插入功率模块上的 37 引脚连接插座。
- 确保开发板已安装 dsPIC30F6010 器件。
- 将电机端子连接到功率模块中标有 R， Y， B 的输出端子。将相 1、相 2 和相 3 分别连接到 R、 Y 和 B。

将编码器端子接至 MCDB 上的正交编码器接口（Quadrature Encoder Interface， QEI）端子排，应确保 MCDB 上的引脚名称与编码器上的信号名称相符。最后将 9V 电源接至 MCDB 上的 J2。

端口使用

表 2 介绍了本应用中 dsPIC DSC 器件端口的使用情况。该信息可帮助用户进行硬件定义。矢量控制应用所需的 I/O 引脚以粗体显示。本应用中还使用了其他引脚，如用于 LCD 接口的引脚，这些引脚并不是电机控制功能所必需的。在用户的最终设计中可选择使用这些 I/O 连接。

表 2： dsPIC® DSC 器件端口使用情况一览

引脚	功能	类型	在本应用中的使用
端口 A			
RA9	VREF-	O	LED1 和 D6 （高电平有效）
RA10	VREF+	O	LED2 和 D7 （高电平有效）
RA14	INT3	O	LED3 和 D8 （高电平有效）
RA15	INT4	O	LED4 和 D9 （高电平有效）
端口 B			
RB0	**PGD/EMUD/AN0/CN2**	**AI**	**相 1 电流 / 器件编程引脚**
RB1	**PGC/EMUC/AN1/CN3**	**AI**	**相 2 电流 / 器件编程引脚**
RB2	AN2/SS1/LVDIN/CN4	AI	本应用中未用
RB3	**AN3/INDX/CN5**	**I**	**QEI 索引**
RB4	**AN4/QEA/CN6**	**I**	**QEI A**
RB5	**AN5/QEB/CN7**	**I**	**QEI B**
RB6	AN6/OCFA	AI	本应用中未用
RB7	AN7	AI	电位器 （VR1）
RB8	AN8	AI	本应用中未用
RB9	AN9	AI	本应用中未用
RB10	AN10	AI	本应用中未用
RB11	AN11	AI	本应用中未用
RB12	AN12	AI	本应用中未用
RB13	AN13	AI	本应用中未用
RB14	AN14	AI	本应用中未用
RB15	AN15/OCFB/CN12	O	本应用中未用
端口 C			
RC1	T2CK	O	LCD R/$\overline{W}$
RC3	T4CK	O	LCD RS
RC13	EMUD1/SOSC2/CN1	—	备用的 ICD2 通信引脚
RC14	EMUC1/SOSC1/T1CK/CN0	—	备用的 ICD2 通信引脚
RC15	OSC2/CLKO	—	—
Port D			
RD0	EMUC2/OC1	I/O	LCD D0
RD1	EMUD2/OC2	I/O	LCD D1
RD2	OC3	I/O	LCD D2
RD3	OC4	I/O	LCD D3
RD4	OC5/CN13	O	本应用中未用
RD5	OC6/CN14	O	本应用中未用
RD6	OC7/CN15	O	用于诊断输出的 PWM
RD7	OC8/CN16/UPDN	O	用于诊断输出的 PWM
RD8	IC1	I	本应用中未用
RD9	IC2	I	本应用中未用
RD10	IC3	I	本应用中未用
RD11	IC4	O	演示板 PWM 输出缓冲器使能 （低电平有效）
RD12	IC5	—	本应用中未用
RD13	IC6/CN19	O	LCD E
RD14	IC7/CN20	—	本应用中未用

表 2： dsPIC® DSC 器件端口使用情况一览 （续）

引脚	功能	类型	在本应用中的使用
RD15	IC8/CN21	—	本应用中未用
端口 E			
RE0	**PWM1L**	**O**	**相 1 L**
RE1	**PWM1H**	**O**	**相 1 H**
RE2	**PWM2L**	**O**	**相 2 L**
RE3	**PWM2H**	**O**	**相 2 H**
RE4	**PWM3L**	**O**	**相 3 L**
RE5	**PWM3H**	**O**	**相 3 H**
RE6	PWM4L	O	本应用中未用
RE7	PWM4H	O	本应用中未用
RE8	FLTA/INT1	I	功率模块故障信号 （低电平有效）
RE9	FLTB/INT2	O	功率模块故障复位信号 （高电平有效）
Port F			
RF0	C1RX	I	本应用中未用
RF1	C1TX	O	本应用中未用
RF2	U1RX	I	本应用中未用
RF3	U1TX	O	本应用中未用
RF4	U2RX/CN17	I	本应用中未用
RF5	U2TX/CN18	O	本应用中未用
RF6	EMUC3/SCK1/INT0	I	本应用中未用
RF7	SDI1	I	本应用中未用
RF8	EMUD3/SDO1	O	本应用中未用
Port G			
RG0	C2RX	O	本应用中未用
RG1	C2TX	O	本应用中未用
RG2	SCL	I/O	本应用中未用
RG3	SDA	I/O	本应用中未用
RG6	SCK2/CN8	I	按钮 1 （S4）（低电平有效）
RG7	SDI2/CN9	I	按钮 2 （S5）（低电平有效）
RG8	SDO2/CN10	I	按钮 3 （S6）（低电平有效）
RG9	SS2/CN11	I	按钮 4 （S7）（低电平有效）

项目建立和器件编程

建议使用 MPLAB IDE v6.50 或更新的版本来建立项目以及对器件进行编程。用户可选用以下两种方法之一将源代码烧写到 dsPIC DSC 器件中：

1. 将应用源代码提供的预编译 hex 文件导入 MPLAB IDE 并对器件进行编程，或
2. 可在 MPLAB IDE 中建立一个新的项目，对源代码进行编译并对器件进行编程。

导入 HEX 文件

用户如果没有安装 MPLAB C30 编译器，将不能对应用程序进行编译。此时可采用提供的 hex 文件。用户使用的硬件配置应与本文中 **“演示硬件”** 一节中介绍的相同。

建立新项目

MPLAB C30 v. 1.20 编译器用于对应用源代码进行编译。欲对源代码进行编译，将所有汇编文件（.s 扩展名）以及 C 文件将入新的项目中。将器件链接描述文件添加到项目文件中。假定 C30 编译器已安装到缺省路径，应使用链接描述文件 p30f6010.gld （该文件位于 c:\pic30_tools\support\gld 目录中）。用户还应为编译选项设定汇编器和 C 编译器路径。

这些路径为 c:\pic30_tools\support\inc½Õ c:\pic30_tools\support\h.

器件工作频率

本文所提供源代码中设定使用 7.37 MHz 的晶振和器件振荡器的 8XPLL 选项，所以器件工作速度为 14.76MIPS。如果使用不同频率的晶振，需对 UserParms.h 文件中的一些设定值进行修改。有关 UserParms.h 中设定值调整的更多信息，可参见本文档中的**软件调节**一节。如果使用不同的振荡器选项，也需修改 config.s 文件。

软件操作

通过演示程序所提供的一些基本功能，用户可对系统对所要求转速的 2:1 阶跃变化的响应性能进行评估。

本应用提供了两种控制模式，允许系统全闭环工作或工作在传统的开环恒压 / 频比模式。

通过四个按钮来控制不同的工作模式。

速度给定值由电位器 VR2 提供，该电位器采用双向控制，零速度给定对应电位器的中间位置。

按钮

按钮 1 （S4）

按按钮 1 可切换系统的运行状态。如果按钮断开，系统将处于运行状态；如果按钮闭合，则系统将停止运行。该按钮也可用来通过重新起动电机来清除任何硬件故障。

按钮 2 （S5）

按钮 2 用于切换开环模式和闭环模式。缺省情况下，系统以开环模式启动。

按钮 3 （S6）

按钮 3 用于控制是否将将速度给定除以一个 2 的因子。该按钮在半转速模式下使用。

按钮 4 （S7）

在演示代码中，按钮 4 不具有任何功能，但仍提供了该按钮的处理代码。因此用户可在其中添加自己的功能。

LED

LED 1 （D6）

当系统运行时，LED 1 点亮。该信号由中断服务程序来进行调节。中断服务程序的长度可以通过观察该信号为高电平的时间来测量。

LED 2 （D7）

当系统处于闭环模式时点亮。

LED 3 （D8）

当转速达到全速时点亮，当转速为半速时关闭。

LED 4 （D9）

本应用中未使用。

FDW/REV （D5）

连接到 D5 的 RD7 端口引脚用作诊断功能的输出比较通道 （OC8）。因此， D5 的状态在本应用中没有任何意义。

如果未使用诊断输出，可通过 dsPIC DSC 器件上的 QEI 对 D5 进行直接驱动。通过 QEICON 寄存器中的控制位可使能 RD7 作为方向状态输出引脚。当该功能使能时，D5 将被点亮以表示电机正向旋转。

LCD

LCD 是用户反馈的主要途径。当程序处于待机模式时，显示提醒用户按下 S4 按钮起动电机。当程序运行时，将显示转速 （RPM）。在主循环中对 LCD 的状态进行更新，而且可方便地增加其他显示参数。

故障排除

如果电机不能运行于开环模式：

- 检查功率模块故障指示灯。如有必要，复位 dsPIC DSC 器件以清除故障。
- 检查功率模块是否处于上电状态。检查模块中的母线电压 LED。

电机运行于开环模式，但不能运行于闭环模式。

- 确保 S2 处于 “模拟”位置。
- 确保对 LK1 和 LK2 配置正确。
- 检查编码器的接线连接是否正确。
- 编码器信号可能相对于电机接线和旋转方向反向了。如果怀疑是这个问题，将编码器接线中的 A 和 B 信号连接交换。编码器的接线方式还取决于编码器安装在电机的前端还是后端。

软件调节

诊断模式

诊断模式可使用户使用多余的输出比较 （OC） 通道 OC7 和 OC8 来观察内部程序变量。在诊断模式下，这些通道用作 PWM 输出，然后使用简单的 RC 滤波电路对其进行处理，用作简单的 DAC 输出功能，在示波器上显示内部变量随时间的变化历史。

dsPIC30F6010器件的OC7和OC8通道位于引脚RD6和RD7。这两个引脚的外接线位于 dsPICDEM MC1 电机控制开发板的插头 J7。

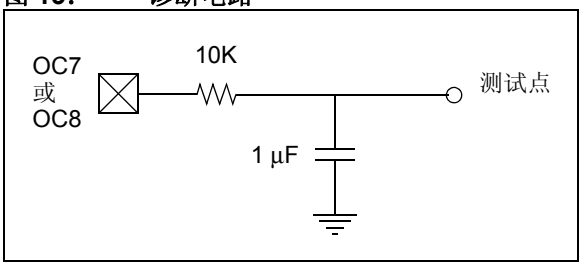
使能诊断模式

只需去除 `UserParms.h` 文件中 `#define DIAGNOSTICS` 语句两端的注释，并重新编译应用程序，即可使能诊断输出功能。

诊断模式的硬件配置

为使用诊断功能，用户需在开发板中接入两个 RC 低通滤波电路。RC 滤波器应接至器件的 RD6 和 RD7 引脚。在大多数情况下，使用一个 10k 欧姆电阻和 一个 1 µF 电容即可。如果没有上述取值的元件，亦可选择数值相近的元件。

图 13： 诊断电路



PID 增益调节

PID 控制器的 P 增益设定了系统的总响应能力。在初次对控制器进行参数整定时， I 和 D 增益应设置为 0。随后增大 P 增益直至系统对给定值变化的响应良好，不存在过大的超调或振荡。使用较小的 P 增益将“松散”地控制系统，而较大的值则“紧密”地控制系统。此时，系统响应将可能不收敛到给定值。

选取合适的 P 增益后，可缓慢增加 I 增益，消除系统误差。在大多数系统中，只需选取较小的 I 增益。应注意 I 增益的影响，如果取值过大，可能抵消 P 项的作用，使得整个控制系统的响应速度变慢，并导致系统在给定点附近振荡。如果出现振荡，通过减小 I 增益并增大 P 增益通常可解决此问题。

本应用包含了限制积分饱和的项，积分误差使输出参数饱和时会出现积分饱和现象。积分误差增加将不会对输出造成影响。如果允许对误差进行累计，当误差减小时，累计误差将会减小至导致输出饱和的限定值以下。Kc 系数用来对不需要的累计进行限制。在大多数情况下，该参数取值可与 Ki 相同。

所有三个控制器的输出参数都存在一个最大值。这些值可在 `UserParms.h` 文件中找到，且通过设定以避免在执行 `SVGen()` 子程序时出现饱和。

控制环的相关性

本应用中存在三个相互关联的 PI 控制环。外环控制电机转速。两个内部的控制环分别对变换后的电机电流 I_d 和 I_q 进行控制。如前所述， I_d 控制环负责控制磁通，而 I_q 值负责控制电机转矩。

转矩模式

在对三个控制环的参数进行调整时，将外环与内环分开进行调整是有益的。通过去除 `UserParms.h` 文件中的 `#define TORQUE_MODE`语句两端的注释，即可使电机运行在转矩模式。此时将忽略外部转速控制环，且电位器的给定值将直接作为 I_q 控制环的给定值。

推荐的控制环调节过程

如果需要对控制环进行调整，采用如前所述忽略转速控制环的方法是有帮助的。在大多数情况下，I_d 和 I_q 控制闭环的 PI 系数应设定为相同的值。一旦电机在转矩模式下有良好的转矩响应，即可使能并调整转速控制环。

示波器波形图示例

下面的示波器波形图显示了使用诊断输出功能和应用参数正确整定时的系统响应。

图 14 中显示了变换后的正交相电流 （I_q）和电机机械转速响应波形。假定应用参数正确整定，I_q 值与电机转矩成正比。该值包含在 `ParkParm` 数据结构中。电机机械转速包含在 `EncoderParm` 数据结构中。

图中显示了在控制环参数正确整定时的系统响应示例。可以看到，电机转速响应（下面的波形）几乎不存在超调或振铃。正交电流响应（上面的波形）非常迅速，仅在电机转速到达新给定速度时存在较小的超调或振铃。

图 14： Iq 与转速， 500RPM 至 1000RPM 阶跃响应

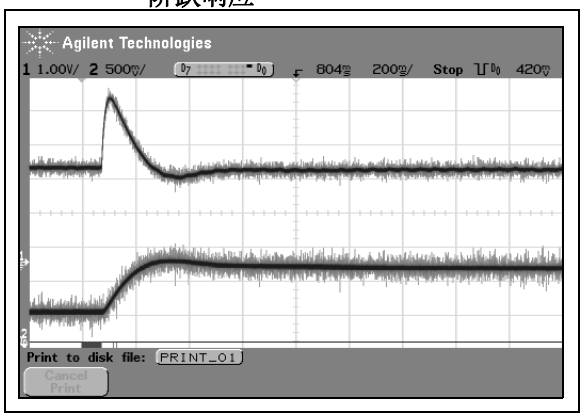


图 15 对比了采用正确调整的 PI 控制环参数和正确的电机时间常数，电机给定转速从 1000 RPM 至 2000 RPM 阶跃变化时，实际的交流相电流和电机转速响应。相电流直接通过电机控制开发系统上的两个相电流传感器之一测量。通过 EncoderParm 数据结构获得转速数据，并发送至一个 PWM 诊断输出，显示在示波器上。在该示波器图中，可观察到转速快速上升至新的给定点，几乎没有超调和振铃。此外，速度变化时相电流的幅值并无剧烈变化。

图 15： 相电流与转速， 1000 至 2000RPM 阶跃响应， TR = 0.078 秒

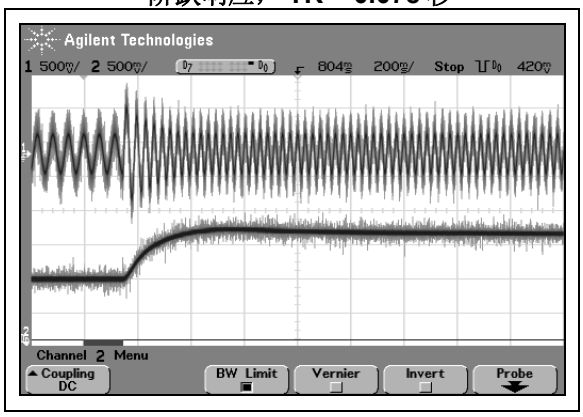


图 16 显示了与图 15 相同的相电流和转速数据。此时，在开环模式下，给定转速从 1000 RPM 至 2000 RPM 阶跃变化。在开环模式下，实现速度变化需要更大的电流幅值和更长的时间。图 15 和图 16 的对比清楚显示了矢量控制的优点。在闭环模式下速度变化所需的电流更小。

图 16： 相电流与转速，1000 至 2000RPM 阶跃响应，开环

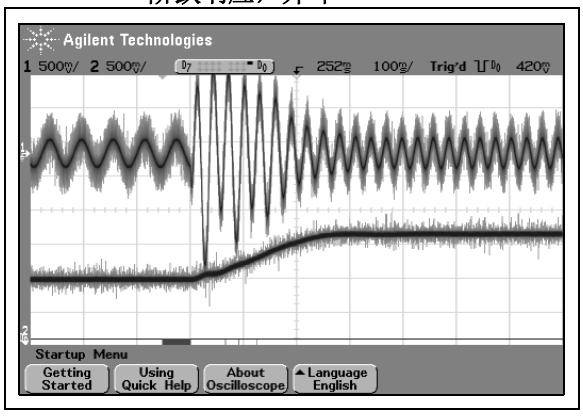
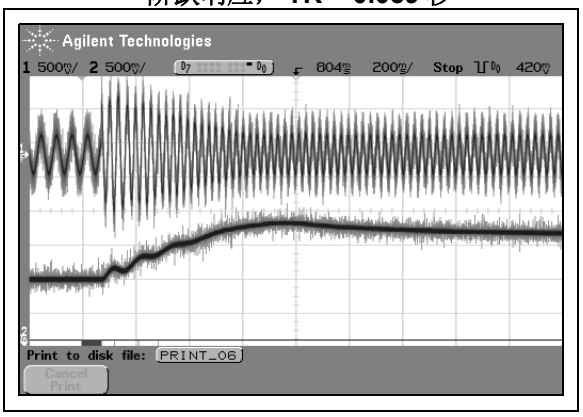


图17显示了使用不正确转子时间常数值时的阶跃响应。阶跃变化的实现需要更大的电流和更多时间。

图 17： 相电流与转速， 1000 至 2000RPM 阶跃响应， TR = 0.039 秒



附录 A 参考文献

1. *Vector Control and Dynamics of AC Drives*, D. W. Novotny, T. A. Lipo, Oxford University Press, 2003, ISBN: 0 19 856439 2.
2. *Modern Power Electronics and AC Drives*, Bimal K. Bose, Pearson Education, 2001, ISBN: 0 13 016743 6.

附录 B 源代码

该附录包括以下所列文件的源代码。这些是与矢量控制算法有关的主要文件，列表中没有包括其他有关用户界面的文件。

如果您正在阅读该应用笔记的电子版，可通过点击下面所列文件名浏览至特定文件。

头文件

UserParms.h

C 文件

ACIM.c

Encoder.c

InitCurModel.c

汇编文件

CalcRef.s

CalcVel.s

ClarkePark.s

CurModel.s

FdWeak.s

InvPark.s

MeasCur.s

OpenLoop.s

PI.s

ReadADC0.s

SVGen.s

Trig.s

UserParms.h

```
//#define TORQUE_MODE
#define DIAGNOSTICS

//****************  振荡器 ************************************
#define dFoscExt            7372800                     // 外部晶振或时钟频率 (Hz)
#define dPLL                8                           // PLL 比率
#define dLoopTimeInSec      0.00005                     // PWM 周期 - 100 uS, 10 khz PWM
#define dDeadTimeSec        0.000002                    // 以秒为单位的死区时间
// Derived
#define dFosc               (dFoscExt*dPLL)             // 时钟频率 (Hz)
#define dFcy                (dFosc/4)                   // 指令频率 (Hz)
#define dTcy                (1.0/dFcy)                  // 指令周期 (s)
#define dDeadTime           (int)(dDeadTimeSec*dFcy)    // 以 dTcy 为单位的死区时间
#define dLoopInTcy          (dLoopTimeInSec/dTcy)       // 以 Tcy 为单位的基本循环周期
#define dDispLoopTime       0.100                       // 显示和按钮状态查询循环

//****************  电机参数 ********************************
#define diPoles             1                           // 极对数
#define diCntsPerRev        2000                        // 每转的编码器线数
#define diNomRPM            3600                        // 铭牌电机 RPM
#define dfRotorTmConst      0.078                       // 以秒为单位的转子时间常数，来自 mfgr

//**************** 测量 ************************************
#define diIrpPerCalc        30                          // 每次速度计算的 PWM 循环次数

//************** PI 系数 ************************************
#define dDqKp               0x2000                      // 4.0      (NKo = 4)
#define dDqKi               0x0100;                     // 0.125
#define dDqKc               0x0100;                     // 0.125
#define dDqOutMax           0x5A82;                     // 0.707    设定该值以避免饱和

#define dQqKp               0x2000;                     // 4.0      (NKo = 4)
#define dQqKi               0x0100;                     // 0.125
#define dQqKc               0x0100;                     // 0.125
#define dQqOutMax           0x5A82;                     // 0.707    设定该值以避免饱和

#define dQrefqKp            0x4000                      // 8.0      (NKo = 4)
#define dQrefqKi            0x0800                      // 1.0
#define dQrefqKc            0x0800                      // 1.0
#define dQrefqOutMax        0x3FFF                      // 0.4999   设定该值以避免饱和

//************** ADC 换算 ************************************
// 标定常数：               由校准或硬件设计确定。
#define dqK                 0x3FFF;                     // 等于 0.4999
#define dqKa                0x3FFF;                     // 等于 0.4999
#define dqKb                0x3FFF;                     // 等于 0.4999

//************** 弱磁 ***************************************
// 在恒转矩范围内的磁通给定值。
// 根据经验确定、给出额定压 / 频比
#define dqK1                3750;                       //
```

ACIM.c

```
/*****************************************************************************
*                                                                            *
*    作者 : John Theys/Dave Ross                                             *
*                                                                            *
*    文件名 :       ACIM.c                                                   *
*    日期 :         10/31/03                                                 *
*    文件版本 :     3.00                                                     *
*                                                                            *
*    使用工具 : MPLAB      -> 6.43                                           *
*               编译器     -> 1.20.00                                        *
*                                                                            *
*    链接文件 :    p30f6010.gld                                              *
*                                                                            *
*                                                                            *
*****************************************************************************
*10/31/03  2.00     发布     电机运行正常，但仍有些遗留的小问题
*
*12/19/03  2.01     完成结构，为所有用户定义变量创建 UserParms.h。
*
*02/12/043.00- 从项目中去除了不需要的文件。
*             - 将 iRPM 改为 int 以纠正浮点计算问题。
*             -CalcVel() 和转速控制环仅在 iIrpPerCalc 指定的数个循环周期后执行
*
*             - 增加了 iDispLoopCount 变量以安排显示和按钮子程序的执行时间
*             -trig.s 文件改为使用程序空间来存储正弦数据。
*             - 增加了 DiagnosticsOutput() 函数，该函数使用输出比较通道来输出控
*               制变量信息。
*             - 增加了 TORQUE_MODE 定义以忽略转速控制环。
*             - 关闭 curmodel.s 文件中的 SATDW 位。自动饱和功能阻止转差角计算正确
*               翻转返回。
*****************************************************************************
*   代码说明
*
*   该文件给出了使用 dsPIC30F 实现的三相交流感应电机矢量控制实例。
*   采用空间矢量调制作为控制策略。
*****************************************************************************/

/************************** 全局定义 ***********************/

#define INITIALIZE
#include "Motor.h"
#include "Parms.h"
#include "Encoder.h"
#include "SVGen.h"
#include "ReadADC.h"
#include "MeasCurr.h"
#include "CurModel.h"
#include "FdWeak.h"
#include "Control.h"
#include "PI.h"
#include "Park.h"
#include "OpenLoop.h"
#include "LCD.h"
#include "bin2dec.h"
#include "UserParms.h"

/********************** 全局定义结束 *******************/

unsigned short uWork;
short iCntsPerRev;
short iDeltaPos;
```

```
union   {
        struct
            {
            unsigned DoLoop:1;
            unsigned OpenLoop:1;
            unsigned RunMotor:1;
            unsigned Btn1Pressed:1;
            unsigned Btn2Pressed:1;
            unsigned Btn3Pressed:1;
            unsigned Btn4Pressed:1;
            unsigned ChangeMode:1;
            unsigned ChangeSpeed:1;
            unsigned    :7;
            }bit;
        WORD Word;
        } uGF;                          // 通用标志

tPIParm     PIParmQ;
tPIParm     PIParmQref;
tPIParm     PIParmD;

tReadADCParm ReadADCParm;

int iRPM;
WORD iMaxLoopCnt;
WORD iLoopCnt;
WORD iDispLoopCnt;

/*********************************************************************/
void __attribute__((__interrupt__)) _ADCInterrupt(void);
void SetupBoard( void );
bool SetupParm(void);
void DoControl( void );
void Dis_RPM( BYTE bChrPosC, BYTE bChrPosR );
void DiagnosticsOutput(void);

/******************* 主函数开头 ************************/

int main ( void )
{
    SetupPorts();
    InitLCD();

    while(1)
        {
        uGF.Word = 0;                       // 清除标志

        // 初始化模式
        uGF.bit.OpenLoop = 1;               // 以开环模式起动

        // 初始化 LED
        pinLED1 = 0;
        pinLED2 = !uGF.bit.OpenLoop;
        pinLED3 = 0;
        pinLED4 = 0;

        // 初始化控制板
        SetupBoard();

        // 对用户指定参数进行初始化并在出错时停止
        if( SetupParm() )
            {
            // 错误
            uGF.bit.RunMotor=0;
            return;
```

```
    }

// 清零 i 和
PIParmD.qdSum = 0;
PIParmQ.qdSum = 0;
PIParmQref.qdSum = 0;

iMaxLoopCnt = 0;

Wrt_S_LCD("Vector Control  ", 0 , 0);
Wrt_S_LCD("S4-Run/Stop     ", 0, 1);

// 使能 ADC 中断并开始主循环定时
IFS0bits.ADIF = 0;
IEC0bits.ADIE = 1;

if(!uGF.bit.RunMotor)
    {
    // 初始化电流偏移量补偿
    while(!pinButton1)                      // 在此处等待直至按钮 1 按下
        {
        ClrWdt();

        // 开始偏移量累加                    // 并在等待时对电流偏移量进行累加
        MeasCompCurr();

        }
    while(pinButton1);                      // 当按钮 1 释放时
    uGF.bit.RunMotor = 1;                   // 随后起动电机
    }

// 电机运行
uGF.bit.ChangeMode = 1;
// 使能电机控制 PCB 上的驱动器 IC
pinPWMOutputEnable_ = 0;

Wrt_S_LCD("RPM=            ", 0, 0);
Wrt_S_LCD("S5-Cls. Lp S6-2x", 0, 1);

// 电机运行循环
while(1)
    {
    ClrWdt();

    // 如果使用 OC7 和 OC8 显示矢量控制变量，
    // 调用更新代码。
    #ifdefDIAGNOSTICS
    DiagnosticsOutput();
    #endif

    // 每隔 50 ms 执行更新 LCD 显示和查询按钮状态的代码。
    //

    if(iDispLoopCnt >= dDispLoopCnt)
    {
    //Display RPM
    Dis_RPM(5,0);

    // 按钮 1 控制电机的起停
        if(pinButton1)
            {
            if( !uGF.bit.Btn1Pressed )
                uGF.bit.Btn1Pressed  = 1;
            }
        else
```

```
    {
    if( uGF.bit.Btn1Pressed )
        {
        // 按钮刚被释放
        uGF.bit.Btn1Pressed  = 0;
        // 开始停止过程
        uGF.bit.RunMotor = 0;
        pinPWMOutputEnable_ = 1;
        break;
        }
    }

// 在运行时按钮 2 将控制开 / 闭环模式之间的切换
if(pinButton2)
    {
    if( !uGF.bit.Btn2Pressed )
        uGF.bit.Btn2Pressed  = 1;
    }
else
    {
    if( uGF.bit.Btn2Pressed )
        {
        // 按钮刚释放
        uGF.bit.Btn2Pressed  = 0;
        uGF.bit.ChangeMode = 1;
        uGF.bit.OpenLoop = ! uGF.bit.OpenLoop;
        pinLED2 = !uGF.bit.OpenLoop;
        }
    }

// 在运行时按钮 3 将加倍 / 减半速度或转矩给定
if(pinButton3)
    {
    if( !uGF.bit.Btn3Pressed )
        uGF.bit.Btn3Pressed  = 1;
        LATGbits.LATG0 = 0;
    }
else
    {
    if( uGF.bit.Btn3Pressed )
        {
        // 按钮刚释放
        uGF.bit.Btn3Pressed  = 0;
        uGF.bit.ChangeSpeed = !uGF.bit.ChangeSpeed;
        pinLED3 = uGF.bit.ChangeSpeed;
        LATGbits.LATG0 = 1;
        }
    }

// 按钮 4 不具备任何功能
if(pinButton4)
    {
    if( !uGF.bit.Btn4Pressed )
        uGF.bit.Btn4Pressed  = 1;
    }
else
    {
    if( uGF.bit.Btn4Pressed )
        {
        // 按钮刚被释放
        uGF.bit.Btn4Pressed  = 0;
        //*** 此处加入按钮 4 功能的代码
        }
    }
```

```
            }       // 显示和按钮查询代码结束

            }       // 电机运行循环结束


        }           //主循环结束

                    // 不应执行到此处
    while(1){}
}

//------------------------------------------------------------------------
// 对 Id 控制环、 Iq 控制环和速度控制环中的每个控制环执行一次 PI 迭代
void DoControl( void )
{
short i;

    // 假定 ADC 通道 0 具有来自速度电位器 （AN7）的有符号小数格式原始 A/D 值

    ReadSignedADC0( &ReadADCParm );

    // 设定给定速度
    if(uGF.bit.ChangeSpeed)
        CtrlParm.qVelRef = ReadADCParm.qADValue/8;
    else
        CtrlParm.qVelRef = ReadADCParm.qADValue/16;

    if( uGF.bit.OpenLoop )
        {
        // 开环：强制旋转角、 Vd 和 Vq

        if( uGF.bit.ChangeMode )
            {
            // 改变为开环模式
            uGF.bit.ChangeMode = 0;
            // 同步角度
            OpenLoopParm.qAngFlux = CurModelParm.qAngFlux;

            // 未使用 VqRef 和 VdRef
            CtrlParm.qVqRef = 0;
            CtrlParm.qVdRef = 0;
            }

        OpenLoopParm.qVelMech = CtrlParm.qVelRef;

        // 为 CorrectPhase 所需的给定值和符号
        // 计算 1.15 格式的转子磁通旋转角
        CurModelParm.qVelMech = EncoderParm.qVelMech;
        CurModel();

        ParkParm.qVq = 0;

        if( OpenLoopParm.qVelMech >= 0 )
            i = OpenLoopParm.qVelMech;
        else
            i = -OpenLoopParm.qVelMech;

        uWork = i <<2;

        if( uWork > 0x5a82 )
            uWork = 0x5a82;

        if( uWork < 0x1000 )
            uWork = 0x1000;
```

```
        ParkParm.qVd = uWork;

        OpenLoop();
        ParkParm.qAngle = OpenLoopParm.qAngFlux;

        }
    else
        // 闭环矢量控制
        {

        if( uGF.bit.ChangeMode )
            {
            // 改变为闭环模式
            uGF.bit.ChangeMode = 0;

            // 同步角度以及准备 qdImag
            CurModelParm.qAngFlux = OpenLoopParm.qAngFlux;
            CurModelParm.qdImag = ParkParm.qId;
            }

        // 根据电流模型计算角度
        CurModelParm.qVelMech = EncoderParm.qVelMech;

        CurModel();

        ParkParm.qAngle = CurModelParm.qAngFlux;

        // 计算弱磁控制模式的 qVdRef
        FdWeakening();

        // 设定给定速度

        // 如果应用运行在转矩模式，转速控制环将被忽略。
        // 从电位器读取的转速给定值将直接用作转矩给定 VqRef。
        #ifdefTORQUE_MODE
        CtrlParm.qVqRef    = CtrlParm.qVelRef;

        #else
        // 通过对比每一次转速计算中的中断数和速度计数采样数来确定是否可以获得新的转速信息。
        //
        // 如果可以获得新的转速信息，则计算新的转速值并执行
        // 转速控制环。
        if(EncoderParm.iVelCntDwn == EncoderParm.iIrpPerCalc)
            {
            // 根据编码器累计计数值来计算转速
        CalcVel();
        // 执行转速控制环
        PIParmQref.qInMeas = EncoderParm.qVelMech;
        PIParmQref.qInRef  = CtrlParm.qVelRef;
        CalcPI(&PIParmQref);
        CtrlParm.qVqRef    = PIParmQref.qOut;
        }
        #endif

        // Q 的 PI 控制
        PIParmQ.qInMeas = ParkParm.qIq;
        PIParmQ.qInRef  = CtrlParm.qVqRef;
        CalcPI(&PIParmQ);
        ParkParm.qVq    = PIParmQ.qOut;

        // D 的 PI 控制
```

```
        PIParmD.qInMeas = ParkParm.qId;
        PIParmD.qInRef  = CtrlParm.qVdRef;
        CalcPI(&PIParmD);
        ParkParm.qVd    = PIParmD.qOut;


        }
}

//-----------------------------------------------------------------------
// ADC 中断服务程序执行速度计算以及电压矢量更新循环。
// ADC 采样和转换由 PWM 周期触发。
// 速度计算假定计算之间的间隔时间是固定的。
//-----------------------------------------------------------------------

void __attribute__((__interrupt__)) _ADCInterrupt(void)
{
        IFS0bits.ADIF = 0;
        // 递增控制显示和按钮功能执行的计数变量。
        //
        iDispLoopCnt++;

        // 累计自上一次中断后的编码器计数
        CalcVelIrp();

        if( uGF.bit.RunMotor )
            {

            // 置位用于诊断的 LED1
            pinLED1 = 1;

            // 使用 TMR1 来测量用于诊断的中断时间
            TMR1 = 0;
            iLoopCnt = TMR1;

             MeasCompCurr();

            // 根据 qSin、qCos、qIa、qIb 计算 qId、qIq
            ClarkePark();

            // 计算控制值
            DoControl();

                // 根据 qAngle 计算 qSin、qCos
                SinCos();

                // 根据 qSin、qCos、qVd、qVq 计算 qValpha、qVbeta
                InvPark();

                // 根据 qValpha、qVbeta 计算 Vr1、Vr2、Vr3。
                CalcRefVec();

                // 根据 Vr1、Vr2、Vr3 计算和设定 PWM 占空比
                CalcSVGen();

                // 测量循环时间
                iLoopCnt = TMR1 - iLoopCnt;
                if( iLoopCnt > iMaxLoopCnt )
                    iMaxLoopCnt = iLoopCnt;

                // 清零用于诊断的 LED1
                pinLED1 = 0;
                }
}
```

```
//------------------------------------------------------------------------
// SetupBoard
//
// 初始化控制板
//------------------------------------------------------------------------

void SetupBoard( void )
{
    BYTE b;

    // 禁止 ADC 中断
    IEC0bits.ADIE = 0;

    // 复位电机控制功率模块上的任何故障。
    pinFaultReset = 1;
    for(b=0;b<10;b++)
        Nop();
    pinFaultReset = 0;


    // 确保 PFC 开关是关闭的。
    pinPFCFire = 0;
    // 确保制动开关是关闭的。
    pinBrakeFire = 0;
}

//------------------------------------------------------------------------
// Dis_RPM
//
// 显示 RPM
//------------------------------------------------------------------------

void Dis_RPM( BYTE bChrPosC, BYTE bChrPosR )
{

    if (EncoderParm.iDeltaCnt < 0)
        Wrt_S_LCD("-", bChrPosC, bChrPosR);
    else
        Wrt_S_LCD(" ", bChrPosC, bChrPosR);

    iRPM =
EncoderParm.iDeltaCnt*60/(MotorParm.fLoopPeriod*MotorParm.iIrpPerCalc*EncoderParm.iCntsPerRev);
    Wrt_Signed_Int_LCD( iRPM, bChrPosC+1, bChrPosR);
}
//------------------------------------------------------------------------
bool SetupParm(void)
{
    // 开启抗饱和功能以确保能够平滑处理溢出。
    CORCONbits.SATA  = 0;

    // 设置所需参数

    // 选取定标值为 8 倍速度和电流标称值

    // 在定标时使用 8 倍电机标称机械转速 （单位为 RPM）
    MotorParm.iScaleMechRPM  = diNomRPM*8;

    // 极对数
    MotorParm.iPoles        = diPoles ;

    // 由 dsPIC DSC 正交编码配置检测的每转编码器计数值。
    //
    MotorParm.iCntsPerRev  = diCntsPerRev;
```

```
    // 以秒为单位的转子时间常数
    MotorParm.fRotorTmConst = dfRotorTmConst;

    // 基本循环周期 （单位为秒）。（PWM 中断周期）
    MotorParm.fLoopPeriod = dLoopInTcy * dTcy;  // 循环周期 （以周期为单位） * 秒 / 周期

    // 编码器转速中断周期 （单位为秒）。
    MotorParm.fVelIrpPeriod = MotorParm.fLoopPeriod;

    // 每次转速计算的 vel 中断数。
    MotorParm.iIrpPerCalc = diIrpPerCalc;       // 以循环为单位

    // 电机的定标机械速度 （单位为转 / 秒）
    MotorParm.fScaleMechRPS = MotorParm.iScaleMechRPM/60.0;

    // 定标后的电机磁通转速 （单位为转 / 秒）
    // 通过该值定标后的所有无量纲磁通转速。
    MotorParm.fScaleFluxRPS = MotorParm.iPoles*MotorParm.fScaleMechRPS;

    // 磁通矢量每转一周的最小周期时间 （单位为秒）
    MotorParm.fScaleFluxPeriod = 1.0/MotorParm.fScaleFluxRPS;

    // 在最大磁通转速时的每个 LoopTime 的转数系数
    MotorParm.fScaleFracRevPerLoop = MotorParm.fLoopPeriod * MotorParm.fScaleFluxRPS;

    // 定标后的电机磁通转速 （单位为弧度 / 秒）
    // 通过该值定标后的所有无量纲转速 （单位为弧度 / 秒）。
    MotorParm.fScaleFluxSpeed = 6.283 * MotorParm.fScaleFluxRPS;

    // iScaleMechRPM 时的编码器计数频率
    MotorParm.lScaleCntRate = MotorParm.iCntsPerRev * (MotorParm.iScaleMechRPM/60.0);



// ============= 开环 ======================

    OpenLoopParm.qKdelta = 32768.0 * 2 * MotorParm.iPoles * MotorParm.fLoopPeriod *
MotorParm.fScaleMechRPS;
    OpenLoopParm.qVelMech = dqOL_VelMech;
    CtrlParm.qVelRef = OpenLoopParm.qVelMech;

    InitOpenLoop();

// ============= 编码器 ===============

    if( InitEncoderScaling() )
        // 出错
        return True;

// ============= ADC - 测量电流和电位器 ======================

    // 定标常数：由校准或硬件设计决定。
    ReadADCParm.qK      = dqK;

    MeasCurrParm.qKa    = dqKa;
    MeasCurrParm.qKb    = dqKb;

    // 初始偏移量
    InitMeasCompCurr( 450, 730 );

// ============= 电流模型 ===============

    if(InitCurModelScaling())
        // 出错
        return True;
```

```
// ============= 弱磁 ===============
    // 恒转矩范围的弱磁常数
    FdWeakParm.qK1 = dqK1;        // 磁通给定值

// ============= PI D项 ===============
    PIParmD.qKp = dDqKp;
    PIParmD.qKi = dDqKi;
    PIParmD.qKc = dDqKc;
    PIParmD.qOutMax = dDqOutMax;
    PIParmD.qOutMin = -PIParmD.qOutMax;

    InitPI(&PIParmD);

// ============= PI Q项 ===============
    PIParmQ.qKp = dQqKp;
    PIParmQ.qKi = dQqKi;
    PIParmQ.qKc = dQqKc;
    PIParmQ.qOutMax = dQqOutMax;
    PIParmQ.qOutMin = -PIParmQ.qOutMax;

    InitPI(&PIParmQ);

// ============= PI Qref项 ===============
    PIParmQref.qKp = dQrefqKp;
    PIParmQref.qKi = dQrefqKi;
    PIParmQref.qKc = dQrefqKc;
    PIParmQref.qOutMax = dQrefqOutMax;
    PIParmQref.qOutMin = -PIParmQref.qOutMax;

    InitPI(&PIParmQref);

// ============= SVGen ===============
    // 将PWM周期设定为循环时间
    SVGenParm.iPWMPeriod = dLoopInTcy;

// ============= TIMER #1 ======================
    PR1 = 0xFFFF;
    T1CONbits.TON = 1;
    T1CONbits.TCKPS = 1;     // 预分频比为8时 => 一个单位为1.08504 uS

// ============= 电机PWM ======================

    PDC1 = 0;
    PDC2 = 0;
    PDC3 = 0;
    PDC4 = 0;

    // 中心对齐PWM。
    // 注：PWM周期设定为dLoopInTcy/2，但由于先采用递增计数随后为递减计数，
    // => 在计数到零时中断标志置1 => 因此实际中断周期为dLoopInTcy
    //
    PTPER = dLoopInTcy/2;    // 将PWM周期设定为循环时间，该参数在parms.h中定义

    PWMCON1 = 0x0077;        // 使能PWM 1,2,3对工作在互补模式
    DTCON1 = dDeadTime;      // 死区时间
    DTCON2 = 0;
    FLTACON = 0;             // 未使用PWM故障引脚
    FLTBCON = 0;
    PTCON = 0x8002;          // 使能PWM中心对齐

    // SEVTCMP：特殊事件比较计数寄存器
    // 相对于PWM周期的ADC捕获设定相位： 0 偏移量和递增计数
    SEVTCMP = 2;          // 不能为0 -> 关断触发器（从doc丢失）
```

```
    SEVTCMPbits.SEVTDIR = 0;

// ============= 编码器 ===============

    MAXCNT = MotorParm.iCntsPerRev;
    POSCNT = 0;
    QEICON = 0;
    QEICONbits.QEIM = 7;     // 通过 MAXCNT 脉冲复位 x4
    QEICONbits.POSRES = 0;   // 不要让索引脉冲复位计数器
    QEICONbits.SWPAB = 0;    // 方向
    DFLTCON = 0;             // 数字滤波器设定为关闭

// ============= ADC - 测量电流和电位器给定值 ======================
// ADC 设定为对以下通道同时进行采样：
//      CH0=AN7, CH1=AN0, CH2=AN1, CH3=AN2
// 采样由 PWM 触发，且采样结果以有符号小数形式存放。

    ADCON1 = 0;
    // 有符号小数格式 （DOUT = sddd dddd dd00 0000）
    ADCON1bits.FORM = 3;
    // 电机控制 PWM 间隔终止采样并启动转换
    ADCON1bits.SSRC = 3;
    // 同时采样选择位 （仅当 CHPS =01 或 1x 时应用）
    // 同时采样 CH0、 CH1、 CH2 和 CH3 （当 CHPS = 1x 时）
    // 同时采样 CH0 和 CH1 （当 CHPS=01 时）
    ADCON1bits.SIMSAM = 1;
    //在上一次转换结束后立即开始采样。
    // SAMP 位自动置位。
    ADCON1bits.ASAM = 1;


    ADCON2 = 0;
    // 同时采样 CH0、 CH1、 CH2、 CH3 （当 CHPS = 1x 时）
    ADCON2bits.CHPS = 2;


    ADCON3 = 0;
    // A/D 转换时钟选择位 = 8 * Tcy
    ADCON3bits.ADCS = 15;


    /* ADCHS：ADC 输入通道选择寄存器 */
    ADCHS = 0;
    // CH0 为 AN7
    ADCHSbits.CH0SA = 7;
    // CH1 正极性输入为 AN0， CH2 正极性输入为 AN1， CH3 正极性输入为 AN2
    ADCHSbits.CH123SA = 0;


    /* ADPCFG：ADC 端口配置寄存器 */
    // 将所有端口设置为数字端口
    ADPCFG = 0xFFFF;
    ADPCFGbits.PCFG0 = 0;    // AN0 模拟
    ADPCFGbits.PCFG1 = 0;    // AN1 模拟
    ADPCFGbits.PCFG2 = 0;    // AN2 模拟
    ADPCFGbits.PCFG7 = 0;    // AN7 模拟

    /* ADCSSL：ADC 输入扫描选择寄存器 */
    ADCSSL = 0;

    // 开启 A/D 模块
    ADCON1bits.ADON = 1;

    #ifdefDIAGNOSTICS
    // 对用于诊断模式的输出比较通道 7 和 8 进行初始化。
```

```
    // PWM 模式中使用比较
    // Timer2 用作时基
    PR2 = 0x1FFF;
    OC7CON = 0x0006;
    OC8CON = 0x0006;
    T2CONbits.TON = 1;
    #endif


    return False;
}

#ifdefDIAGNOSTICS
void DiagnosticsOutput(void)
{
int Data;

    if(IFS0bits.T2IF)
        {
        IFS0bits.T2IF = 0;
        Data = (ParkParm.qIq >> 4) + 0xfff;
        if(Data > 0x1ff0) Data = 0x1ff0;
        if(Data < 0x000f) Data = 0x000f;
        OC7RS = Data;
        Data =  (EncoderParm.qVelMech) + 0x0fff;
        if(Data > 0x1ff0) Data = 0x1ff0;
        if(Data < 0x000f) Data = 0x000f;
        OC8RS = Data;
        }

}
#endif
```

Encoder.c

```
// 编码器子程序的定标

#include "general.h"
#include "Parms.h"
#include "Encoder.h"

/*******************************************************
InitEncoderScaling

对编码器子程序的定标常量进行初始化。

    函数参数：
        CntsPerRev：来自正交编码器的每转编码器计数值
        ScalingSpeedInRPS：用于基本转速定标的每秒转数
        IrpPerCalc：每次转速计算的 CalcVelIrp 中断数
        VelIrpPeriod：VelCalcIrp 中断间的周期 （单位为秒）

对于 CalcAng：
    运行时公式：
    qMechAng = qKang * (POSCNT*4) / 2^Nang
    定标公式：
        qKang = (2^15)*(2^Nang)/CntsPerRev.
对于 CalcVelIrp、CalcVel：
    运行时公式：
        qMechVel = qKvel * (2^15 * Delta / 2^Nvel)
    定标公式：
        fVelCalcPeriod = fVelIrpPeriod * iIrpPerCalc
        MaxCntRate = CntsPerRev * ScaleMechRPS
        MaxDeltaCnt = fVelCalcPeriod * MaxCntRate
        qKvel = (2^15)*(2^Nvel)/MaxDeltaCnt
*******************************************************/

bool InitEncoderScaling( void )
{
    float fVelCalcPeriod, fMaxCntRate;
    long  MaxDeltaCnt;
    long  K;

    EncoderParm.iCntsPerRev = MotorParm.iCntsPerRev;

    K = 32768;
    K *= 1 << Nang;
    EncoderParm.qKang = K/EncoderParm.iCntsPerRev;

    EncoderParm.iIrpPerCalc = MotorParm.iIrpPerCalc;
    fVelCalcPeriod = MotorParm.fVelIrpPeriod * MotorParm.iIrpPerCalc;
    fMaxCntRate = EncoderParm.iCntsPerRev * MotorParm.fScaleMechRPS;
    MaxDeltaCnt = fVelCalcPeriod * fMaxCntRate;

    // qKvel = (2^15)*(2^Nvel)/MaxDeltaCnt
    K = 32768;
    K *= 1 << Nvel;
    K /= MaxDeltaCnt;
    if( K >= 32768 )
        // 出错
        return True;
    EncoderParm.qKvel = K;

    // 对 CalcVelIrp 使用的局部变量进行初始化。
    InitCalcVel();
    return False;
}
```

InitCurModel.c

```
// 电流模型子程序定标

#include "general.h"
#include "Parms.h"
#include "CurModel.h"

/*******************************************************
InitCurModelScaling

对电流模型子程序中的定标常数进行初始化。

物理常数：
  fRotorTmConst         转子时间常数，单位为秒。

公式的物理形式：
  励磁电流（安培）：
     Imag = Imag + (fLoopPeriod/fRotorTmConst)*(Id - Imag)

  转差速度，单位为 RPS：
     VelSlipRPS = (1/fRotorTmConst) * Iq/Imag / (2*pi)

  转子磁通速度，单位为 RPS：
     VelFluxRPS = iPoles * VelMechRPS + VelSlipRPS

  转子磁通角（弧度）：
     AngFlux = AngFlux + fLoopPeriod * 2 * pi * VelFluxRPS

定标后的变量：
  qImag       采用最大电流定标后的励磁电流。
  qVelSlip    采用 fScaleMechRPS 定标后的机械转差速度，单位为 RPS。
  qAngFlux    采用 pi 定标后的磁通角。

定标后的公式：
  qImag       = qImag + qKcur * (qId - qImag)
  qVelSlip    = Kslip * qIq/qImag
  qAngFlux    = qAngFlux + Kdelta * (qVelMech + qVelSlip)

定标因子：
  qKcur = (2^15) * (fLoopPeriod/fRotorTmConst)
  qKdelta = (2^15) * 2 * iPoles * fLoopPeriod * fScaleMechRPS
  qKslip = (2^15)/(2 * pi * fRotorTmConst * iPoles * fScaleMechRPS)



*******************************************************/

bool InitCurModelScaling( void )
{
    CurModelParm.qKcur = 32768.0 * MotorParm.fLoopPeriod / MotorParm.fRotorTmConst;

    CurModelParm.qKdelta = 32768.0 * 2 * MotorParm.iPoles * MotorParm.fLoopPeriod *
MotorParm.fScaleMechRPS;

    CurModelParm.qKslip = 32768.0/(6.2832 * MotorParm.iPoles *
MotorParm.fScaleMechRPS*MotorParm.fRotorTmConst);

    // 允许的最大转差速度
    CurModelParm.qMaxSlipVel = 32768.0/8;

    // 对 CurrModel 使用的局部变量进行初始化。
    InitCurModel();
    return False;
}
```

MeasCur.s

```
;*********************************************************************
; MeasCompCurr
;
; 说明:
;   读 ADC 通道 1 和通道 2，将其换算为有符号小数值。
;   使用 qKa 和 qKb 并将结果存放到 ParkParm 的 qIa 和 qIb 中。
;   ADC-Ave 的运行平均值被保持并在换算前从 ADC 值中减去。
;
;
;   具体来说，偏移量将作为 32 位有符号整数进行累计。
;       iOffset += (ADC-Offset)
;   并采用以下公式通过偏移量来校正原始的 ADC 读数
;       CorrADC = ADCBUFn - iOffset/2^16
;   将给出一个偏移时间常数 ~ MeasurementPeriod*2^16
;
;   在转换结束之前不要调用该子程序。
;
;   定标常数 qKa 和 qKb 必须在其他代码中设定，使
;       qIa = 2 * qKa * CorrADC1
;       qIb = 2 * qKb * CorrADC2
;   采用 2 作为因子以允许 qKa 和 qKb 保持 1.15 格式。
;
; 函数原型:
;       void MeasCompCurr( void );
;       void InitMeasCompCurr( short iOffset_a, short iOffset_b );
;
; 开头:          必须调用 InitMeasCompCurr。
; 进入时:        MeasCurrParm 结构必须包含 qKa 和 qKb。
;                ADC 通道 1 和 2 必须包括有符号小数值。
; 退出时:        ParkParm 将包含 qIa 和 qIb。
;
; 参数:
;   输入参数:
;       无
;   返回值:
;       Void
;   所需的 SFR 设定:
;       CORCON.SATA  = 0
;   如果累加器可能溢出，必须设定:
;       CORCON.SATDW = 1
;
;   所需的支持子程序:
;       无
;   局部堆栈使用:
;       无
;   修改的寄存器:
;       w0,w1,w4,w5
;   执行时间:
;       29 个周期
;*********************************************************************

        global   _MeasCompCurr
        global   MeasCompCurr

_MeasCompCurr:
MeasCompCurr:

    ;; CorrADC1 = ADCBUF1 - iOffsetHa/2^16
    ;; qIa = 2 * qKa * CorrADC1
        mov.w       _MeasCurrParm+ADC_iOffsetHa,w0
        sub.w       _ADCBUF1,WREG                   ; w0 = ADC - 偏移量
        clr.w       w1
        btsc        w0,#15
        setm        w1
```

```
        mov.w       w0,w5
        mov.w       _MeasCurrParm+ADC_qKa,w4
        mpy         w4*w5,A
        sac         A,#-1,w4
        mov.w       w4,_ParkParm+Park_qIa

    ;; iOffset += (ADC- 偏移量 )
            add     _MeasCurrParm+ADC_iOffsetLa
            mov.w   w1,w0
            addc    _MeasCurrParm+ADC_iOffsetHa

    ;; CorrADC2 = ADCBUF2 - iOffsetHb/2^16
    ;; qIb = 2 * qKb * CorrADC2
        mov.w       _MeasCurrParm+ADC_iOffsetHb,w0
        sub.w       _ADCBUF2,WREG                   ; w0 = ADC - 偏移量
        clr.w       w1
        btsc        w0,#15
        setm        w1
        mov.w       w0,w5
        mov.w       _MeasCurrParm+ADC_qKb,w4
        mpy         w4*w5,A
        sac         A,#-1,w4
        mov.w       w4,_ParkParm+Park_qIb

    ;; iOffset += (ADC-Offset)
        add         _MeasCurrParm+ADC_iOffsetLb
        mov.w       w1,w0
        addc        _MeasCurrParm+ADC_iOffsetHb

        return
```

ClarkePark.s

```
;*********************************************************************
; ClarkePark
;
; 说明:
;   计算 Clarke 和 Park 变换。
;   假定 Cos 和 Sin 值在 qSin 和 qCos 中。
;
;       Ialpha = Ia
;       Ibeta  = Ia*dOneBySq3 + 2*Ib*dOneBySq3;
;           其中 Ia+Ib+Ic = 0
;
;       Id =  Ialpha*cos( 角度 ) + Ibeta*sin( 角度 )
;       Iq = -Ialpha*sin( 角度 ) + Ibeta*cos( 角度 )
;
;   该子程序同样适用于整数定标和 1.15 定标格式。
;
; 函数原型:
;
;   void ClarkePark( void )
;
; 进入时:    ParkParm 结构必须包含 qSin、 qCos、 qIa 和 qIb。
; 退出时:    ParkParm 将包含 qId 和 qIq。
;
; 参数:
;   输入参数:
;   返回值:
;       Void
;   所需的 SFR 设定:
;       CORCON.SATA  = 0
;   如果 (Ia+2*Ib)/sqrt(3) 可能出现溢出，必须设定
```

```
;       CORCON.SATDW = 1
;
; 所需的支持子程序：
;       无
; 局部堆栈使用：
;       无
; 修改的寄存器：
;       w3 -> w7
; 执行时间：
;       20 个周期
;*****************************************************************
;
            include "general.inc"
; 外部引用
            include "park.inc"
; 寄存器使用
            .equ ParmW,     w3                  ; 指向 ParkParm 结构的指针
            .equ Sq3W,      w4                  ; OneBySq3
            .equ SinW,      w4                  ; 替代 Work0W
            .equ CosW,      w5
            .equ IaW,       w6                  ; qIa 的拷贝
            .equ IalphaW,   w6                  ; 替代 Ia
            .equ IbW,       w7                  ; qIb 的拷贝
            .equ IbetaW,    w7                  ; 用 Ibeta 替代 Ib
; 常量
            equ OneBySq3,   0x49E7              ; 1/sqrt(3)，采用 1.15 格式
;=================== 代码 =====================
            section  .text
            global   _ClarkePark
            global   ClarkePark

_ClarkePark:
ClarkePark:
    ;; Ibeta = Ia*OneBySq3 + 2*Ib*OneBySq3;

        mov.w   #OneBySq3,Sq3W                  ; 1/sqrt(3)，采用 1.15 格式
        mov.w   _ParkParm+Park_qIa,IaW
        mpy     Sq3W*IaW,A
        mov.w   _ParkParm+Park_qIb,IbW
        mac     Sq3W*IbW,A
        mac     Sq3W*IbW,A
        mov.w   _ParkParm+Park_qIa,IalphaW
        mov.w   IalphaW,_ParkParm+Park_qIalpha
        sac     A,IbetaW
        mov.w   IbetaW,_ParkParm+Park_qIbeta

    ;; 已经计算 Ialpha 和 Ibeta。现在进行旋转。

    ;; 得到 ParkParm 结构的 qSin、qCos 。
        mov.w   _ParkParm+Park_qSin,SinW
        mov.w   _ParkParm+Park_qCos,CosW

    ;; Id =  Ialpha*cos( 角度 ) + Ibeta*sin( 角度 )

        mpy     SinW*IbetaW,A                   ; Ibeta*qSin -> A
        mac     CosW*IalphaW,A                  ; 将 Ialpha*qCos 加到 A
        mov.w   #_ParkParm+Park_qId,ParmW
        sac     A,[ParmW++]                     ; 存放到 qId，将指针递增 1 指向 qIq

    ;; Iq = -Ialpha*sin(Angle) + Ibeta*cos(Angle)
        mpy     CosW*IbetaW,A                   ; Ibeta*qCos -> A
        msc     SinW*IalphaW,A                  ; 从 A 减去 Ialpha*qSin
        sac     A,[ParmW]                       ; 存入 qIq
        return
        .end
```

CurModel.s

```
;*********************************************************************
; 子程序： CurModel
;*********************************************************************
; 为文件中所有子程序共有
            .include "general.inc"
            .include "curmodel.inc"
            .include "park.inc"
;*********************************************************************
; CurModel
;
; 说明：
;
; 物理常数：
;   fRotorTmConst           转子时间常数，单位为秒
;
; 公式的物理形式：
;   励磁电流 （安培）：
;       Imag = Imag + (fLoopPeriod/fRotorTmConst)*(Id - Imag)
;
;   转差速度，单位为 RPS：
;       VelSlipRPS = (1/fRotorTmConst) * Iq/Imag / (2*pi)
;
;   转子磁通速度，单位为 RPS：
;       VelFluxRPS = iPoles * VelMechRPS + VelSlipRPS
;
;   转子磁通角 （弧度）：
;       AngFlux = AngFlux + fLoopPeriod * 2 * pi * VelFluxRPS
;
; 定标后的变量：
;   qdImag              采用最大电流定标的励磁电流 （1.31）
;   qVelSlip            采用 fScaleMechRPS 定标后的机械转差速度 （单位为 RPS）
;   qAngFlux            采用 pi 定标后的磁通角
;
; 定标后的公式：
;   qdImag      = qdImag + qKcur * (qId - qdImag)
;   qVelSlip    = qKslip * qIq/qdImag
;   qAngFlux    = qAngFlux + qKdelta * (qVelMech + qVelSlip)
;
; 定标因子：
;   qKcur       = (2^15) * (fLoopPeriod/fRotorTmConst)
;   qKdelta     = (2^15) * 2 * iPoles * fLoopPeriod * fScaleMechRPS
;   qKslip      = (2^15)/(2 * pi * fRotorTmConst * iPoles * fScaleMechRPS)
;
; 函数原型：
;
;   void CurModel( void )
;
; 进入时：        CurModelParm 结构必须包含 qKcur、 qKslip、 iKpoles、
;                 qKdelta、 qVelMech 和 qMaxSlipVel
; 退出时：        CurModelParm 将包含 qAngFlux、 qdImag 和 qVelSlip
;
; 参数：
;   输入参数：
;       无
;   返回值：
;       Void
;   所需的 SFR 设定：
;       CORCON.SATA     = 0
;       CORCON.IF       = 0
;
;   所需的支持子程序：
;       无
;   局部堆栈使用：
;       0
```

```
;   修改的寄存器：
:       w0-w7,AccA
;   执行时间：
;       72 个指令周期
;*****************************************************************
;
;=================== 代码 =====================
            .section  .text

;   用于 CurModel 的寄存器
            .equ SignW,     w2                              ; 跟踪符号变化
            .equ ShiftW,    w3                              ; 执行除法之前的移位位数
            .equ IqW,       w4                              ; Q 电流 （1.15）
            .equ KslipW,    w5                              ; Kslip 常数 （1.15）
            .equ ImagW,     w7                              ; 励磁电流 （1.15）

            .global     _CurModel
            .global     CurModel

_CurModel:
CurModel:
        ;; qdImag = qdImag + qKcur * (qId - qdImag)       ;; 励磁电流
            mov.w       _CurModelParm+CurMod_qdImag,w6
            mov.w       _CurModelParm+CurMod_qdImag+2,w7
            lac         w7,A
            mov.w       w6,ACCALL

            mov.w       _ParkParm+Park_qId,w4
            sub.w       w4,w7,w4                            ; qId-qdImagH
            mov.w       _CurModelParm+CurMod_qKcur,w5

            mac         w4*w5,A                             ; 将 Kcur*(Id-Imag) 加到 Imag
            sac         A,w7
            mov.w       ACCALL,w6
            mov.w       w6,_CurModelParm+CurMod_qdImag
            mov.w       w7,_CurModelParm+CurMod_qdImag+2

    ;; qVelSlip = qKslip * qIq/qdImag

    ;; 首先将 qIqW 和 qdImagW 置为正数，并将符号位存放在 SignW 中
            clr         SignW                               ; 将标志符号设定为正

    ;; if( IqW < 0 ) => 翻转 SignW 并设定 IqW = -IqW
            mov.w       _ParkParm+Park_qIq,IqW
            cp0         IqW
            bra         Z,jCurModSkip
            bra         NN,jCurMod1
            neg         IqW,IqW
            com         SignW,SignW                         ; 翻转符号位
jCurMod1:
    ;; if( ImagW < 0 ) => 翻转 SignW 并设定 ImagW = -ImagW
            cp0         ImagW
            bra         NN,jCurMod2
            neg         ImagW,ImagW
            com         SignW,SignW                         ; 翻转符号位
jCurMod2:
    ;; 在 Acc A 中计算 Kslip*|IqW| 以保持 1.31 格式
            mov.w       _CurModelParm+CurMod_qKslip,KslipW
            mpy         IqW*KslipW,A

    ;; 确保 denominator > numerator，否则跳过项
            sac         A,w0                                ; 暂时的
            cp          ImagW,w0                            ; |qdImag| - |Kslip*qIq|
            bra         LEU,jCurModSkip                     ; 跳过项：|qdImag| <= |Kslip*qIq|
```

```
    ;; 在 6010 <SILICON_ERR> 以后版本中将不再需要。
            clr.w       ShiftW

    ;; 计算不将最高有效位直接置 1（保留符号位）的情况下，要将 ImagW 移位多少位。
    ;;
            ff1l        ImagW,ShiftW
            sub.w       ShiftW,#2,ShiftW                ; 为将 1 放入 bit 14，需要移位的位数
    ;; 移位：ImagW = ImagW << ShiftW
            sl          ImagW,ShiftW,ImagW
    ;; 对 AccA 进行移位，需要将 (-ShiftW) 左移。
            neg         ShiftW,ShiftW
    ;; |Kslip*qIq| = |Kslip*qIq| << ShiftW
            sftac       A,ShiftW

    ;; 执行除法操作 |qKslip*qIq|/|ImagW|。此时结果将为正且 < 1.0，同时具有最高的精度。
    ;;
    ;;

            sac         A,w6
            repeat      #17
            divf        w6,ImagW                        ; w0 = KslipW*IqW/ImagW, w1 = remainder
    ;; 限制最大转差速度
            mov.w       _CurModelParm+CurMod_qMaxSlipVel,w1
            cp          w1,w0                           ; qMaxSlipSpeed - | Kslip*qIq/qdImag |
            bra         NN,jCurMod4
    ;; 结果太大：用 qMaxSlipSpeed 代替
            mov.w       w1,w0
            bra         jCurMod4

jCurModSkip:
    ;; 整个项被跳过 - 将其置为 = 0
            clr.w       w0

jCurMod4:
    ;; 设定正确的符号
            btsc        SignW,#0
            neg         w0,w0
    ;; 用于测试
            mov.w       w0,_CurModelParm+CurMod_qVelSlip
    ;; 加入机械转速
            mov.w       _CurModelParm+CurMod_qVelMech,w4
            add.w       w0,w4,w4
            mov.w       w4,_CurModelParm+CurMod_qVelFlux
    ;; 将 AngFlux 载入 Acc A
            mov.w       _CurModelParm+CurMod_qAngFlux,w1
            lac         w1,A

            mov.w       _CurModelParm+CurMod_qKdelta,w5
            mac         w4*w5,A

            sac         A,w4
            mov.w       w4,_CurModelParm+CurMod_qAngFlux
    return
```

InvPark.s

```
;*********************************************************************
; InvPark
;
; 说明：
;   计算 Park 反变换。假定 Cos 和 Sin 值位于 ParkParm 结构中。
;
;           Valpha =  Vd*cos(Angle) - Vq*sin(Angle)
;           Vbeta  =  Vd*sin(Angle) + Vq*cos(Angle)
;   该子程序同样适用于整数定标和 1.15 定标格式。
;
; 函数原型：
;   void InvPark( void )
; 进入时：        ParkParm 结构必须包含 qCos、 qSin、 qVd 和 qVq。
; 退出时：        ParkParm 将包含 qValpha 和 qVbeta。
;
; 参数：
;   输入参数：                      无
;   返回值：                        Void
;   所需的 SFR 设定：               CORCON.SATA  = 0
;   所需的支持子程序：              None
;   局部堆栈使用：                  None
;   修改的寄存器：                  w3 -> w7, A
;   执行时间：                      大约 14 个指令周期
;*********************************************************************
;
        include "general.inc"
;   外部引用
        include "park.inc"
;   寄存器使用
        .equ ParmW,     w3          ; 指向 ParkParm 结构的指针
        .equ SinW,      w4
        .equ CosW,      w5
        .equ VdW,       w6          ; qVd 的拷贝
        .equ VqW,       w7          ; qVq 的拷贝

;=================== 代码 =====================

        .section    .text
        .global     _InvPark
        .global     InvPark

_InvPark:
InvPark:
    ;; 从 ParkParm 结构获得 qVd 和 qVq
        mov.w       _ParkParm+Park_qVd,VdW
        mov.w       _ParkParm+Park_qVq,VqW
    ;; 从 ParkParm 结构获得 qSin 和 qCos
        mov.w       _ParkParm+Park_qSin,SinW
        mov.w       _ParkParm+Park_qCos,CosW

;; Valpha =  Vd*cos(Angle) - Vq*sin(Angle)
        mpy     CosW*VdW,A          ; Vd*qCos -> A
        msc     SinW*VqW,A          ; 从 A 减去 Vq*qSin

        mov.w   #_ParkParm+Park_qValpha,ParmW
        sac     A,[ParmW++]         ; 存储到 qValpha，指针递增 1 指向 qVbeta

    ;; Vbeta  =  Vd*sin(Angle) + Vq*cos(Angle)
        mpy     SinW*VdW,A          ; Vd*qSin -> A
        mac     CosW*VqW,A          ; 将 Vq*qCos 加到 A
        sac     A,[ParmW]           ; 存储到 Vbeta

        return
```

CalcRef.s

```
;*********************************************************************
; CalcRefVec
;
; 说明：
;   根据 qValpha 和 qVbeta 计算定标后的参考矢量 （Vr1, Vr2, Vr3）。
;   该方法是一种修正后的 Clarke 反变换。与常规 Clarke 反变换相比，将 Valpha 和 Vbeta
;   进行了交换。
;
;       Vr1 = Vbeta
;       Vr2 = (-Vbeta/2 + sqrt(3)/2 * Valpha)
;       Vr3 = (-Vbeta/2 - sqrt(3/2) * Valpha)
;
; 函数原型：
;
; void CalcRefVec( void )
;
; 进入时：ParkParm 结构必须包含 qCos、 qSin、 qValpha 和 qVbeta。
; 退出时：    SVGenParm 将包含 qVr1、 qVr2 和 qVr3
;
; 参数：
;   输入参数：
;       无
; 返回值：
;       Void
; 所需的 SFR 设定：
;       CORCON.SATA = 0
; 所需的支持子程序：
;       无
; 局部堆栈使用：
;       无
; 修改的寄存器：
;       w0, w4, w5, w6
; 执行时间：
;       大约 20 个指令周期
;*********************************************************************
;
        .include "general.inc"

; 外部引用
        .include "park.inc"
        .include "SVGen.inc"
; 寄存器使用
        .equ WorkW,         w0                      ; 工作
        .equ ValphaW,       w4                      ; qValpha （定标后）
        .equ VbetaW,        w5                      ; qVbeta （定标后）
        .equ ScaleW,        w6                      ; 定标
; 常量
        .equ Sq3OV2,0x6ED9                          ; sqrt(3)/2，采用 1.15 格式
;=================== 代码 ======================

        .section            .text
        .global             _CalcRefVec
        .global             CalcRefVec

_CalcRefVec:
CalcRefVec:
    ;; 从 ParkParm 结构获得 qValpha 和 qVbeta。
        mov.w               ParkParm+Park_qValpha,ValphaW
        mov.w               _ParkParm+Park_qVbeta,VbetaW
    ;; 使 Vr1 = Vbeta
        mov.w               VbetaW,_SVGenParm+SVGen_qVr1
    ;; 装载 Sq(3)/2
        mov.w               #Sq3OV2,ScaleW
```

```
;; AccA = -Vbeta/2
    neg.w               VbetaW,VbetaW
    lac                 VbetaW,#1,A
;; Vr2 = -Vbeta/2 + sqrt(3)2 * Valpha)
    mac                 ValphaW*ScaleW,A ; 将 Valpha*sqrt(3)/2 加到 A
    sac                 A,WorkW
    mov.w               WorkW,_SVGenParm+SVGen_qVr2
;; AccA = -Vbeta/2
    lac                 VbetaW,#1,A
;; Vr3 = (-Vbeta/2 - sqrt(3)2 * Valpha)
    msc                 ValphaW*ScaleW,A ; 从 A 减去 Valpha*sqrt(3)/2。
    sac                 A,WorkW
    mov.w               WorkW,_SVGenParm+SVGen_qVr3
    return
    .end
```

CalcVel.s

```
;*****************************************************************
; 子程序: InitCalcVel, CalcVel
;
;*****************************************************************
; 为文件中所有子程序共有

        .include "general.inc"
        .include "encoder.inc"

;*****************************************************************
; void InitCalcVel(void)
;       对局部转速变量进行初始化。
;       在进入时 iIrpPerCalc 必须被设定。
;*****************************************************************

; InitCalcVel 的寄存器使用

        .equ Work0W,  w4    ; 工作寄存器
        .equ PosW,    w5    ; 当前位置: POSCNT

;*****************************************************************

        .global   _InitCalcVel
        .global   InitCalcVel
_InitCalcVel:
InitCalcVel:

    ;; 此后的 5 条指令禁止中断。
        DISI      #5

    ;; 装载 iPrevCnt 和 zero Delta
    ;; 编码器值。注意: 要获得精确的转速, qVelMech 必须计算两次。
    ;;
        mov.w     POSCNT,PosW          ; 当前编码器值
        mov.w     PosW,_EncoderParm+Encod_iPrevCnt
        clr.w     _EncoderParm+Encod_iAccumCnt

    ;; 装载 iVelCntDwn
        mov.w     _EncoderParm+Encod_iIrpPerCalc,WREG
        mov.w     WREG,_EncoderParm+Encod_iVelCntDwn

        return

;*****************************************************************
; CalcVelIrp
;
; 以指定的间隔, 通过定时器中断调用。
;
; 中断间隔 VelPeriod, 必须小于最大转速时转过 1/2 转所需的最小时间。
;
; 该子程序将对 iIrpPerCalc 中断的编码器计数变化进行累加,
; 时间周期 = iIrpPerCalc * VelPeriod, 随后将累加值复制到
; iDeltaCnt 以供 CalcVel 子程序进行转速计算使用。
; 该累加值被置回零并开始一个新的累加操作。
;
; 函数原型:              void CalcVelIrp( void );
;
; 进入时:                EncoderParm 必须包含 iPrevCnt、 iAccumCnt 和 iVelCntDwn
;
; 退出时:                EncoderParm 将包含 iPrevCnt、 iAccumCnt 和 iDeltaCnt
;                        (如果向下计数到达零)。
;
```

```
; 参数：
;   输入参数                          无
;
;   返回：
;       Void
;
;   所需的 SFR 设定：                 无
;
;   所需的支持子程序：                无
;
;   局部堆栈使用：                    3
;
;   修改的寄存器：                    无
;
;   执行时间：                        大约 29 个指令周期 （如果产生了新的 iDeltaCnt）。
;
;=====================================================
; 等效的 C 代码
;   {
;   register short Pos, Delta;
;
;   Pos = POSCNT;
;
;   Delta = Pos - EncoderParm.iPrevCnt;
;   EncoderParm.iPrevCnt = Pos;
;
;   if( iDelta >= 0 )
;       {
;       // Delta > 0 或因为
;       //      1) vel > 0 或
;       //      2) Vel < 0 且编码器计数值出现翻转返回。
;
;       if( Delta >=  EncoderParm.iCntsPerRev/2 )
;           {
;           // Delta >= EncoderParm.iCntsPerRev/2 => 负转速，出现翻转返回
;
;           Delta -= EncoderParm.iCntsPerRev;
;           }
;       }
;   else
;       {
;       // Delta < 0 或因为
;       //      1) vel < 0 或
;       //      2) Vel > 0 并出现翻转返回
;;
;       if( Delta < -EncoderParm.iCntsPerRev/2 )
;           {
;           // Delta < -EncoderParm.iCntsPerRev/2 => 正转速，出现翻转返回
;
;           Delta += EncoderParm.iCntsPerRev;
;           }
;       }
;
;   EncoderParm.iAccumCnt += Delta;
;
;   EncoderParm.iVelCntDwn--;
;   if(EncoderParm.iVelCntDwn)
;       return;
;
;   iVelCntDwn = iIrpPerCalc;
;   qVelMech = qKvel * iAccumCnt * 2^Nvel;
;   EncoderParm.iAccumCnt = 0;
;}
```

```
;=================== 代码 =====================
; CalcVelIrp 的寄存器使用
        .equ PosW,      w0                  ; 当前位置：POSCNT

        .equ WorkW,     w4                  ; 工作寄存器
        .equ DeltaW,    w6                  ; NewCnt - PrevCnt

        .global   _CalcVelIrp
        .global   CalcVelIrp

_CalcVelIrp:
CalcVelIrp:

    ;; 保存寄存器内容
        push            w0
        push            w4
        push            w6

    ;; Pos = uTestPos;

 .ifdef SIMU
        mov.w      _uTestPos,PosW                       ; 编码器值  ??
 .else
        mov.w      POSCNT,PosW                          ; 编码器值
 .endif

        mov.w      _EncoderParm+Encod_iPrevCnt,WorkW

    ;; 使用新的 cnt 更新前一个 cnt。
        mov.w      PosW,_EncoderParm+Encod_iPrevCnt

    ;; 计算 Delta = New - Prev
        sub.w      PosW,WorkW,DeltaW
        bra        N,jEncoder5                          ; Delta < 0

    ;; Delta > 0 或因为
    ;;      1) vel > 0 或
    ;;      2) Vel < 0 并出现翻转返回

        lsr.w      _EncoderParm+Encod_iCntsPerRev,WREG    ; WREG = CntsPerRev/2

    ;; Delta < CntsPerRev/2?
        sub.w      DeltaW,w0,WorkW                      ; Delta-CntsPerRev/2
        bra        N,jEncoder20                         ; 0 < Delta < CntsPerRev/2, Vel > 0

    ;; Delta >= CntsPerRev/2 => 负转速，出现翻转返回
    ;; Delta = Delta - CntsPerRev

        mov.w      _EncoderParm+Encod_iCntsPerRev,w0
        sub.w      DeltaW,w0,DeltaW

    ;; Delta < 0, Vel < 0
        bra        jEncoder20

 jEncoder5:
    ;; Delta < 0 或因为
    ;;       1) vel < 0 或
    ;;       2) Vel > 0 并出现翻转返回

        lsr.w      _EncoderParm+Encod_iCntsPerRev,WREG   ; WREG = CntsPerRev/2

    ;; Is Delta + CntsPerRev/2 < 0
        add.w      DeltaW,w0,WorkW  ; Delta+CntsPerRev/2
        bra        NN,jEncoder20       ; -CntsPerRev/2 <= Delta < 0, Vel > 0
```

```
    ;; Delta < -CntsPerRev/2 => 正转速，出现翻转返回
    ;; Delta = Delta + CntsPerRev

        mov.w       _EncoderParm+Encod_iCntsPerRev,w0
        add.w       DeltaW,w0,DeltaW

    ;; Delta < -CntsPerRev/2, Vel > 0

jEncoder20:

    ;; Delta 现在包含位置的有符号变化

    ;; EncoderParm.Delta += Delta;
        mov.w       DeltaW,w0
        add.w       _EncoderParm+Encod_iAccumCnt

    ;; EncoderParm.iVelCntDwn--;
    ;; if(EncoderParm.iVelCntDwn) return;

        dec.w       _EncoderParm+Encod_iVelCntDwn
        cp0.w       _EncoderParm+Encod_iVelCntDwn
        bra         NZ,jEncoder40

    ;; 重新装载 iVelCntDwn: iVelCntDwn = iIrpPerCalc;
        mov.w       _EncoderParm+Encod_iIrpPerCalc,WREG
        mov.w       WREG,_EncoderParm+Encod_iVelCntDwn

将 iAccumCnt 复制到 iDeltaCnt，然后 iAccumCnt = 0
        mov.w       _EncoderParm+Encod_iAccumCnt,DeltaW
        mov.w       DeltaW,_EncoderParm+Encod_iDeltaCnt
        clr.w       _EncoderParm+Encod_iAccumCnt

jEncoder40:

    ;; 恢复寄存器内容
        pop         w6
        pop         w4
        pop         w0
        return

;*****************************************************************
; CalcVel
;
; 根据中断程序 CalcVelIrp 产生的上一个 iDeltaCnt 值计算 qVelMech。
;
;
; 函数原型:                 void CalcVel( void );
;
; 进入时:                   EncoderParm 必须包含 iDeltaCnt 和 qKvel
;
; 退出时:                   EncoderParm 将包含 qVelMech
;
; 参数:
; 输入参数：无
; 返回值:
;   Void
;
; 所需的 SFR 设定:              无
;
; 所需的支持子程序:            无
;
; 局部堆栈使用:                无
;
; 修改的寄存器:                无
```

```
;
; 执行时间:                              大约 8 个指令周期
;
;*********************************************************************

        .global   _CalcVel
        .global   CalcVel
_CalcVel:
CalcVel:
    ;; qVelMech = qKvel * ( Delta / 2^Nvel / 2^15)


    ;; iDeltaCnt 为一个整数，但作为 Q15， iDeltaCnt=(iDeltaCnt/2^15)
        mov.w      _EncoderParm+Encod_iDeltaCnt,DeltaW
        mov.w      _EncoderParm+Encod_qKvel,WorkW

        mpy        WorkW*DeltaW,A                ; dKvel * (Delta/2^15)
        sac        A,#(Nvel-15),WorkW            ; 左移 15-Nvel 位


    ;; qVelMech = qKvel * Q15( Delta / 2^Nvel )
        mov.w      WorkW,_EncoderParm+Encod_qVelMech
        return


        .end
```

FdWeak.s

```
;*********************************************************************
; 子程序：FdWeak
;
;*********************************************************************

; 为文件中所有子程序共用

        .include "general.inc"
        .include "Control.inc"
        .include "FdWeak.inc"

;*********************************************************************
; FdWeak
;
; 说明：
;
; 公式：
;
; 定标因子：
; 函数原型：
;
; void FdWeak( void )
;
; 进入时：    FdWeakParm 结构必须包含：_FdWeakParm+FdWeak_qK1
;
; 退出时：    FdWeakParm 将包含：_CtrlParm+Ctrl_qVdRef
;
; 参数：
; 输入参数：无
;
; 返回值：
;   Void
;
; 所需的 SFR 设定：
;       CORCON.SATA             = 0
;       CORCON.IF               = 0
;
; 所需的支持子程序：无
; 局部堆栈使用：                  0
; 修改的寄存器：                  ??w4,w5,AccA
; 执行时间：                      ??8 个指令周期
;
;*********************************************************************
;
;==================== 代码 ======================
        .section                .text

;FdWeak 的寄存器使用

        .global                 _FdWeakening
        .global                 FdWeakening

_FdWeakening:
FdWeakening:

        mov.w                   _FdWeakParm+FdWeak_qK1,w0
        mov.w                   w0,_CtrlParm+Ctrl_qVdRef
        return

        .end
```

OpenLoop.s

```
;*********************************************************************
; 子程序 : OpenLoop
;*********************************************************************
; 为文件中所有子程序所共用

        .include "general.inc"
        .include "openloop.inc"
;*********************************************************************
; OpenLoop
;
; 说明:
; 公式:
;       qDeltaFlux = Kdelta * qVelMech
;       qAngFlux = qAngFlux + Kdelta * qVelMech              ;; 转子磁通角
;
;       qKdelta = (2^15) * 2 * iPoles * fLoopPeriod * fScaleMechRPS
;           其中 qVelMech 为经过 fScaleMechRPS 标定后的机械速度，单位为 RPS
;           需要 iPoles 来从机械转速获得磁通转速，
;           2 用来将 +/- 2*pi 换算为 +/- pi
; 函数原型:
;
; void OpenLoop( void )
;
; 进入时:    OpenLoopParm 结构必须包含
;
; 退出时:    OpenLoopParm 将包含
;
; 参数:
; 输入参数:                     无
;
; 返回值:
;   Void
;
; 所需的 SFR 设定:
;       CORCON.SATA             = 0
;       CORCON.IF               = 0
;
; 所需的支持子程序:             无
; 局部堆栈使用:                 0
; 修改的寄存器:                 ??w4,w5,AccA
; 执行时间:                     ??8 个指令周期
;*********************************************************************
;
;=================== 代码 =====================
        .section  .text

; OpenLoop 的寄存器使用

        .equ Work0W,    w4                      ; 工作寄存器
        .equ Work1W,    w5                      ; 工作寄存器

        .global         _OpenLoop
        .global         OpenLoop
```

```
_OpenLoop:
OpenLoop:
        mov.w           _OpenLoopParm+OpLoop_qVelMech,Work0W
        mov.w           _OpenLoopParm+OpLoop_qKdelta,Work1W
        mpy             Work0W*Work1W,A
        sac             A,Work0W
        mov.w           Work0W,_OpenLoopParm+OpLoop_qDeltaFlux

    ;; qAngFlux = qAngFlux + qDeltaFlux
        mov.w           OpenLoopParm+OpLoop_qAngFlux,Work1W
        add.w           Work0W,Work1W,Work0W
        mov.w           Work0W,_OpenLoopParm+OpLoop_qAngFlux
        return

;*********************************************************************
; void InitOpenLoop(void)
;          对 OpenLoop 中的局部变量进行初始化。
;*********************************************************************

; InitOpenLoop 的寄存器使用


;*********************************************************************

        .global         _InitOpenLoop
        .global         InitOpenLoop
_InitOpenLoop:
InitOpenLoop:

        clr.w           _OpenLoopParm+OpLoop_qAngFlux
        clr.w           _OpenLoopParm+OpLoop_qDeltaFlux
        return

        .end
```

PI.s

```
;*********************************************************************
; PI
;
; 说明：计算 PI 校正。
;
;void CalcPI( tPIParm *pParm)
;{
;   Err  = InRef - InMeas
;   U  = Sum + Kp * Err
;   if( U > Outmax )
;       Out = Outmax
;   else if( U < Outmin )
;       Out = Outmin
;   else
;       Out = U
;   Exc = U - Out
;   Sum = Sum + Ki * Err - Kc * Exc
;}
;
;void InitPI( tPIParm *pParm)
;{
;   Sum = 0
;   Out = 0
;}
;
;-----------------------------
; PI 常数的表示：
; 用 2 的幂调整常数 Kp，将其定标为 1.15 格式表示。
; 当计算完成时，去除这种调整。
;
;
; Kp 定标为：Kp = qKp * 2^NKo
;
; Ki 和 Kc 定标为：Ki = qKi, Kc = qKc
;
;
; 函数原型：
;
; void InitPI( tPIParm *pParm)
; void CalcPI( tPIParm *pParm)
;
; 进入时：    PIParm 结构必须包含 qKp、 qKi、 qKc、 qOutMax、 qOutMin、
;           InRef 和 InMeas
; 退出时：    PIParm 将包含 qOut
;
; 参数：
; 输入参数：tPIParm *pParm
;
; 返回值：
;   Void
;
; 所需的 SFR 设定：
;       CORCON.SATA= 0
;       CORCON.IF  = 0
;
; 所需的支持子程序：              无
; 局部堆栈使用：                  0
; 修改的寄存器：                  w0-w6， AccA
;
; 执行时间：
;  最大 31 个指令周期， 最小 28 个指令周期
;*********************************************************************
```

```
;
        .include "general.inc"

; 外部引用
        .include "PI.inc"

; 寄存器使用

        .equ BaseW0,    w0                      ; parm 结构的基址

        .equ OutW1,     w1                      ; 输出
        .equ SumLW2,    w2                      ; 积分和
        .equ SumHW3,    w3                      ; 积分和

        .equ ErrW4,     w4                      ; 误差项：InRef-InMeas
        .equ WorkW5,    w5                      ; 工作寄存器
        .equ Unlimit    W6,w6                   ; U：无限制输出
        .equ WorkW7,    w7                      ; 工作寄存器
;=================== 代码 =====================

        .section    .text

        .global     _InitPI
        .global     InitPI
_InitPI:
InitPI:
        mov.w       w1,[BaseW0+PI_qOut]
            return

        .global     _CalcPI
        .global     CalcPI

_CalcPI:
CalcPI:
    ;; Err  = InRef - InMeas

        mov.w       [BaseW0+PI_qInRef],WorkW7
        mov.w       [BaseW0+PI_qInMeas],WorkW5
        sub.w       WorkW7,WorkW5,ErrW4

    ;; U  = Sum + Kp * Err * 2^NKo
        lac         [++BaseW0],B                ; AccB = Sum
        mov.w       [--BaseW0],WorkW5
        mov.w       WorkW5,ACCBLL

        mov.w       [BaseW0+PI_qKp],WorkW5
        mpy         ErrW4*WorkW5,A
        sftac       A,#-NKo                     ; AccA = Kp*Err*2^NKo
        add         A                           ; Sum = Sum + Kp*Err*2^NKo
        sac         A,UnlimitW6                 ; 在测试前存储 U

    ;; if( U > Outmax )
    ;;  Out = Outmax
    ;; else if( U < Outmin )
    ;;  Out = Outmin
    ;; else
    ;;  Out = U

        mov.w       [BaseW0+PI_qOutMax],OutW1
        cp          UnlimitW6,OutW1
        bra         GT,jPI5                     ; U > Outmax; OutW1 = Outmax
```

```
        mov.w       [BaseW0+PI_qOutMin],OutW1
        cp          UnlimitW6,OutW1
        bra         LE,jPI5                     ; U < Outmin； OutW1 = Outmin

        mov.w       UnlimitW6,OutW1             ; OutW1 = U
jPI5:
        mov.w   OutW1,[BaseW0+PI_qOut]

    ;; Ki * Err
        mov.w       [BaseW0+PI_qKi],WorkW5
        mpy         ErrW4*WorkW5,A

    ;; Exc = U - Out
        sub.w       UnlimitW6,OutW1,UnlimitW6

    ;; Ki * Err - Kc * Exc
        mov.w       [BaseW0+PI_qKc],WorkW5
        msc         WorkW5*UnlimitW6,A

    ;; Sum = Sum + Ki * Err - Kc * Exc
        add         A

        sac         A,[++BaseW0]                ; 存储 Sum
        mov.w       ACCALL,WorkW5
        mov.w       WorkW5,[--BaseW0]
        return

        .end
```

ReadADC0.s

```
;*********************************************************************
; ReadADC0 和 ReadSignedADC0
;
; 说明：
;   读 ADC 的通道 0，使用 qK 对其进行换算，并将结果存放到 qADValue。
;   在转换结束之前不要调用该子程序。
;
;   ReadADC0 范围是 qK*(0.0 ->0.9999)。
;   ReadSignedADC0 范围是 qK*(-1.0 ->0.9999)。
;
;   换算常数 qK，必须在其他代码中设定，使得
;         iResult = 2 * qK * ADCBUF0
; 设定因子为 2 以允许 qK 采用 1.15 格式表示。
;
;
; 函数原型：
;
; void ReadADC0( tReadADCParm* pParm )            : 计算无符号值 0 -> 2*qK
; void ReadSignedADC0( tReadADCParm* pParm )      : 计算有符号值 -2*qK -> 2*qK
;
; 进入时：          ReadADCParm 结构必须包含 qK。 ADC 通道 0 必须包含有符号小数值。
;
; 退出时：          ReadADCParm 将包含 qADValue
;
; 参数：
; 输入参数：        无
;
; 返回值：
;    Void
;
; 所需的 SFR 设定：
;         CORCON.SATA     = 0
;    如果累加器有可能溢出，必须设定
;         CORCON.SATDW    = 1
;
; 所需的支持子程序：无
; 局部堆栈使用：无
; 修改的寄存器：w0,w4,w5
; 执行时间：13 个周期
;
;*********************************************************************
;
        .include "general.inc"

; 外部引用
        .include "ReadADC.inc"

; 寄存器使用
        .equ ParmBaseW,w0   ; parm 结构基址
        .equ Work0W,    w4
        .equ Work1W,    w5

;=================== 代码 =====================

        .section    .text
        .global     _ReadADC0
        .global     ReadADC0
```

```
_ReadADC0:
ReadADC0:

    ;; iResult = 2 * qK * ADCBUF0

        mov.w       [ParmBaseW+ADC_qK],Work0W
        mov.w       _ADCBUF0,Work1W

    ;; 从有符号小数变为无符号小数，即，将
    ;; -1->.9999 转换为 0 -> 0.9999
        btg         Work1W,#15
        lsr.w       Work1W,Work1W

        mpy         Work0W*Work1W,A
        sac         A,#-1,Work0W
        mov.w       Work0W,[ParmBaseW+ADC_qADValue]
        return


        .global     _ReadSignedADC0
        .global     ReadSignedADC0

_ReadSignedADC0:
ReadSignedADC0:

    ;; iResult = 2 * qK * ADCBUF0

        mov.w       [ParmBaseW+ADC_qK],Work0W
        mov.w       _ADCBUF0,Work1W

        mpy         Work0W*Work1W,A
        sac         A,#-1,Work0W
        mov.w       Work0W,[ParmBaseW+ADC_qADValue]
        return

        .end
```

SVGen.s

```
;*****************************************************************
; SVGen
;
; 说明：计算和装载 SVGen  PWM 值。
;
; 函数原型：
;   void CalcSVGen( void )
;
; 进入时：SVGenParm 结构必须包含 qVr1、 qVr2 和 qVr3
; 退出时：PWM 寄存器已装载
;
; 参数：
;   输入参数：
;       无
;   返回值：
;       Void
;   所需的 SFR 设定：
;       CORCON.SATA  = 0
;       CORCON.IF    = 0
;   所需的支持子程序：
;       无
;   局部堆栈使用：
;       0
;   修改的寄存器：
;       w0, w2, w3, w4, w5, w6, AccA
;   执行时间：
;       34 个指令周期
;*****************************************************************
; C 版本代码
; void CalcRefVec( void )
; {
;   if( Vr1 >= 0 )
;       {
;       // (xx1)
;       if( Vr2 >= 0 )
;           {
;           // (x11)
;           // 由于不可能采用 Sector 7，因此一定是 Sector 3
;           // Sector 3: (0,1,1)  0-60 电角度
;           T1 = Vr2
;           T2 = Vr1
;           CalcTimes();
;           dPWM1 = Ta
;           dPWM2 = Tb
;           dPWM3 = Tc
;           }
;       else
;           {
;           // (x01)
;           if( Vr3 >= 0 )
;               {
;               // Sector 5: (1,0,1)  120-180 电角度
;               T1 = Vr1
;               T2 = Vr3
;               CalcTimes();
;               dPWM1 = Tc
;               dPWM2 = Ta
;               dPWM3 = Tb
;               }
```

```
;           else
;               {
;               // Sector 1: (0,0,1)  60-120 电角度
;               T1 = -Vr2;
;               T2 = -Vr3;
;               CalcTimes();
;               dPWM1 = Tb
;               dPWM2 = Ta
;               dPWM3 = Tc
;               }
;           }
;       }
;   else
;       {
;       // (xx0)
;       if( Vr2 >= 0 )
;           {
;           // (x10)
;           if( Vr3 >= 0 )
;               {
;               // Sector 6: (1,1,0)  240-300 电角度
;               T1 = Vr3
;               T2 = Vr2
;               CalcTimes();
;               dPWM1 = Tb
;               dPWM2 = Tc
;               dPWM3 = Ta
;               }
;           else
;               {
;               // Sector 2: (0,1,0)  300-0 电角度
;               T1 = -Vr3
;               T2 = -Vr1
;               CalcTimes();
;               dPWM1 = Ta
;               dPWM2 = Tc
;               dPWM3 = Tb
;               }
;           }
;       else
;           {
;           // (x00)
;           // 由于不可能采用 Sector 0，因此一定是 Sector 4。
;           // Sector 4: (1,0,0)  180-240 电角度
;           T1 = -Vr1
;           T2 = -Vr2
;           CalcTimes();
;           dPWM1 = Tc
;           dPWM2 = Tb
;           dPWM3 = Ta
;
;           }
;       }
; }
;
; void CalcTimes(void)
; {
;   T1 = PWM*T1
;   T2 = PWM*T2
;   Tc = (PWM-T1-T2)/2
;   Tb = Ta + T1
;   Ta = Tb + T2
; }
;*****************************************************************
;
```

```
        .include "general.inc"

; 外部引用
        .include "Park.inc"
        .include "SVGen.inc"
        .include "CurModel.inc"
; 寄存器使用
        .equ WorkW,         w1                      ; 工作寄存器
        .equ T1W,           w2
        .equ T2W,           w3

        .equ WorkDLoW,      w4                      ; 双字（乘法结果）
        .equ Vr1W,          w4
        .equ TaW,           w4
        .equ WorkDHiW,      w5                      ; 双字（乘法结果）
        .equ Vr2W,          w5
        .equ TbW,           w5
        .equ Vr3W,          w6
        .equ TcW,           w6

        .equ dPWM1,         PDC1
        .equ dPWM2,         PDC2
        .equ dPWM3,         PDC3
;=================== 代码 =====================

        .section            .text
        .global             _CalcSVGen
        .global             CalcSVGen

 _CalcSVGen:
 CalcSVGen:
    ;; 获得 qVr1、 qVr2 和 qVr3
        mov.w               _SVGenParm+SVGen_qVr1,Vr1W
        mov.w               _SVGenParm+SVGen_qVr2,Vr2W
        mov.w               _SVGenParm+SVGen_qVr3,Vr3W
    ;; 测试 Vr1
        cp0                 Vr1W
        bra                 LT,jCalcRef20           ; Vr1W < 0
    ;; 测试 Vr2
        cp0                 Vr2W
        bra                 LT,jCalcRef10           ; Vr2W < 0
    ;; 由于不可能采用 Sector 7，因此一定是 Sector 3。
    ;; Sector 3: (0,1,1)  0-60 电角度
    ;; T1 = Vr2
    ;; T2 = Vr1
        mov.w               Vr2W,T2W
        mov.w               Vr1W,T1W
        rcall               CalcTimes
    ;; dPWM1 = Ta
    ;; dPWM2 = Tb
    ;; dPWM3 = Tc
        mov.w           TaW,dPWM1
        mov.w           TbW,dPWM2
        mov.w           TcW,dPWM3
        return
```

```
jCalcRef10:
   ;; 测试 Vr3
       cp0             Vr3W
       bra             LT,jCalcRef15          ; Vr3W < 0
   ;; Sector 5: (1,0,1)  120-180 电角度
   ;; T1 = Vr1
   ;; T2 = Vr3
       mov.w           Vr1W,T2W
       mov.w           Vr3W,T1W
       rcall           CalcTimes
   ;; dPWM1 = Tc
   ;; dPWM2 = Ta
   ;; dPWM3 = Tb
       mov.w           TcW,dPWM1
       mov.w           TaW,dPWM2
       mov.w           TbW,dPWM3
       return

jCalcRef15:
   ;; Sector 1: (0,0,1)  60-120 电角度
   ;; T1 = -Vr2
   ;; T2 = -Vr3
       neg.w           Vr2W,T2W
       neg.w           Vr3W,T1W
       rcall           CalcTimes
   ;; dPWM1 = Tb
   ;; dPWM2 = Ta
   ;; dPWM3 = Tc
       mov.w           TbW,dPWM1
       mov.w           TaW,dPWM2
       mov.w           TcW,dPWM3
       return

jCalcRef20:
   ;; 测试 Vr2
       cp0             Vr2W
       bra             LT,jCalcRef30          ; Vr2W < 0
   ;; Test Vr3
       cp0             Vr3W
       bra             LT,jCalcRef25          ; Vr3W < 0
   ;; Sector 6: (1,1,0)  240-300 电角度
   ;; T1 = Vr3
   ;; T2 = Vr2
       mov.w           Vr3W,T2W
       mov.w           Vr2W,T1W
       rcall           CalcTimes
   ;; dPWM1 = Tb
   ;; dPWM2 = Tc
   ;; dPWM3 = Ta
       mov.w           TbW,dPWM1
       mov.w           TcW,dPWM2
       mov.w           TaW,dPWM3
       return
jCalcRef25:
   ;; Sector 2: (0,1,0)  300-360 电角度
   ;; T1 = -Vr3
   ;; T2 = -Vr1
       neg.w           Vr3W,T2W
       neg.w           Vr1W,T1W
       rcall           CalcTimes

   ;; dPWM1 = Ta
   ;; dPWM2 = Tc
   ;; dPWM3 = Tb
       mov.w               TaW,dPWM1
```

```
        mov.w               TcW,dPWM2
        mov.w               TbW,dPWM3
        return
 jCalcRef30:
    ;; 由于不可能是 Sector 0，因此一定是 Sector 4
    ;; Sector 4: (1,0,0)  180-240 电角度
    ;; T1 = -Vr1
    ;; T2 = -Vr2
        neg.w               Vr1W,T2W
        neg.w               Vr2W,T1W
        rcall               CalcTimes
    ;; dPWM1 = Tc
    ;; dPWM2 = Tb
    ;; dPWM3 = Ta
        mov.w               TcW,dPWM1
        mov.w               TbW,dPWM2
        mov.w               TaW,dPWM3
        return
;*********************************************************************
; CalcTimes
;
; void CalcTimes(void)
; {
;   T1 = PWM*T1
;   T2 = PWM*T2
;   Tc = (PWM-T1-T2)/2
;   Tb = Ta + T1
;   Ta = Tb + T2
; }
;
; 执行时间：17 个指令周期
;*********************************************************************
 CalcTimes:

    ;; T1 = PWM*T1
    ;; 由于 T1 为 1.15 格式而 PWM 采用整数格式，我们按照整数格式进行乘法
    ;; 2*PWM*T1，并使用结果的高位字
    ;; 装载 PWMPeriod
        sl.w                _SVGenParm+SVGen_iPWMPeriod,WREG  ; 进行乘法 PWM * 2 以允许
                                                                电压满量程
        mul.us              w0,T1W,WorkDLoW
        mov.w               WorkDHiW,T1W
    ;; T2 = PWM*T2
        mul.us              w0,T2W,WorkDLoW
        mov.w               WorkDHiW,T2W
    ;; Tc = (PWM-T1-T2)/2
        ;mov.w              _SVGenParm+SVGen_iPWMPeriod,WorkW
        mov.w               _SVGenParm+SVGen_iPWMPeriod,WREG
        sub.w               w0,T1W,WorkW                      ;PWM-T1
        sub.w               WorkW,T2W,WorkW                   ; -T2
        asr.w               WorkW,WorkW                       ; /2
        mov.w               WorkW,TcW                         ; 存储 Tc
    ;; Tb = Tc + T1
        add.w               WorkW,T1W,WorkW
        mov.w               WorkW,TbW
    ;; Ta = Tb + T2
        add.w               WorkW,T2W,WorkW
        mov.w               WorkW,TaW
        return
```

Trig.s

```
;*********************************************************************
; Trig
;
; 说明：
; 使用基于 128 字查找表的线性插值法计算指定角度的正弦和余弦值。
;
;
; 该子程序同样适用于整数定标和 1.15 定标格式。
;
; 对于整数定标格式，定标后， 0 <= 角度 < 2*pi
; 对应于 0 <= 角度 < 0xFFFF。最终的正弦和余弦返回值
; 定标为 -32769 -> 32767，即 (0x8000 -> 0x7FFF)。
;
; 对于 1.15 定标格式，定标后， -pi <= 角度 < pi
; 对应于 -1 -> 0.9999，即 (0x8000 <= 角度 < 0x7FFF)。最终的
; 正弦和余弦返回值定标为 -1 -> 0.9999，
; 即 (0x8000 -> 0x7FFF)。
;
; 函数原型：
;   void SinCos( void )
;
;   进入时：     ParkParm 结构必须包含 qAngle
;   退出时：     ParkParm 将包含 qSin 和 qCos。 qAngle 不变。
;
; 参数：
;   输入参数：
;       无
;   返回值：
;       Void
;   所需的 SFR 设定：
;       CORCON.IF = 0
;   所需的支持子程序：
;       无
;   局部堆栈使用：
;       0
;   修改的寄存器：
;       w0-w7
;   执行时间：
;       大约 28 个指令周期
;*********************************************************************
;
        .include "general.inc"

;   外部引用
        .include "park.inc"
;   常量
        .equ TableSize,128
;   局部寄存器使用
        .equ Work0W,                w0              ; 工作寄存器
        .equ Work1W,                w1              ; 工作寄存器
        .equ RemainderW,            w2              ; 插值的小数：0->0xFFFF
        .equ IndexW,                w3              ; 表的索引
        .equ pTabPtrW,              w4              ; 指向表的指针
        .equ pTabBaseW,             w5              ; 指向表基址的指针
        .equ Y0W,                   w6              ; Y0 = SinTable[Index]
        .equ ParkParmW,             w7              ; ParkParm 结构的基址

    ;; 注：RemainderW 和 Work0W 必须是偶数寄存器

;=================== 局部数据 ====================

        .section .ndata, "d"
 SinTable(正弦表):
```

```
    .word 0,1608,3212,4808,6393,7962,9512,11039
    .word 12540,14010,15446,16846,18205,19520,20787,22005
    .word 23170,24279,25330,26319,27245,28106,28898,29621
    .word 30273,30852,31357,31785,32138,32413,32610,32728
    .word 32767,32728,32610,32413,32138,31785,31357,30852
    .word 30273,29621,28898,28106,27245,26319,25330,24279
    .word 23170,22005,20787,19520,18205,16846,15446,14010
    .word 12540,11039,9512,7962,6393,4808,3212,1608
    .word 0,-1608,-3212,-4808,-6393,-7962,-9512,-11039
    .word -12540,-14010,-15446,-16846,-18205,-19520,-20787,-22005
    .word -23170,-24279,-25330,-26319,-27245,-28106,-28898,-29621
    .word -30273,-30852,-31357,-31785,-32138,-32413,-32610,-32728
    .word -32767,-32728,-32610,-32413,-32138,-31785,-31357,-30852
    .word -30273,-29621,-28898,-28106,-27245,-26319,-25330,-24279
    .word -23170,-22005,-20787,-19520,-18205,-16846,-15446,-14010
    .word -12540,-11039,-9512,-7962,-6393,-4808,-3212,-1608

;==================== 代码 ====================
        .section            .text
        .global             _SinCos
        .global             SinCos

_SinCos:
SinCos:
    ;; ParkParm 结构中 qAngle、 qSin 和 qCos 组的基址
        mov.w               #_ParkParm+#Park_qAngle,ParkParmW

    ;; 计算用于 Sin 的预取和插值的索引和余数
        mov.w               #TableSize,Work0W
        mov.w               [ParkParmW++],Work1W        ; 装载 qAngle，并将指针递增 1 指向 qCos
        mul.uu              Work0W,Work1W,RemainderW    ; IndexW 中的高位字

    ;; 由于偏移量以字节为单位，而不是以字为单位，因此索引值加倍
        add.w               IndexW,IndexW,IndexW

    ;; 注意此时 IndexW 寄存器具有值 0x00nn，其中 nn
    ;; 为对 TabBase（表基址）的偏移量 （以字节为单位）。如果下面我们总是对 IndexW 寄存器使用字节操作，
    ;; 对于 128 字大小的查找表，程序将自动实现正确的返回翻转。
    ;;

        mov.w               #SinTable,pTabBaseW         ; 指向表基址的指针

    ;; 检查余数是否为零
        cp0.w               RemainderW
        bra                 nz,jInterpolate

    ;; 余数为零允许我们跳过插值操作，直接使用表中的值
    ;;

        add.w               IndexW,pTabBaseW,pTabPtrW
        mov.w               [pTabPtrW],[ParkParmW++]    ; 写 qSin，并使指针递增 1 指向 qCos

    ;; 在 Sin 索引上加上 0x40 得到 Cos 索引。这样可能超出表的末尾。
    ;; 但是如果我们仅使用字节操作，则将会实现自动的翻转返回。
        add.b               #0x40,IndexW
        add.w               IndexW,pTabBaseW,pTabPtrW
        mov.w               [pTabPtrW],[ParkParmW]      ; 写 qCos
        return

jInterpolate:

    ;; 获得 Y1-Y0 = SinTable[Index+1] - SinTable[Index]
        add.w               IndexW,pTabBaseW,pTabPtrW
        mov.w               [pTabPtrW],Y0W              ; Y0
        inc2.b              IndexW,IndexW               ; (Index += 2)&0xFF
```

```
    add.w               IndexW,pTabBaseW,pTabPtrW
    subr.w              Y0W,[pTabPtrW],Work0W      ; Y1 - Y0

;; 计算 Delta = (Remainder*(Y1-Y0)) >> 16
    mul.us              RemainderW,Work0W,Work0W

;; Work1W 包含 (Remainder*(Y1-Y0)) 的高位字
;; *pSin = Y0 + Delta
    add.w               Work1W,Y0W,[ParkParmW++]   ; 写 qSin，并将指针递增 1 指向 qCos

;; ================= COS ========================

;; 将 Sin 索引加上 0x40 得到 Cos 索引。这样可能超出表的末尾。
;; 但是如果我们仅使用字节操作，将会实现自动的翻转返回。
;; 实际上只加上了 0x3E，这是因为在前面索引递增了 2。
    add.b               #0x3E,IndexW
    add.w               IndexW,pTabBaseW,pTabPtrW

;; 获得 Y1-Y0 = SinTable[Index+1] - SinTable[Index]
    add.w               IndexW,pTabBaseW,pTabPtrW
    mov.w               [pTabPtrW],Y0W             ; Y0

    inc2.b              IndexW,IndexW          ; (Index += 2)&0xFF
    add.w               IndexW,pTabBaseW,pTabPtrW
    subr.w              Y0W,[pTabPtrW],Work0W  ; Y1 - Y0

;; 计算 Delta = (Remainder*(Y1-Y0)) >> 16
    mul.us              RemainderW,Work0W,Work0W

;; Work1W 包含 (Remainder*(Y1-Y0)) 的高位字
;; *pCos = Y0 + Delta
    add.w               Work1W,Y0W,[ParkParmW] ; 写 qCos
     return
    .end
```

版本历史

版本 A（2005 年 6 月）

本文档的初始版本。

版本 B（2007 年 10 月）

此版本改正了图 3 中的第二个公式。

注：

请注意以下有关 Microchip 器件代码保护功能的要点：

- Microchip 的产品均达到 Microchip 数据手册中所述的技术指标。
- Microchip 确信：在正常使用的情况下， Microchip 系列产品是当今市场上同类产品中最安全的产品之一。
- 目前，仍存在着恶意、甚至是非法破坏代码保护功能的行为。就我们所知，所有这些行为都不是以 Microchip 数据手册中规定的操作规范来使用 Microchip 产品的。这样做的人极可能侵犯了知识产权。
- Microchip 愿与那些注重代码完整性的客户合作。
- Microchip 或任何其他半导体厂商均无法保证其代码的安全性。代码保护并不意味着我们保证产品是 “牢不可破”的。

代码保护功能处于持续发展中。 Microchip 承诺将不断改进产品的代码保护功能。任何试图破坏 Microchip 代码保护功能的行为均可视为违反了《数字器件千年版权法案（Digital Millennium Copyright Act）》。如果这种行为导致他人在未经授权的情况下，能访问您的软件或其他受版权保护的成果，您有权依据该法案提起诉讼，从而制止这种行为。

ISBN：978-1-60932-708-8

QUALITY MANAGEMENT SYSTEM
CERTIFIED BY DNV
ISO/TS 16949:2002

Microchip 位于美国亚利桑那州 Chandler 和 Tempe 与位于俄勒冈州 Gresham 的全球总部、设计和晶圆生产厂及位于美国加利福尼亚州和印度的设计中心均通过了 ISO/TS-16949:2002 认证。公司在 PIC® MCU 与 dsPIC® DSC、KEELOQ® 跳码器件、串行 EEPROM、单片机外设、非易失性存储器和模拟产品方面的质量体系流程均符合 ISO/TS-16949:2002。此外， Microchip 在开发系统的设计和生产方面的质量体系也已通过了 ISO 9001:2000 认证。

全球销售及服务网点

美洲

公司总部 Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 1-480-792-7200
Fax: 1-480-792-7277
技术支持：
http://support.microchip.com
网址：www.microchip.com

亚特兰大 Atlanta
Duluth, GA
Tel: 1-678-957-9614
Fax: 1-678-957-1455

波士顿 Boston
Westborough, MA
Tel: 1-774-760-0087
Fax: 1-774-760-0088

芝加哥 Chicago
Itasca, IL
Tel: 1-630-285-0071
Fax: 1-630-285-0075

克里夫兰 Cleveland
Independence, OH
Tel: 1-216-447-0464
Fax: 1-216-447-0643

达拉斯 Dallas
Addison, TX
Tel: 1-972-818-7423
Fax: 1-972-818-2924

底特律 Detroit
Farmington Hills, MI
Tel: 1-248-538-2250
Fax: 1-248-538-2260

科科莫 Kokomo
Kokomo, IN
Tel: 1-765-864-8360
Fax: 1-765-864-8387

洛杉矶 Los Angeles
Mission Viejo, CA
Tel: 1-949-462-9523
Fax: 1-949-462-9608

圣克拉拉 Santa Clara
Santa Clara, CA
Tel: 1-408-961-6444
Fax: 1-408-961-6445

加拿大多伦多 Toronto
Mississauga, Ontario,
Canada
Tel: 1-905-673-0699
Fax: 1-905-673-6509

亚太地区

亚太总部 Asia Pacific Office
Suites 3707-14, 37th Floor
Tower 6, The Gateway
Harbour City, Kowloon
Hong Kong
Tel: 852-2401-1200
Fax: 852-2401-3431

中国 – 北京
Tel: 86-10-8528-2100
Fax: 86-10-8528-2104

中国 – 成都
Tel: 86-28-8665-5511
Fax: 86-28-8665-7889

中国 – 重庆
Tel: 86-23-8980-9588
Fax: 86-23-8980-9500

中国 – 香港特别行政区
Tel: 852-2401-1200
Fax: 852-2401-3431

中国 – 南京
Tel: 86-25-8473-2460
Fax: 86-25-8473-2470

中国 – 青岛
Tel: 86-532-8502-7355
Fax: 86-532-8502-7205

中国 – 上海
Tel: 86-21-5407-5533
Fax: 86-21-5407-5066

中国 – 沈阳
Tel: 86-24-2334-2829
Fax: 86-24-2334-2393

中国 – 深圳
Tel: 86-755-8203-2660
Fax: 86-755-8203-1760

中国 – 武汉
Tel: 86-27-5980-5300
Fax: 86-27-5980-5118

中国 – 西安
Tel: 86-29-8833-7252
Fax: 86-29-8833-7256

中国 – 厦门
Tel: 86-592-238-8138
Fax: 86-592-238-8130

中国 – 珠海
Tel: 86-756-321-0040
Fax: 86-756-321-0049

台湾地区 – 高雄
Tel: 886-7-213-7830
Fax: 886-7-330-9305

台湾地区 – 台北
Tel: 886-2-2500-6610
Fax: 886-2-2508-0102

亚太地区

台湾地区 – 新竹
Tel: 886-3-6578-300
Fax: 886-3-6578-370

澳大利亚 Australia - Sydney
Tel: 61-2-9868-6733
Fax: 61-2-9868-6755

印度 India - Bangalore
Tel: 91-80-3090-4444
Fax: 91-80-3090-4123

印度 India - New Delhi
Tel: 91-11-4160-8631
Fax: 91-11-4160-8632

印度 India - Pune
Tel: 91-20-2566-1512
Fax: 91-20-2566-1513

日本 Japan - Yokohama
Tel: 81-45-471- 6166
Fax: 81-45-471-6122

韩国 Korea - Daegu
Tel: 82-53-744-4301
Fax: 82-53-744-4302

韩国 Korea - Seoul
Tel: 82-2-554-7200
Fax: 82-2-558-5932 或
82-2-558-5934

马来西亚 Malaysia - Kuala Lumpur
Tel: 60-3-6201-9857
Fax: 60-3-6201-9859

马来西亚 Malaysia - Penang
Tel: 60-4-227-8870
Fax: 60-4-227-4068

菲律宾 Philippines - Manila
Tel: 63-2-634-9065
Fax: 63-2-634-9069

新加坡 Singapore
Tel: 65-6334-8870
Fax: 65-6334-8850

泰国 Thailand - Bangkok
Tel: 66-2-694-1351
Fax: 66-2-694-1350

欧洲

奥地利 Austria - Wels
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

丹麦 Denmark-Copenhagen
Tel: 45-4450-2828
Fax: 45-4485-2829

法国 France - Paris
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

德国 Germany - Munich
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

意大利 Italy - Milan
Tel: 39-0331-742611
Fax: 39-0331-466781

荷兰 Netherlands - Drunen
Tel: 31-416-690399
Fax: 31-416-690340

西班牙 Spain - Madrid
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

英国 UK - Wokingham
Tel: 44-118-921-5869
Fax: 44-118-921-5820

07/15/10