Application Report
SPRA588

Implementation of a Speed Field Oriented Control of 3-phase PMSM Motor using TMS320F240

Erwan Simon *Digital Control Systems*

Abstract

This application report presents a solution to control a 3-phase Permanent Magnet Synchronous motor using the Texas Instruments (TI™) TMS320F240 digital signal processor (DSP). This processor is part of a new family of DSPs that enable cost-effective design of intelligent controllers for brushless motors. The use of this DSP yields enhanced operations, fewer system components, lower system cost and increased efficiency. The control method presented is field oriented control (FOC). The sinusoidal voltage waveforms are generated by the DSP using the space vector modulation technique. A practical solution is described and results are given in this application report.

Contents

Figures

Tables

Introduction

A brushless Permanent Magnet Synchronous motor (PMSM) has a wound stator, a permanent magnet rotor assembly and internal or external devices to sense rotor position. The sensing devices provide logic signals for electronically switching the stator windings in the proper sequence to maintain rotation of the magnet assembly. The combination of an inner permanent magnet rotor and outer windings offers the advantages of low rotor inertia, efficient heat dissipation, and reduction of the motor size. Moreover, the elimination of brushes reduces noise, EMI generation and suppresses the need of brushes maintenance.

Two configurations of permanent magnet brushless motor are usually considered: the trapezoidal type and the sinusoidal type. Depending on how the stator is wounded, the back-electromagnetic force will have a different shape (the BEMF is induced in the stator by the motion of the rotor). To obtain the maximum performance from each type of PMSM, an appropriate control strategy has to be implemented. The trapezoidal BEMF motor called DC brushless motor (BLDC) uses a "two phases on" strategy, whereas the sinusoidal BEMF motor offers its best performances when driven by sinusoidal currents (three phases on strategy).

This application report presents the implementation of a control for sinusoidal PMSM motor.

The sinusoidal voltage waveform applied to this motor is created by using the Space Vector modulation technique.

The Field Oriented Control algorithm will enable real-time control of torque and rotation speed. As this control is accurate in every mode of operation (steady state and transient), no oversize of the power transistors is necessary. The transient currents are constantly controlled in amplitude. Moreover, no torque ripple appears when driving this sinusoidal BEMF motor with sinusoidal currents.

PMSM Model

The operation of a brushless PM motor relies on the conversion of electrical energy to magnetic energy and then from magnetic energy to mechanical energy. It is possible to generate a magnetic rotating field by applying sinusoidal voltages to the 3 stator phases of a 3 phase motor. A resulting sinusoidal current flows in the coils and generates the rotating stator flux.

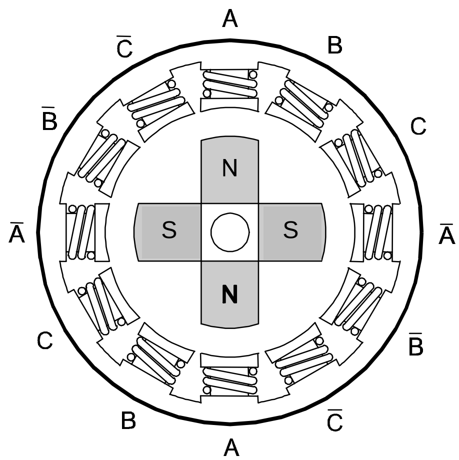
The rotation of the rotor shaft is then created by attraction of the permanent rotor flux with the stator flux.

Speed and Position Definition

In electric motors, two measures of position and speed are usually defined: mechanical and electrical. The mechanical position is related to the rotation of the rotor shaft. When the rotor shaft has accomplished 360 mechanical degrees, the rotor is back in the same position where it started.

The electrical position of the rotor is related to the rotation of the rotor magnetic field. In Figure 1, the rotor needs only to move 180 mechanical degrees to obtain an identical magnetic configuration as when it started. The electrical position of the rotor is then related to the number of magnetic pole pairs on it.

Figure 1. Three-phase Motor with 4 Magnet Poles (2 Pole Pair)



The electrical position of the rotor is linked to the mechanical position of the rotor by the relationship

$\theta_e = \theta_m * p$ (p is the number of pole pair).

As the speed is related to the position by $\omega = d\theta/dt$, a similar relationship also exists towards electrical speed and mechanical speed.

$\omega_e = \omega_m * p$

The notions of electrical position of the rotor and mechanical speed are extensively used in this report.

Electrical Equations

$$v_a = V \cos(w_e * t)$$

$$v_b = V \cos(w_e * t - \frac{2p}{3})$$

$$v_c = V \cos(w_e * t - \frac{4p}{3})$$

To create the rotating stator flux, the commonly applied phase voltages present a phase shift of 120 electrical degrees from one to another that takes into account the mechanical 120 degrees angle between coils.

A one phase electrical equation can be written like :

$$v = Z * i = Ri + \frac{d\Psi}{dt} = Ri + \frac{d}{dt}(Li + \Psi_m(q))$$

where ψ_m corresponds to the amplitude of the natural magnetic flux of the permanent magnets. The term $\frac{d}{dt}\Psi m(q)$ corresponds to the back-emf (induced voltage) and can also be written like $\frac{d\Psi m(q)}{dq} * w_e$, where ω_e corresponds to the electrical speed.

Supposing that the machine is sinusoidal, the induced voltage has the following form:

$$\overline{E} = \begin{bmatrix} E_a(q) \\ E_b(q) \\ E_c(q) \end{bmatrix} = -w_e * \Psi m \begin{bmatrix} \sin(q_e) \\ \sin(q_e - \frac{2p}{3}) \\ \sin(q_e - \frac{4p}{3}) \end{bmatrix} = w_e * \Psi m * [K(q_e)]$$

From the electrical power delivered to the motor, a part of it is transformed in Joule losses, another part is going to the energy stored in the magnetic field and the last part is transformed in mechanical energy (torque production).

In the PMSM case, the torque is expressed by:

$Te = p * [I_s]^t * \Psi m * [K(q_e)]$, where p is the number of pole pairs.

It can be proven that the best solution to produce a constant torque is to drive a sinusoidal motor by sinusoidal currents.

$$Te = p\Psi_m(I_a * K_a(q) + I_b * K_b(q) + I_c * K_c(q))$$

Knowing that :

$$I_a = I_s \sin(w_e * t)$$

$$I_b = I_s \sin(w_e * t - \frac{2p}{3})$$

$$I_c = I_s \sin(w_e * t - \frac{4p}{3})$$

We obtain

$Te = p * \Psi_m * I_s(\sin^2(wt) + \sin^2(wt - \frac{2p}{3}) + \sin^2(wt - \frac{4p}{3})) = \frac{3}{2} p * \Psi_m * I_s$. It will be further shown that the FOC enables a continuous control of the torque demand without ripples.

Mechanical Equations

The torque created by the energy conversion process is then used to drive mechanical loads. Its expression is related to mechanical parameters via the fundamental law of the dynamics as follows:

$$\sum \overline{T} = J \frac{dw}{dt}$$

Giving:

J : rotor inertia
Kd: viscosity coefficient
Tl: load torque
w_m: mechanical speed

$$J \frac{dw_m}{dt} + k_d w_m + T_l = T_e$$

As the torque is composed of time and electrical position dependent parameters, its efficient and accurate control is not easy with standard methods.

The proposed solution is to overcome this issue is based on the real time implementation of the Field Orientated Control algorithm with a TMS320F240 DSP.

FOC Control for PMSM

The goal of the Field Oriented Control [BPRA073] is to perform real-time control of torque variations demand, to control the rotor mechanical speed and to regulate phase currents in order to avoid current spikes during transient phases.

To perform these controls, the electrical equations are projected from a 3 phase non-rotating frame into a two co-ordinate rotating frame.

This mathematical projection (Clarke & Park) greatly simplifies the expression of the electrical equations and remove their time and position dependencies.

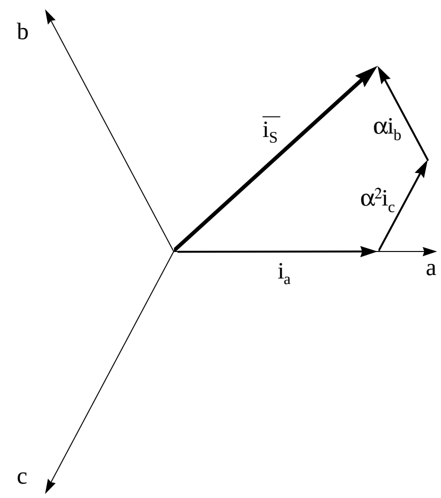
Expression of the Stator Current Vector

As phase current values are used in the general expression of the torque, the expression of their values in the new rotating frame are needed afterwards.

The three sinusoidal currents created by the 120° (electrical) phase shifted voltages applied to the stator are also 120° (electrical) phase shifted one from another.

The stator current vector (Figure 2) is represented in the 3 phase nonrotating frame (a,b,c) and defined by $i_s = i_a + e^{j2p/3} i_b + e^{j4p/3} i_c$

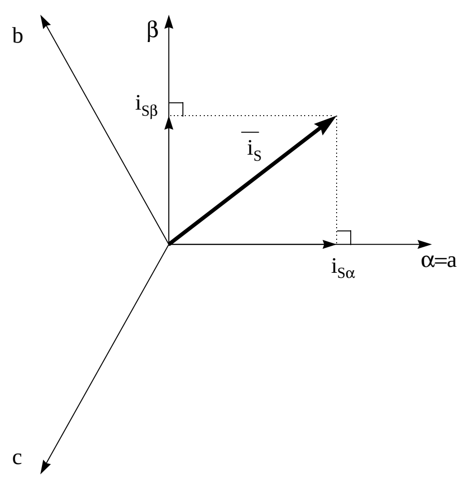
Figure 2. Stator Current Vector



The Clarke and Park Transformations

The idea of the Clarke transformation is that the rotating stator current vector that is the sum of the 3 phase currents can also be generated by a bi-phased system placed on the fixed axis α and β as shown in Figure 3.

Figure 3. (a,b,c)->(a,b) Projection (ClarkeTransformation)



The projection of the stator current vector in this fixed frame gives:

$$i_{sa} = i_a$$

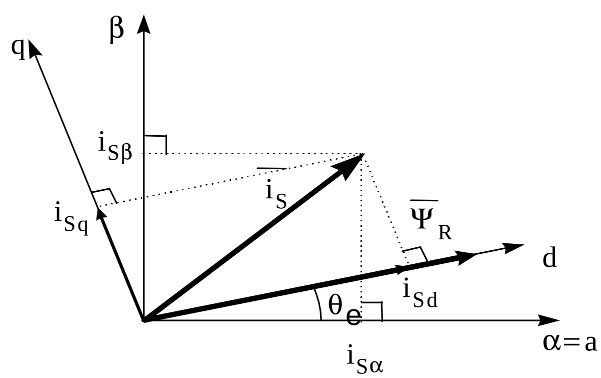
$$i_{sb} = \frac{1}{\sqrt{3}} \cdot i_a + \frac{2}{\sqrt{3}} i_b$$

$$i_a + i_b + i_c = 0$$

In this new frame, the expression of the torque is still dependent on the position of the rotor flux, preventing any easy solution of the electrical differential equation.

To remove this dependency, the electrical equations are projected in a 2-phase (d,q) system (Figure 4) that rotates at the speed of the electrical speed of the rotor and where the d axis is aligned with the electrical position of the rotor flux. In this frame, the electrical expression of the torque becomes independent from θ_e.

Figure 4. (a,b)->(d,q) Projection (Park Transformation)



The equations corresponding to this transformation are given by:

$$i_{sd} = i_{sa} \cdot \cos(q_e) + i_{sb} \cdot \sin(q_e)$$
$$i_{sq} = -i_{sa} \cdot \sin(q_e) + i_{sb} \cdot \cos(q_e)$$

In this new system, the expression of the electrical equations are greatly simplified:

$$V_{sd} = R_s * i_d + \frac{d}{dt} j_{rd} - w_e * j_{rq}$$

$$V_{sq} = R_s * i_q + \frac{d}{dt} j_{rq} + w_e * j_{rd}$$

For a multiple pole synchronous motor, the expression of the torque in (d,q) is:

$$T_e = \frac{3}{2} p \left(y_{rd} * i_{sq} - i_{sd} * y_{rq} \right)$$

Where p is the number of pole pairs.

In the specific case of a permanent magnet synchronous motor without salient poles, most of the natural magnetic flux is on the d axis ($\psi_{rd} >> \psi_{rq}$). Moreover, the stator current vector value is

$$i_s = \sqrt{i_{sd}^2 + i_{sq}^2}$$

In order to optimize the torque production for a given i_S value, the appropriate strategy is to set i_{sdref} to 0.

The action of the current regulators is then to shift the current vector Is onto the q axis.

The torque is now given by

$$T_e \propto y_{rd} * i_{sq}$$

The relationship between mechanical speed and torque is given by the mechanical differential equation.

To overcome the nominal speed limitation, a field-weakening algorithm can be implemented with a non-zero $i_{sdref.}$. Setting i_{sdref} to a non-zero value will increase the speed range but the applicable torque must be reduced to ensure that the relationship

$$i_s = \sqrt{i_{sd}^2 + i_{sq}^2} \le i_{s\max}$$

is respected. Moreover, it is not recommended to create a magnetic flux opposed to the natural flux of the permanent magnets over long periods of time. This could lead to demagnetization of the rotor magnets reducing the torque production, as well as excessive heat generation.

PMSM Control Structure

The control scheme proposed for the Speed FOC PMSM drive is shown in Figure 5.

Figure 5. PMSM Control Structure

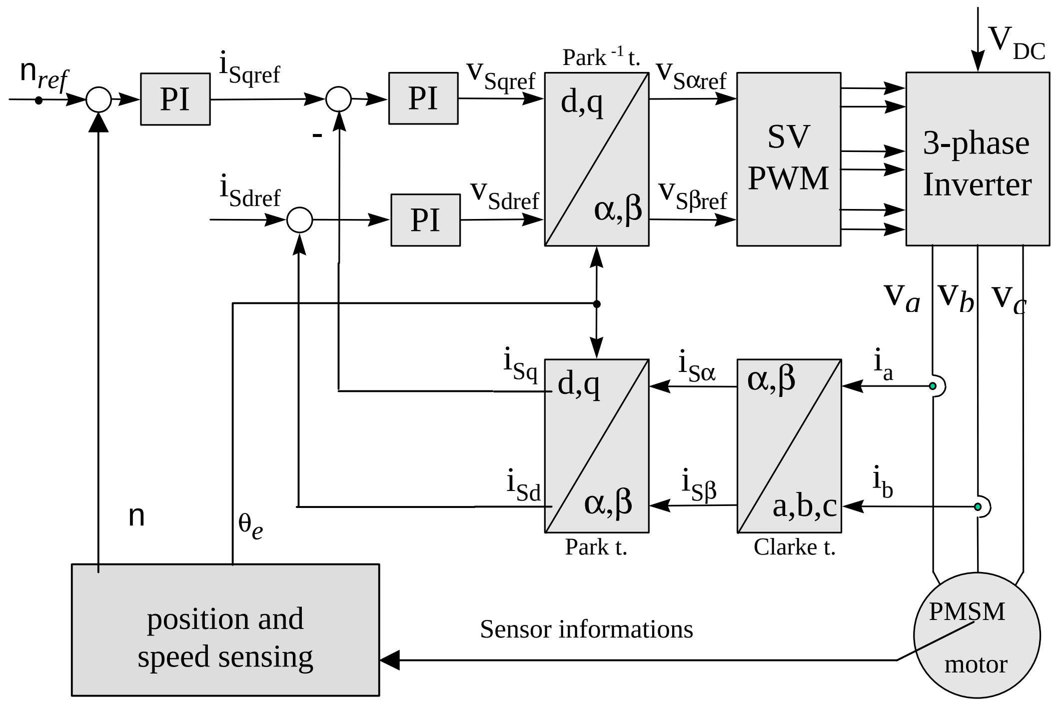


Figure 5 shows the software modules with the hardware of the solution. A detailed description of both aspects will be given in dedicated paragraphs.

i_a and i_b are measured with a current sensor. The Clarke transform is applied to them to determine the stator current projection in a two co-ordinate non-rotating frame.

The Park co-ordinate transformation is then applied in order to obtain this projection in the (d,q) rotating frame.

The (d,q) projections of the stator phase currents are then compared to their reference values I_{sqref} and I_{sdref} (set to 0) and corrected by mean of PI current controllers. The outputs of the current controllers are passed through the inverse Park transform and a new stator voltage vector is impressed to the motor using the Space Vector Modulation technique. In order to control the mechanical speed of the motor (speed FOC), an outer loop is driving the reference current I_{sqref}. The mechanical speed reference is denoted “n_{ref}“ and the mechanical speed “n” for notations compliance with previous FOC application notes.

Application Description

This chapter covers each component put in place to implement the solution of the PMSM drive. The different elements of the application are:

- 6-pole PMSM motor
- DSP development board
- Power board

Motor Characteristics

The PMSM motor used for the application is a 6-pole three phase Y-connected motor. The characteristics of this motor are as follow:

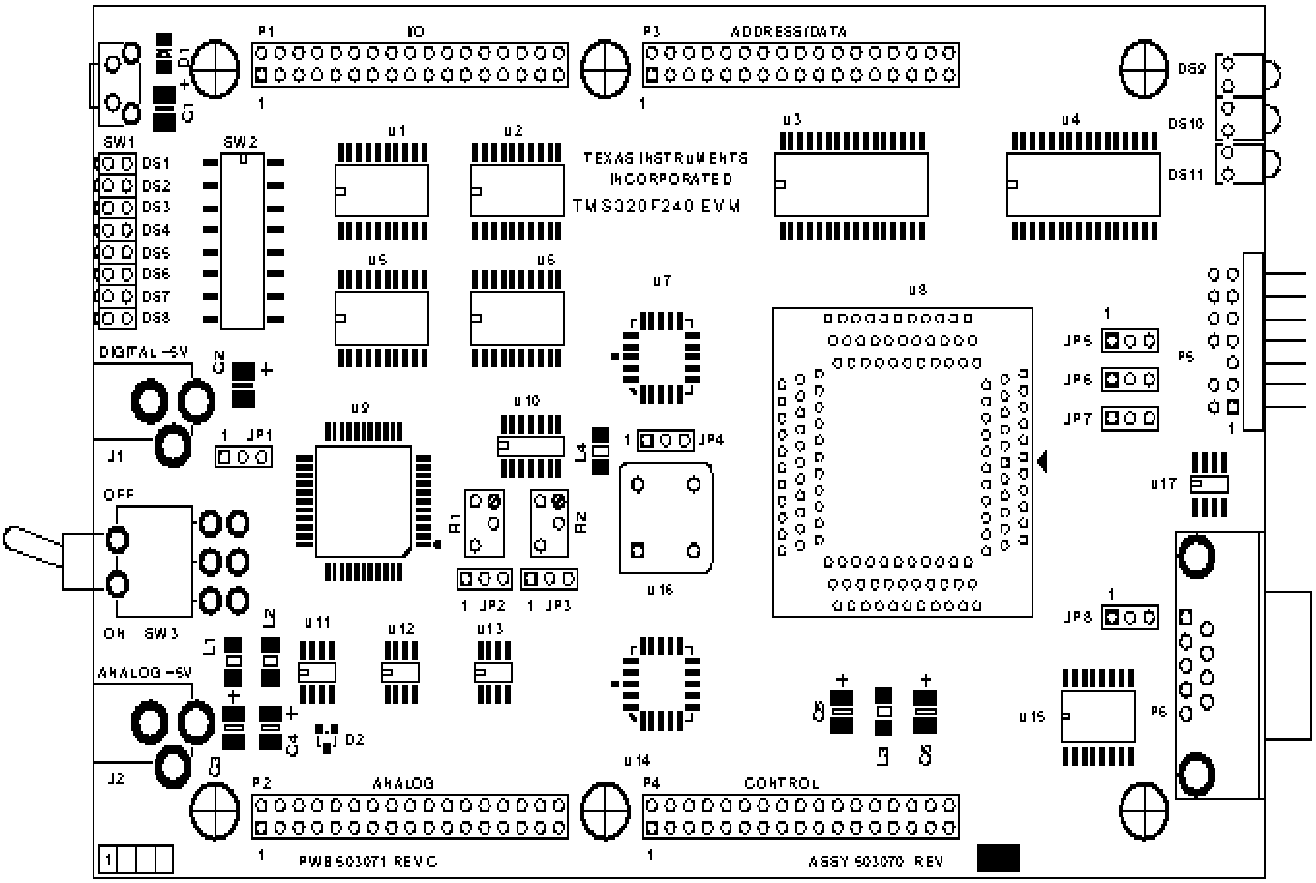
Stator phase line to line inductance:	4.8mH
line to line resistance	2.1Ω
Pole pairs	3
Nominal Torque Tn	2.2Nm
Nominal speed	3000rpm
Motor nominal power Pn	690W
Mechanical time constant	1.5ms
Electrical time constant	2.3ms
Torque constant	0.76Nm/A rms
Voltage constant	65Vpk/krpm

An embedded incremental encoder with a resolution of 1024 lines/revolution provides feedback for speed control.

DSP Development Board

Several TMS320F240 development platforms are available on the market either from TI or from one of its third parties. The TMS320F240 Evaluation Module (Figure 6) introduced by TI has been used in this application. The on-board DACs are used to output the values of several variables (currents, voltages, speed, and position) chosen from the Graphical User Interface presented at the end of this report. This feature is particularly useful in development stage.

Figure 6. Top View of the TMS320F240 Evaluation Module



The PLL unit is set for CPUCLK = 20MHz and SYSCLK = 10Mhz.

To disable the Watchdog unit, set Vccp pin voltage to 5V (JP5 position 2-3)

Power Electronics Board

The power hardware used to implement and test this PMSM drive is based on six power IGBT (IRGPC40F) driven by the DSP Controller via the integrated driver IR2130 This power inverter supports a rectified DC bus voltage of 310V and a maximum current of 10A. The DSP PWM (pulse width modulation) outputs are isolated from the power board by opto-couplers. The phases current sensing is performed via two current voltage transducers (LEM type) supplied with +/-15V. Their maximum input current is +/-10A, which is converted into a 2.5V output voltage.

Software Organization

The program *FOCPMSM.ASM* is based on two modules: the initialization module and the interrupt module.

Initialization Module Description

After a processor reset, the initialization module performs the following tasks:

- DSP setup : core, watchdog, clocks, ADC, SCI, general purpose IO, event manager
- Variables initializations : default values
- Interrupt source selection and enable
- Waiting loop

  The waiting loop implemented corresponds to an interruptible communication between the DSP and a Graphical User Interface. The DSP communicates via its asynchronous serial port to the COM port of a PC. The user can send commands via this RS232 link and update variables and flags from the computer.

Interrupt Module Description

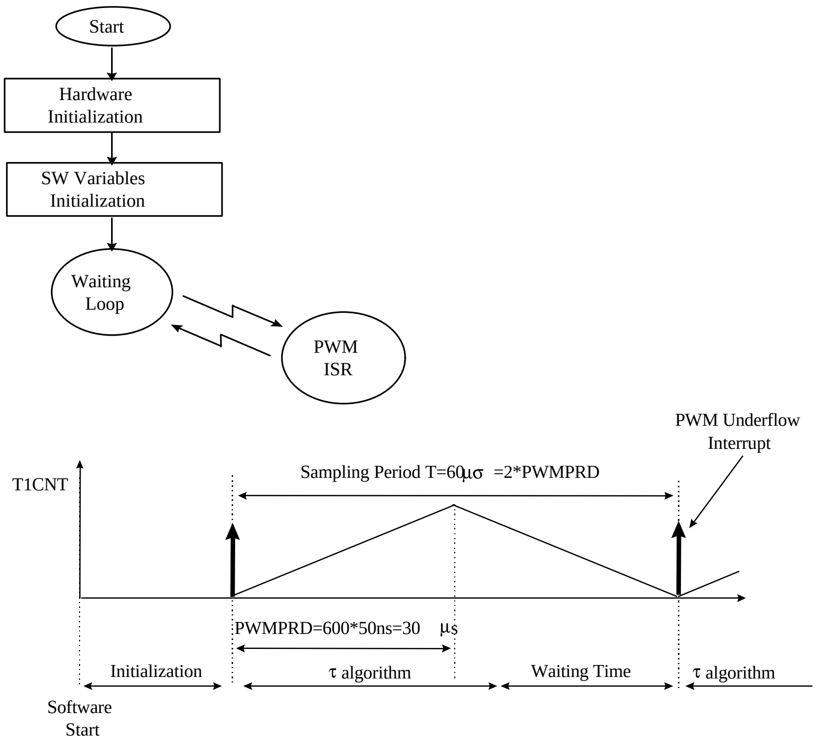
The interrupt module handles the whole FOC algorithm. It is periodically computed according to a fixed PWM (pulse width modulation) period value. The choice of the PWM frequency depends on the motor electrical constant L/R. If the PWM frequency is too low, audible noise can be heard from the motor. Usually, PWM frequencies are in the range of 20 kHz. In this report, a PWM frequency of 16kHz has been chosen.

In Figure 7, the sampling period T of 60 µs (16 kHz) is established by setting the timer period T1PER to 600 (PWMPRD=600). This timer is set in up-down count mode and generates a periodical interrupt on T1 underflow event.

The goal of the interrupt module is to update the stator voltage reference and to ensure the regulation of stator currents and rotor mechanical speed.

After the initialization module has completed, the rotation does not start immediately. As the program is interactive, the DSP waits for the user to select the Init/run menu option that set the internal flag “initphase”.

Figure 7. Software Flowchart and Timing



Depending on the status of this flag, either a magnetic stall or the complete speed FOC algorithm is performed.

If initphase = 0, the magnetic stall places the rotor in a known position at start. It is necessary for two reasons:

- The embedded encoder does not give an absolute information on the rotor position. Only a relative position can be computed from a known position.
- The rotor electrical position needs to be reset for the FOC.

  This stall is performed by applying a constant voltage vector to the stator phase: the constant phase currents flowing in the coils create a fixed stator flux. As a consequence, the rotor flux aligns itself naturally onto this stator flux (the rotor is stalled in this position).

  The component I_q of the stator current vector is set to the value I_{qrinit} (=$I_{nominal}$), I_d is set to 0. The arbitrary angular position of this vector is called θ_e.

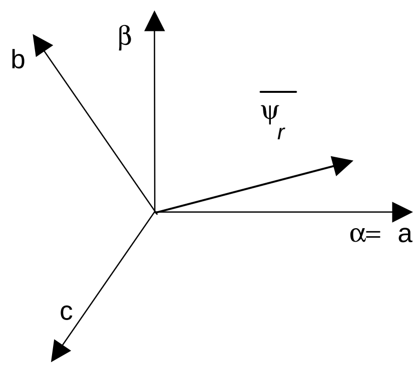
If initphase = 1, the electrical angle θ_e is shifted by 90°. As a consequence, the (d,q) axis are rotated from 90° apart. The d axis corresponds now to the real rotor flux position and the stator current vector I_s is moved to the new q axis. As a consequence, the rotor flux tends to align itself with the new stator flux vector position. As soon as the rotor starts to rotate, relative displacement information is sent to the DSP by the encoder. A new stator vector is computed every interrupt in order to maintain the 90 electrical degrees between the two fluxes.

These two steps are graphically explained below.

Start of the Motion

After reset, the rotor flux is in an unknown position (Figure 8).

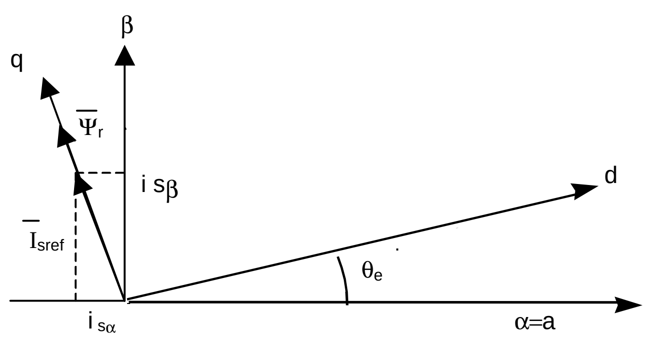
Figure 8. Rotor Flux Position at Standstill



Step 1: Initphase = 0

A fixed stator current vector I_{sref} is applied to the motor. The components of this vector are: I_{sdref}=0, I_{sqref}=Iqrinit (=$I_{nominal}$), θ_e

Figure 9. Stalled Rotor

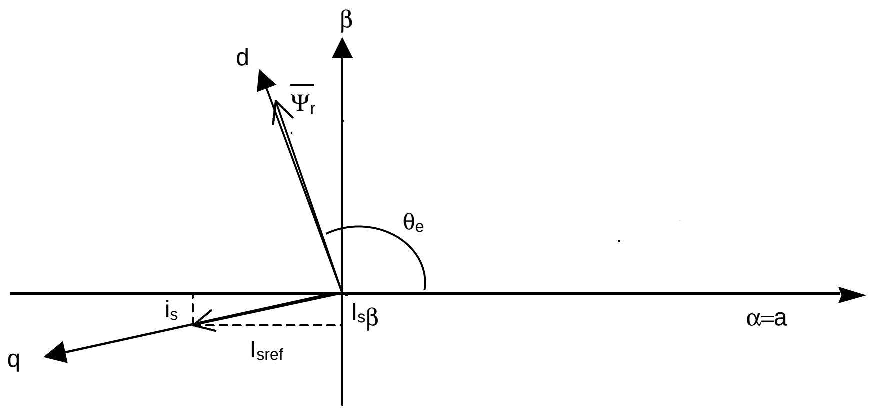


The rotor flux aligns itself to the axis q. For the time being, the (d,q) axis is not yet rotating. The rotor flux is in a known position but this position is not yet aligned with the d axis.

Step 2: Initphase = 1

90 electrical degrees is added to the value of θ_e, this action is equivalent to a frame rotation.

Figure 10. +90° Electrical Shift



Instantaneously, the stator current reference vector is moved 90° apart from its first position. (The rotor is physically at the same position as previously). The d axis now corresponds exactly to the position of the rotor flux.

As there is this 90 ° angular difference between the rotor flux and the stator flux, the interaction of the two fluxes produces torque and the rotor starts to rotate in order to align itself with I_{sref}.

The incremental encoder sends rotor position information to the DSP. This information is stored in a software counter called *"encoder"*.

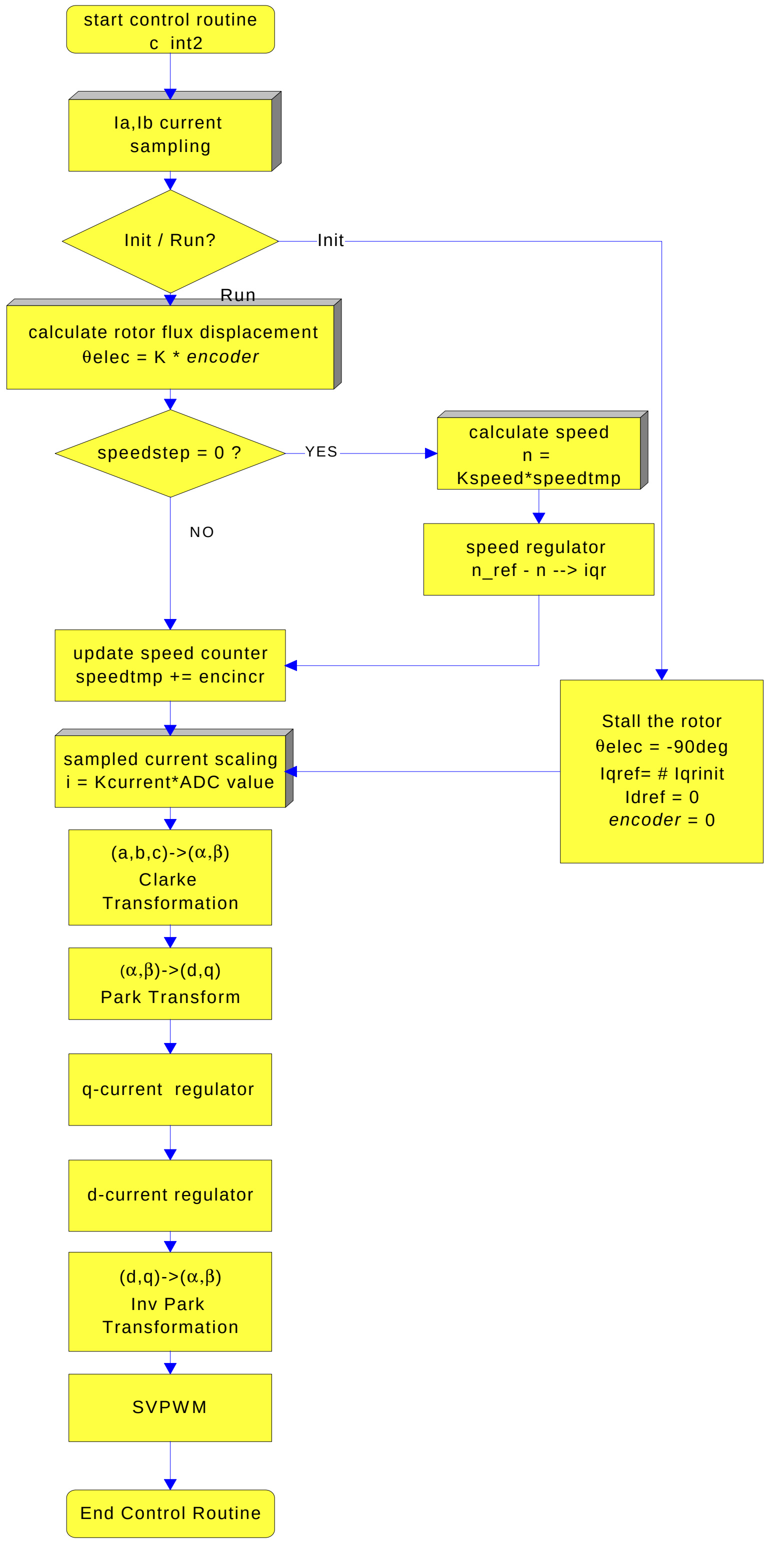
Every PWM interrupt, the stator voltage vector is updated to maintain the 90° between the two magnetic fluxes. This update is done according to the number of increments stored in the variable *encoder*.

For convenience , initial value θ_e of has been chosen in this report to be equal to -90°.

This makes the d axis correspond to the 0° electrical position at start. In fact, the electrical position is now computed with the formula θ_e = K**encoder*. As the number of increments in the variable *encoder* are null after reset, it was convenient to choose the value -90° as first value for θ_e.

The flowchart of the interrupt module is given in Figure 11.

Figure 11. Interrupt Module Flowchart



On the interrupt module flowchart, several software blocks appear. The shadowed blocks correspond to interface modules, whereas the nonshadowed blocks correspond to the core modules. The interface modules are low level routines that convert real wold data into their suitable numerical counterparts. The core modules use these formatted data to execute the several tasks of the FOC.

In order to be able to understand how the software modules have been implemented on the TMS320F240, an overview on the fixed-point arithmetic is needed. The Per-Unit model will also be discussed in the following section.

Fixed-Point Arithmetic

Representation of Numbers

In binary format, a number can be represented in signed magnitude, where the left-most bit represents the sign and the remaining bits represent the magnitude:

+6 (decimal) is represented as 10110_2 (binary) = $\mathbf{1}*(0*2^3+1*2^2+1*2^1+0*2^0)$

-6 (decimal) is represented as 10110_2 (binary) = $\mathbf{-1}*(0*2^3+1*2^2+1*2^1+0*2^0)$

Two's complement is an alternative form of representation used in most processors, including the TMS320. The representation of a positive number is the same in two's complement and in signed magnitude. However, the representation of a negative number is different.

+6 (decimal) is represented as 00110_2 (2s-comp) = $0*2^4+0*2^3+1*2^2+1*2^1+0*2^0$

-6 (decimal) is represented as 11010_2 (2s-comp) = $\mathbf{-1}*2^4+1*2^3+0*2^2+1*2^1+0*2^0$

The above words are represented on 5 bits only. The TMS320F240 is part of the TMS320C2xx 16bit fixed-point DSP family of TI. The native length of a word is 16bit on this family.

To represent real numbers on this fixed-point architecture, a Q_k format has to be chosen by the user. Q_k numbers can be represented by the following general formula:

$$Z = \mathbf{-\,b_{15-k}}*2^{15-k}+ b_{14-k}*2^{14-k} +\ldots b_0+b_{-1}*2^{-1}+ b_{-2}*2^{-2}+\ldots+ b_{-k}*2^{-k}$$

An implied dot separates the integer part from the fractional part of the Q_k number where *k* represents the quantity of fractional bit.

For instance the real number π (3.14159) can be represented in Q_{13} with finite precision as follow :

$$011.0\ 0100\ 1000\ 0111_2 = 0{*}2^2 + 1{*}2^1 + 1{*}2^0 + 0{*}2^{-1} + 0{*}2^{-2} + 1{*}2^{-3} + 0{*}2^{-4} + 0{*}2^{-5} +$$
$$1{*}2^{-6} + 0{*}2^{-7} + 0{*}2^{-8} + 0{*}2^{-9} + 0{*}2^{-10} + 1{*}2^{-11} + 1{*}2^{-12} + 1{*}2^{-13}$$

The number of bits dedicated to the fractional part affects the accuracy of the result while the integer part affects the dynamic range of values that can be represented. The Q_{15} format offers the best precision but only real numbers comprised between –1 and +1 can be represented.

The Q_k format offers a compromise between dynamic range and precision. The Q_{12} numeric format is used in the major part of this report : 4 bits are dedicated to the integer part and 12 bits are dedicated to the fractional part. The precision of this format is 2^{-12} (0.00024414). The represented numbers are in the range of [-8;8] to ensure that values can handle each drive control quantity, not only during steady state operation but also during transient operation.

Arithmetic operations

Multiplication

The following example shows how two real numbers (X and Y) coded in Q_{12} are multiplied

X = -1.125_{10} is represented as $1110.\ 1110\ 0000\ 0000_2$ in Q_{12}

Y = $+1.375_{10}$ is represented as $0001.\ 0110\ 0000\ 0000_2$ in Q_{12}

```
   0001 011(0 0000 0000) (+1.375)
*  1110 111(0 0000 0000) (-1.125)
   ------------------------------
   0001 011
  0 0010 11 .      (2-scompl)      SUM1= 00100001+(zeroes)
 00 0101 1 . .                     SUM2=001001101+(zeroes)
000 0000 . . .                     SUM3=0001001101+(zeroes)
0001 011 . . . .                   SUM4=00011111101+(zeroes)
1 1101 01 . . . . .
------------------------------------
z = 1 1111 0.011 101(00…00)    (-1.546875)
                    18 zeroes
```

The multiplication of a Q_k (2^k) number by a Q_p (2^p) number results in a Q_{k+p} (2^{k+p}) number (the same rule also exists in base 10. ex : $10^3 * 10^5 = 10^8$). In the case of a Q_{12} by Q_{12} multiplication, the virtual dot is shifted and the 24 least significant bits of the 32-bit accumulator represents the fractional part of the result ($Q_{12} * Q_{12} = Q_{24}$).

As the result of the multiplication gives a 30bit number, the SXM bit (sign extension mode) is set to propagate the sign to the two most significant bits of the accumulator.

Z will be stored back in Q_{12} format. To do so, the content of the accumulator is left shifted four times and the upper word of the accumulator is stored in Z.

Z is stored as $1110.\ 0011\ 1010\ 0000_2$ in Q_{12} = -1.546875 (decimal)

Addition

The following example shows how two real numbers (X and Y) coded in Q_{12} are added.

X = $+1.125_{10}$ is represented as $0001.\ 0010\ 0000\ 0000_2$ in Q_{12}

Y = $+1.375_{10}$ is represented as $0001.\ 0110\ 0000\ 0000_2$ in Q_{12}

Z is stored as $0010.\ 1000\ 0000\ 0000_2$ in Q_{12} = 2.5 (decimal)

PU Model and Base Values

The Per Unit model (PU) is associated with reduced value notion. As the TMS320F240 is a fixed-point DSP, it has been shown that the greatest precision is obtained in Q15 format but the dynamic range of this format is small: it is comprised between –1 and +1 only.

Using a fixed-point DSP, it is necessary to reduce the amplitude of the variables in order to get a fractional part with a maximum precision. The notion of Per Unit model is introduced to use this fixed-point feature. It is usually associated with the nominal values of the motor.

The per-unit current is usually defined as $i_{pu} = I / I_{nominal}$

The above equation shows that $i_{pu} = 1$ when the current reaches its nominal value. Instead of using the nominal value as reference, a base value is preferred.

For currents and voltages, the reason to choose a base different from the nominal values is that nominal values usually given by the motor manufacturer are RMS (root mean square).

Then, the preferred Per Unit model for the current is given by:

$i = I / I_{base}$ where $I_{base} = I_{nominal} * \sqrt{2}$

and the PU model for the voltage is given by:

$v = V / V_{base}$ where $V_{base} = V_{nominal} * \sqrt{2}$

For this application, the other PU model defined is:

$$n = \frac{mechanical\ rotor\ speed}{w_{base}}$$

In this application report, the base value of the mechanical speed corresponds to its nominal value.

$$I_{base} = \sqrt{2}I_n = \sqrt{2} \cdot 2.9 = 4.1A$$

$$V_{base} = \sqrt{2}V_n = \sqrt{2} \cdot 127 \cong 180V$$

$$w_{base} = 2pf_n = 2p \cdot 50 = 314.15\frac{rad}{\sec}$$

$$\Psi_{base} = \frac{V_{base}}{w_{base}} = \frac{180}{314.15} = 0.571Wb$$

As mentioned earlier, transient currents (for instance) might reach higher values than their nominal values. Furthermore, the motor speed range might be extended above the nominal speed (field weakening), then every per unit value might be greater than one. This remark forces the implementation to handle these situations and thus the suited numerical format chosen was Q12 for the PU models.

The Q12 representation of 1 is 1000h. The PU value is equal to 1 when the value is equal to its base.

Core Modules

The core modules use formatted data to execute the different tasks of the FOC. The core modules described are:

- Co-ordinate transformations : Clarke, Park, Park^{-1}
- Generation of sinθ, cosθ with a lookup table.
- Variable stator voltage vector generation : Space Vector Modulation algorithm
- Speed regulation, current regulation

Co-ordinate Transformations

As first approach for the application of the fixed-point representation concept, the implementation of the Clarke geometrical transformation is explained below. The other modules (Park, Park^{-1}) are also implemented in the program FOCPMSM.ASM.

These transformations are also explained in *Clarke & Park Transforms on the TMS320C2xx* (BPRA048).

As mentioned previously, the stator phase current vector is projected from a 3-phase (a,b,c) system in a (α,β) non-rotating frame by the Clarke transformation. The mathematical equations are:

$$i_a = i_a$$

$$i_b = \frac{1}{\sqrt{3}} \cdot i_a + \frac{2}{\sqrt{3}} i_b$$

$$i_a + i_b + i_c = 0$$

The following assembly function handles this mathematical transformation:

```
*********************************************
* (a,b,c) -> (alfa,beta) axis transformation
* iSalfa = ia
* iSbeta = (2 * ib + ia) / sqrt(3)
*********************************************
* Input variables : ia,ib       Q12 format
* Output variables : iSalfa, iSbeta   Q12 format
* Local variables modified : tmp      Q12 format

    lacc    ia
    sacl    iSalfa
    lacc    ib,1            ;iSbeta = (2 * ib + ia) / sqrt(3)
    add     ia
    sacl    tmp
    lt      tmp
    mpy     SQRT3inv        ;SQRT3inv = (1 / sqrt(3)) = 093dh
                            ;4.12 format = 0.577350269
    pac
    sach    iSbeta,4
```

This routine gives a practical example of multiplication of Q12 numbers.To easily find the correspondence between the fractional format of $SQRT^{-1}(3)$ and its Q12 equivalent, a simple multiplication by 2^{12} (= 4096) has to be done:

0. 577350269 * 4096 ≈ 2365 → 093Dh

The Clarke (a,b,c)->(α,β) projection requires 9 words of ROM, 5 words of RAM. A complete table of each function requirements is given in the conclusion.

The Park and $Park^{-1}$ are also implemented in FOCPMSM.ASM.

Generation of Sine and Cosine

The Park and $Park^{-1}$ use the value of the rotor electrical position in order to handle the stator current vector projection in a rotating frame. The electrical position is not directly used in this transforms but the sine and cosine values of this electrical position.

To obtain both sine and cosine from the electrical angle, a sine look-up table has been implemented.

The table contains 256 words to represent sine values of electrical angles in the range [0;360°]. As a result, the resolution on θ_e is limited to 360/256 =1.40625°.

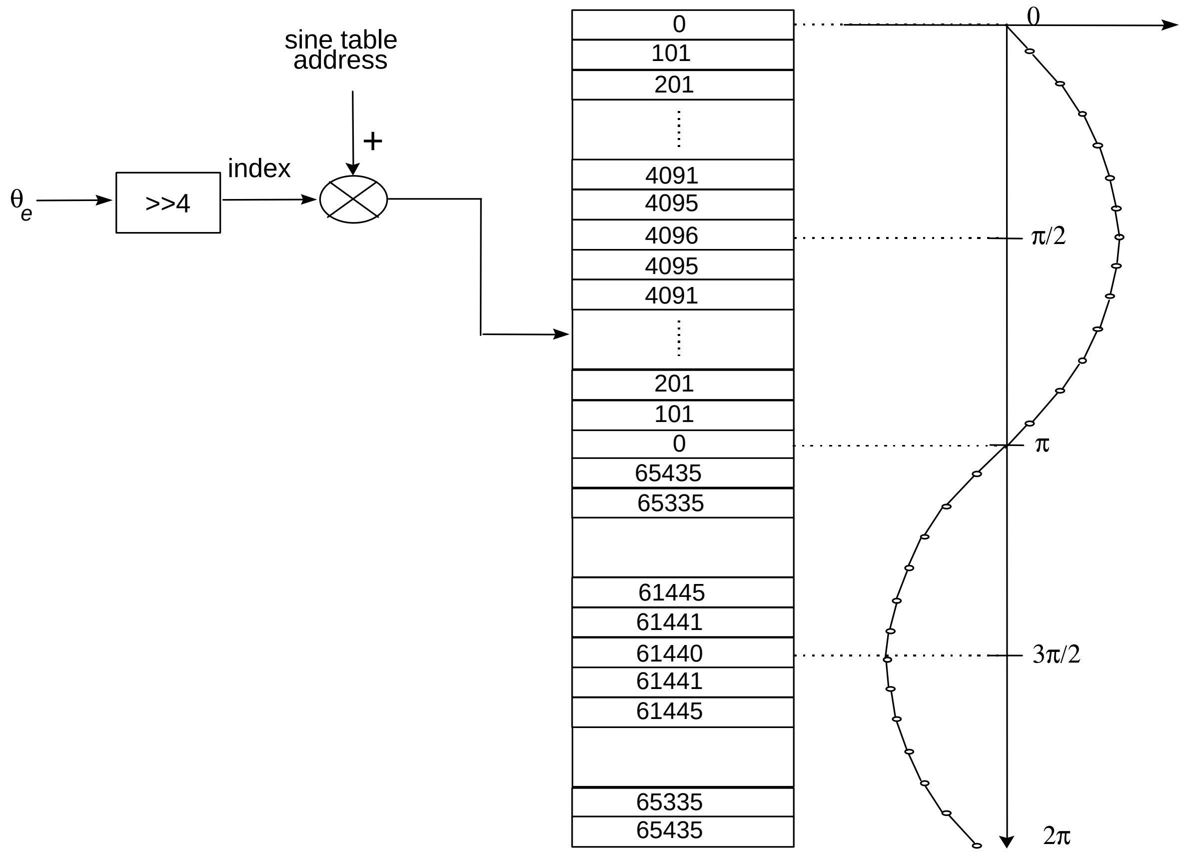
θ_e = electrical angle / 360° (with θ_e in the range [0;FFFh])

θ_e varies from 0 to 4095 (see position sensing module). As only 256 words are available to represent this range, θ_e is divided by 16 and stored into the variable *index* that will be used to address the lookup table.

The content of the table raw pointed by the index is fetched in indirect addressing mode via AR5 auxiliary register. This content coded in Q12 is stored in the variable *sin* that will be used in the Park transforms.

Note that to get the cosine value of the electrical angle, 90° are added to θ_e This operation corresponds to add 64 (256/4) to the value of *index*. The result is stored in the variable *cos*.

Figure 12. Sinq_e, Cosq_e Calculation using the Sine Look-up Table



Space Vector Modulation

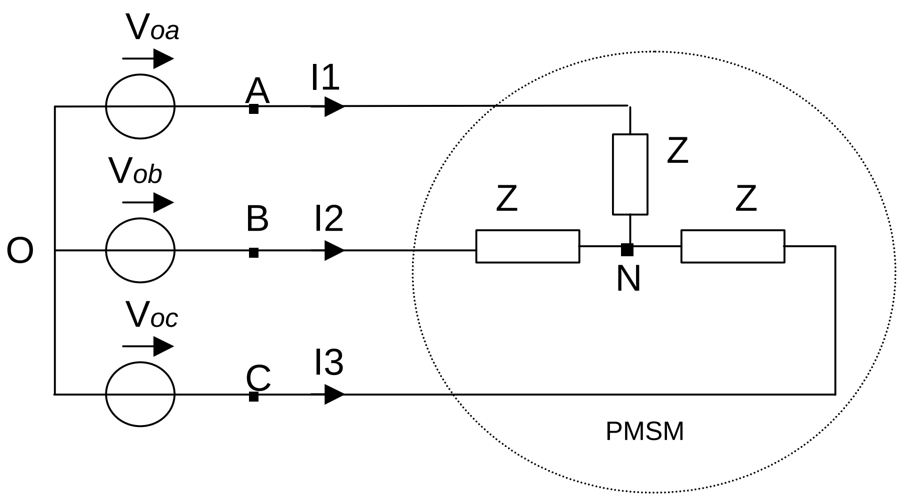
The Space Vector Modulation is used to generate the voltages applied to the stator phases. It uses a special scheme to switch the power transistors to generate pseudo sinusoidal currents in the stator phases.

This switching scheme comes from the translation of the (α,β) voltage reference vector into an amount of time of commutation (on/off) for each power transistors. In order to understand some of the assumptions made in the case of the rectified voltage, a brief description of three phase systems is described in the following section. .

Expression of the 3 Phase Voltages (Phase to Neutral)

Previously, the method used to generate a rotating magnetic field was to use three independent voltage sources that were dephased from 120 degrees from one another.

Figure 13. 3-Phase Equilibrate System



In this standard tri-phased system, 3 sinusoidal voltages are applied to each of the motor phases to generate the sinusoidal currents. These voltages can be expressed as follows:

$$V_{oa} = V\sqrt{2}\cos(w_e * t)$$
$$V_{ob} = V\sqrt{2}\cos(w_e * t - \frac{2p}{3})$$
$$V_{oc} = V\sqrt{2}\cos(w_e * t - \frac{4p}{3})$$

In order to calculate the phase to neutral voltages (respectively V_{an}, V_{bn}, V_{cn}) from the applied source voltages (respectively V_{oa}, V_{ob}, V_{oc}), the assumption is made that the system is equilibrated is made. This leads to the following equations:

$V_{on} = V_{oa} + Z*I_1$

$V_{on} = V_{ob} + Z*I_2$

$V_{on} = V_{oc} + Z*I_3$

then

$3*V_{on} = V_{oa}+V_{ob}+V_{oc}+Z(I_1 + I_2 + I_3)$ where $(I_1 + I_2 + I_3) = 0$

As Von is now expressed by a combination of the source voltages, the phase to neutral voltage for phase A can be calculated as:

$V_{an} = V_{on} - V_{oa} = (1/3)(V_{oa} + V_{ob} + V_{oc}) - V_{oa} = -2/3V_{oa} + 1/3V_{ob} + 1/3V_{oc}$

The same calculation is made for the three phases leading to :

$V_{an} = (1/3)(2*V_{ao}-V_{bo}-V_{co})$

$V_{bn} = (1/3)(2*V_{bo}-V_{ao}-V_{co})$

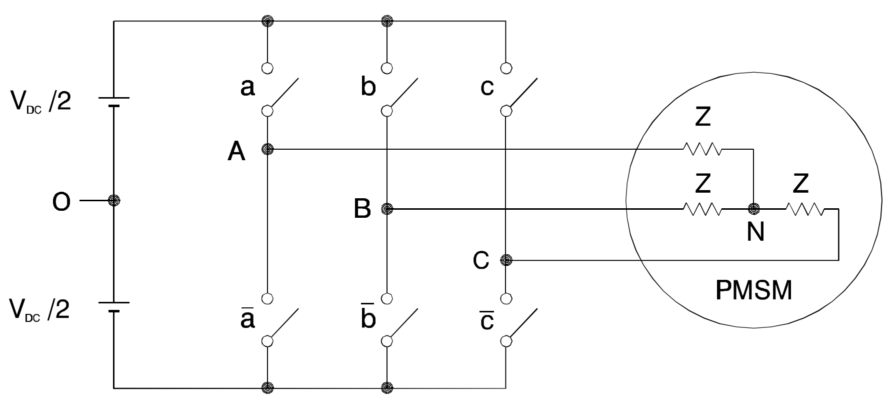
$V_{cn} = (1/3)(2*V_{co}-V_{ao}-V_{bo})$

Application to the Static Power Bridge

In the case of a static power bridge, sinusoidal voltage sources are not used. They are replaced by 6 power transistors that act as on/off switches to the rectified DC bus voltage. The goal is to recreate a sinusoidal current in the coils to generate the rotating field. Owing to the inductive nature of the phases, a pseudo sinusoidal current is created by modulating the duty cycle of the power switches.

In Figure 14, the power transistors are activated by the signals (a,b,c) and their complemented values.

Figure 14. Power Bridge



Only eight combinations of the switches are possible with this configuration (Table 1). The applied voltages are referenced to the virtual middle point of rectified voltage.

Table 1. Power Bridge Output Voltages (V_{AO}, V_{BO}, V_{CO})

A	B	C	V_{AO}	V_{BO}	V_{CO}
0	0	0	-VDC/2	- VDC/2	- VDC/2
0	0	1	- VDC/2	- VDC/2	+ VDC/2
0	1	0	-VDC/2	+VDC/2	- VDC/2
0	1	1	-VDC/2	+VDC/2	+ VDC/2
1	0	0	+VDC/2	-VDC/2	- VDC/2
1	0	1	+VDC/2	-VDC/2	+ VDC/2
1	1	0	+VDC/2	+VDC/2	- VDC/2
1	1	1	+VDC/2	+VDC/2	+ VDC/2

Because of the equations :

$V_{an} = (1/3)(2*V_{ao}-V_{bo}-V_{co})$

$V_{bn} = (1/3)(2*V_{bo}-V_{ao}-V_{co})$

$V_{cn} = (1/3)(2*V_{co}-V_{ao}-V_{bo})$

It is possible to express each phase to neutral voltages, for every combination of the power transistors as listed in Table 2.

Table 2. Power Bridge Output Voltages (V_{AN}, V_{BN}, V_{CN})

A	B	C	V_{AN}	V_{BN}	V_{CN}
0	0	0	0	0	0
0	0	1	- VDC/3	- VDC/3	2VDC/3
0	1	0	- VDC/3	2VDC/3	- VDC/3
0	1	1	-2VDC/3	VDC/3	VDC/3
1	0	0	2VDC/3	- VDC/3	- VDC/3
1	0	1	VDC/3	-2VDC/3	VDC/3
1	1	0	VDC/3	VDC/3	-2VDC/3
1	1	1	0	0	0

Expression of the Stator Voltages in the (α,β) Frame

In the FOC algorithm, the control variables are expressed in a rotating frame. It has been mentioned that the current vector I_{sref} that directly controls the torque is transformed in a voltage reference vector by the Park^{-1} transform. This voltage reference is expressed in the (α,β) frame. To make the relationship between the 3 phase voltages (V_{AN}, V_{BN} and V_{CN}) and the voltage reference vector, the 3 phase voltages are also projected in the (α, β) frame.

The expression of the 3 phase voltages in the (α,β) frame are given by the general Clarke transform equation:

$$\begin{bmatrix} V_{sa} \\ V_{sb} \end{bmatrix} = \frac{2}{3} \begin{bmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \end{bmatrix} \begin{bmatrix} V_{AN} \\ V_{BN} \\ V_{CN} \end{bmatrix}$$

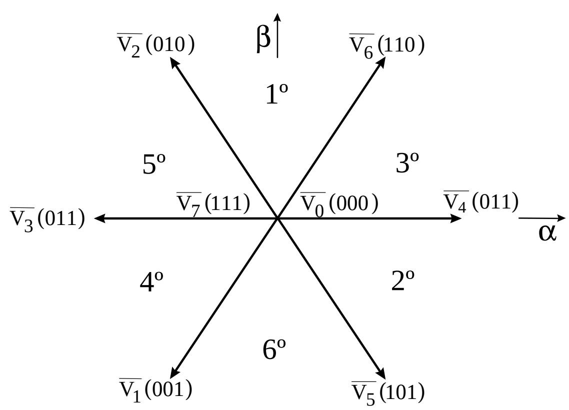
Since only 8 combinations are possible for the power switches, V_{sa} and V_{sb} can also take only a finite number of values in the (α,β) frame according to the status of the transistor command signals (a,b,c).

Table 3. Stator Voltages

A	B	C	Vα	Vβ	
0	0	0	0	0	$\vec{V}_0$
0	0	1	$-\frac{V_{DC}}{3}$	$-\frac{V_{DC}}{\sqrt{3}}$	$\vec{V}_1$
0	1	0	$-\frac{V_{DC}}{3}$	$\frac{V_{DC}}{\sqrt{3}}$	$\vec{V}_2$
0	1	1	$-\frac{2}{3}V_{DC}$	0	$\vec{V}_3$
1	0	0	$\frac{2}{3}V_{DC}$	0	$\vec{V}_4$
1	0	1	$\frac{V_{DC}}{3}$	$-\frac{V_{DC}}{\sqrt{3}}$	$\vec{V}_5$
1	1	0	$\frac{V_{DC}}{3}$	$\frac{V_{DC}}{\sqrt{3}}$	$\vec{V}_6$
1	1	1	0	0	$\vec{V}_7$

The eight voltage vectors defined by the combination of the switches are represented in Figure 15.

Figure 15. Voltage Vectors

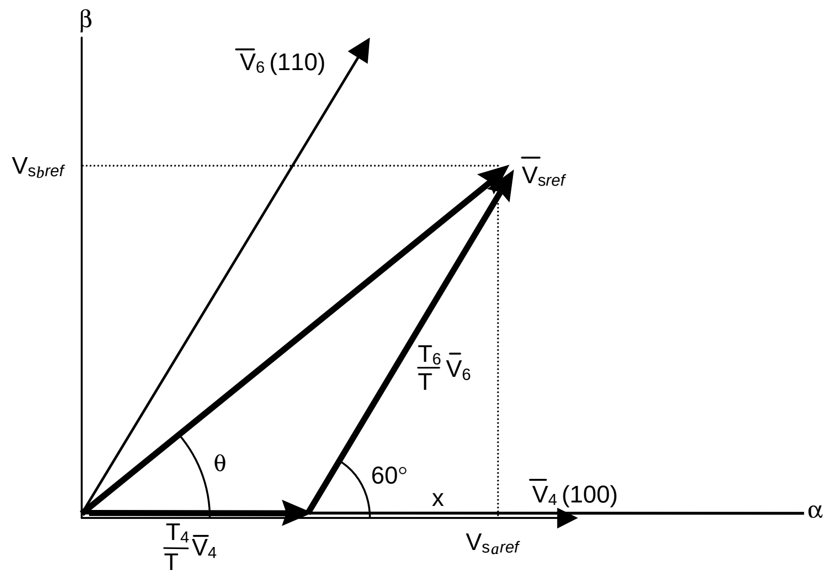


Now, given a reference voltage (coming from the Park^{-1} transform), the following step is to use the 8 above defined vectors to approximate this reference voltage.

Projection of the Stator Reference Voltage Vs

The method used to approximate the desired stator reference voltage with only eight possible states of switches is to combine adjacent vectors of the reference voltage and to modulate the time of application of each adjacent vector.

Figure 16. Projection of the Reference Voltage Vector



In Figure 16, the reference voltage V_{sref} is in the third sector and the application time of each adjacent vector is given by:

$$\begin{cases} T = T_4 + T_6 + T_0 \\ \vec{V}_{sref} = \dfrac{T_4}{T}\vec{V}_4 + \dfrac{T_6}{T}\vec{V}_6 \end{cases}$$

The determination of the amount of times T_4 and T_6 is given by simple projections:

$$\begin{cases} V_{sbref} = \dfrac{T_6}{T}\left\|\vec{V}_6\right\|\cos(30°) \\ V_{saref} = \dfrac{T_4}{T}\left\|\vec{V}_4\right\| + x \\ x = \dfrac{V_{sbref}}{tg(60°)} \end{cases}$$

Finally, with the (α,β) components values of the vectors given in the previous table, the amount of times of application of each adjacent vector is:

$$T_4 = \frac{T}{2V_{DC}}(3V_{saref} - \sqrt{3}V_{sbref})$$

$$T_6 = \sqrt{3}\frac{T}{V_{DC}}V_{sbref}$$

The rest of the period is spent in applying the null vector. The variable T/V_{DC} is named V_{DCinvT}. T is the period of the PWM interrupt and V_{DC} is the rectified DC voltage.

To keep proportions in the software implementation, the variables V_{DC} and VDCinvT are expressed in P.U and in Q12 as follow:

$$v_{DC} = \frac{V_{DC}}{V_{base}} = \frac{310}{180} = 1.722 \Leftrightarrow 1B8Eh \quad 4.12\,\text{f}$$

where V_{DC} is the DC bus voltage and v_{DC} its correspondent PU value.

$$v_{DCinvT} = \frac{T}{2v_{DC}} \Leftrightarrow \frac{PWMPRD}{v_{DC}} = \frac{600}{1.722} = 348 \Leftrightarrow 15Ch$$

For every sector, a commutation duration is calculated. The amount of times of vector application can all be related to the following variables:

$$X = \sqrt{3}v_{DCinvT}v_{Sbref}$$

$$Y = \frac{\sqrt{3}}{2}v_{DCinvT}v_{Sbref} + \frac{3}{2}v_{DCinvT}v_{Saref}$$

$$Z = \frac{\sqrt{3}}{2}v_{DCinvT}v_{Sbref} - \frac{3}{2}v_{DCinvT}v_{Saref}$$

In the previous example for sector 3, T_4 = -Z and T_6 = X.

In order to know which of the above variable apply, the knowledge of the sector in which the reference voltage vector is, is needed.

To determine this sector, a simple approach is to calculate the projections V_a, V_b and V_c of the reference voltage vector in the (a,b,c) plane. These projections are then compared to 0.

The projections V_a, V_b and V_c are given by the Clarke^{-1} transform as follow:

$$v_a = v_{Sbref}$$

$$v_b = \frac{1}{2}(\sqrt{3}v_{Saref} - v_{Sbref})$$

$$v_c = \frac{1}{2}(-\sqrt{3}v_{Saref} - v_{Sbref})$$

The complete algorithm performed by the Space Vector Module is given in the next section.

Space Vector Algorithm

Now that the meaning of the variables has been given, the order in which the steps are processed during the PWM interrupt is given.

The first step is to determine in which sector the voltage vector defined by $v_{Sa_{ref}}$, $v_{Sb_{ref}}$ is found. The following few code lines give the sector as output:

$$
\begin{array}{l}
\text{sector determination} \\
IF\ v_a > 0\ THEN\ A := 1, \qquad ELSE\ A := 0 \\
IF\ v_b > 0\ THEN\ B := 1, \qquad ELSE\ B := 0 \\
IF\ v_c > 0\ THEN\ C := 1, \qquad ELSE\ C := 0 \\
\text{sector} := A + 2B + 4C
\end{array}
$$

The second step is to calculate and saturate the duration of the two sector boundary vectors application as shown below:

$$
\begin{array}{l}
CASE\ \text{sector}\ OF \\
1 \quad t_1 = Z \qquad t_2 = Y \\
2 \quad t_1 = Y \qquad t_2 = -X \\
3 \quad t_1 = -Z \quad t_2 = X \\
4 \quad t_1 = -X \quad t_2 = Z \\
5 \quad t_1 = X \qquad t_2 = -Y \\
6 \quad t_1 = -Y \quad t_2 = -Z \\
\text{end times calculation} \\
\text{Saturations} \\
IF\ (t_1 + t_2) > PWMPRD\ THEN \\
\qquad t_{1SAT} = t_1 \dfrac{PWMPRD}{t_1 + t_2} \\
\qquad t_{2SAT} = t_2 \dfrac{PWMPRD}{t_1 + t_2}
\end{array}
$$

The third step is to compute the three necessary duty cycles. This is shown below:

$$
\begin{cases}
t_{aon} = \dfrac{PWMPRD - t_1 - t_2}{2} \\
t_{bon} = t_{aon} + t_1 \\
t_{con} = t_{bon} + t_2
\end{cases}
$$

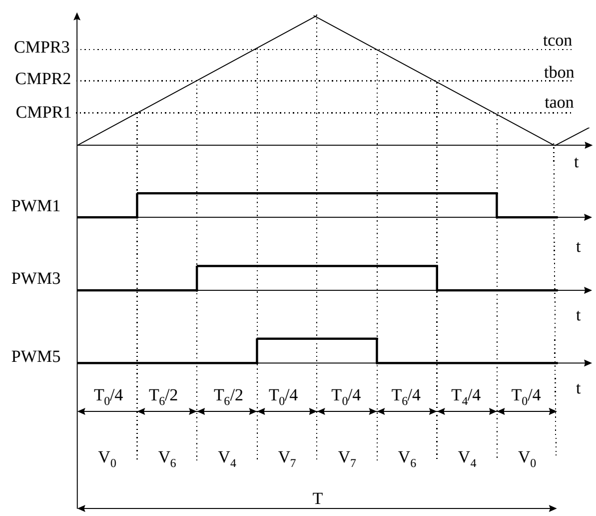
The last step is to assign the right duty cycle (txon) to the right motor phase (in other words, to the right CMPRx) according to the sector. The table below depicts this determination.

Figure 17. Table Assigning the Right Duty Cycle to the Right Motor Phase

Phase \ Sector	1	2	3	4	5	6
CMPR1	tbon	tbaon	taon	tcon	tcon	tbon
CMPR2	taon	tcon	tbon	tbon	taon	tcon
CMPR3	tcon	tbon	tcon	taon	tbon	taon

Figure 18 shows an example of one vector that would be in sector 3.

Figure 18. Sector 3 PWM Patterns and Duty Cycles



Event Manager Configuration

This section describes how to program the TMS320F240 peripherals in order to handle the space vector module.

TIMER1 is the time base of the PWM interrupts generation. It is configured in up-down counting mode to generate the symmetrical PWM patterns. Its frequency is set at 16kHz.

```
PWMPRD          .set    258h    ;PWM Period T=2*600*50ns=60us


    splk    #PWMPRD,T1PER   ;Set PWM interrupt period
    splk    #0,T1CNT
    splk    #0A800h,T1CON   ;Ignore Emulation suspend
                            ;Up/Down count mode
                            ;x/1 prescalar
                            ;Use own TENABLE
                            ;Disable Timer
                            ;Internal Clock Source
                            ;Reload Compare Register when T1CNT=0
                            ;Disable Timer Compare operation
```

The Timer 1 control register T1CON is programmed in order to get a 50ns resolution : the prescalar clock of the timer is set to 1 giving the highest possible resolution. The individual T1 General Purpose Compare register is reloaded every PWM cycle but not used in this application. For this reason, the General Purpose Control Register (GPTCON) is left to its default value. In fact, the only Compare registers used are the 3 Full Compare registers associated to TIMER1.

These 3 Full Compare registers are controlled by the Compare Control register (COMCON). This register is programmed as follow:

```
splk  #0207h,COMCON     ;Reload Full Compare when T1CNT=0
                        ;Disable Space Vector
                        ;Reload Full Compare Action when T1CNT=0
                        ;Enable Full Compare Outputs
                        ;Disable Simple Compare Outputs
                        ;Select GP timer1 as time base
                        ;Full Compare Units in PWM Mode
splk  #8207h,COMCON     ;enable compare operation
```

The Full Compare registers are updated at the end of the PWM interrupt routine with the calculated values t_{aon}, t_{bon}, t_{con} .

The output of the Compare operation are not directly sent to the Output Logic but are previously passed through the PWM Deadband on-chip circuit. Depending on the power bridge pre-driver used, the control register DBTCON has to be programmed.

In this application, an IR2130 from International Rectifier has been used and no deadband time has been programmed because this chosen pre-driver has already an internal deadband time.

```
splk    #0000h,DBTCON   ;no dead band
```

Once the deadband unit has been passed, the signals are sent to the Output Logic (see TMS320C24x Vol2 User's Guide page 2-46) that activates the DSP PWM pins. The polarity of the PWM pins is chosen in the Full Compare Action Control Register (ACTR) as follow:

```
ldp     #DP_EV
splk    #0666h,ACTR     ;Bits 15-12 not used, no space vector
                        ;PWM compare actions
                        ;PWM5/PWM6 - Active Low/Active High
                        ;PWM3/PWM4 - Active Low/Active High
                        ;PWM1/PWM2 - Active Low/Active High
```

The PWM pins are paired to control the high side and the low side of the pre-driver.

PI Regulators

The PI (Proportional-Integral) regulators are implemented with output saturation and with integral component correction. The constants K_i, K_{pi}, K_{cor} (proportional, integral and integral correction components) have been experimentally determined using the Graphic User's interface (option 5 and 6). Their theoretical determination (root locus and pole placement) is beyond the scope of this report. For this application, the current regulator parameters are:

K_i . 0.03⇔07Ah

K_{pi} 0.60⇔999h

K_{cor}. 0.05⇔0cch

The speed regulators parameters are:

K_{ispeed} 0.03⇔7ah

$K_{pispeed}$ 6.5⇔06800h

$K_{corspeed}$ 0.0046⇔12h

All constants are in Q12 format and the integral correction component is calculated by using the formula $K_{cor} = K_i/K_{pi}$.

Interface Modules

The interface modules are low level routines that convert real wold data into their suitable numerical counterparts.

The interface modules described are:

- Current sensing and scaling
- Mechanical position sensing and scaling
- Electrical position and mechanical speed scaling

The Sensing modules handle directly the hardware interface and dialog via the integrated TMS320F240 peripherals.

The Scaling modules transform the information into a fixed-point representation related to a Per Unit model

Current Sensing Module

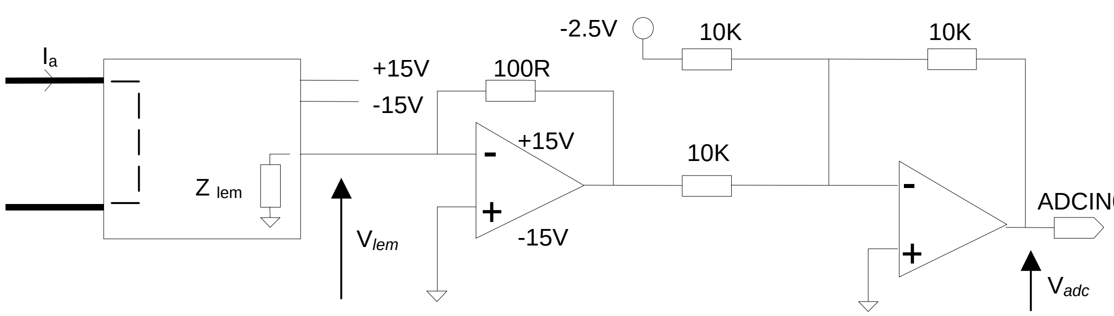
This module handles the conversion of the 3 stator phase currents into their basic binary representation.

Two LEMs (current-voltage transducer) sense the phase currents. They convert the current information into voltage information. These voltages are sampled and converted by the TMS320F240 Analog to Digital Converter and stored in the variable i_a and i_b.

Hardware Solution

Figure 19 represents the hardware interface put in place to realize the described function.

Figure 19. Current Sensing Hardware



The selected ADC inputs pins are ADCIN0 for phase a and ADCIN8 for phase b. As those pins are shared (multiplexed) with general purpose IO pins, the Output Control Register A (OCRA) will be set up to select the ADC input functionality.

The LEM converts the current information from phase a and b into a voltage information (V_{lem})

Ia and Ib are in the range +/-10A

A second translation is performed in two steps in order to adapt V_{lem} to the TMS320F240 ADC input voltage specification:

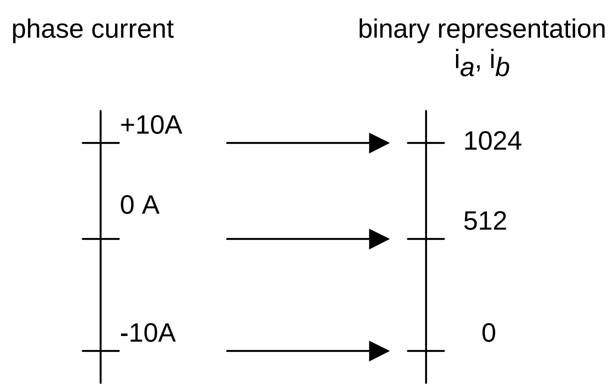
First, a gain is applied to V_{lem} in order to get an intermediate voltage in the range [-2.5v , +2.5V]. Then, a voltage shift of 2.5V is applied to meet the [0,5V] input range of the ADC.

The voltages V_{adcin0} and V_{adcin8} are sample and converted by the dual 10 bit ADC. The result of the conversion is stored in binary format in the variables i_a and i_b.

Sensing Scale Translation

Figure 20 represents the correspondence between the stator phase currents and their binary representations:

Figure 20. Current Sensing Scale Translation



Associated Low-Level Software

Initialization Module

As the TMS320F240 A-to-D converter is made of commuted capacitors, the ADC clock has to be defined according to the global clock (CPUCLK) and divided by a factor of prescaler.

All the internal clocks are derivated form CPUCLK and the oscillator used on the Evaluation Module is a 10MHz one. CPUCLK of 20 MHz (50ns) is created by this oscillator associated to the internal PLL as follows.

```
*****************************************
* Initialization of the TMS320F240 Clocks
*****************************************
    splk    #00000010b,CKCR0;PLL disabled
                            ;LowPowerMode0
                            ;ACLK enabled
                            ;SYSCLK 5MHz
    splk    #10110001b,CKCR1;10MHz CLKIN
                            ;Do not divide PLL
                            ;PLL ratio x2 (CPUCLK=20MHz)
    splk    #10000011b,CKCR0;PLL enabled
                            ;LPM0
                            ;ACLK enabled
                            ;SYSCLK 10MHz
```

The system clock (SYSCLK) had been set to 10Mhz. Setting the prescaler to 10 gives an ADC clock of 1 MHz. The 2-level deep FIFOs are emptied.

```
****************************************************************
* A/D initialization
****************************************************************
    ldp        #DP_PF1
    splk    #0003h,ADC_CNTL2;prescaler set for a 10MHz oscillator
    lacc    ADC_FIFO1       ;empty FIFO
    lacc    ADC_FIFO1
    lacc    ADC_FIFO2
    lacc    ADC_FIFO2
```

To select the pins ADCIN0 and ADCIN8, the internal pin multiplexer controlled by the Output Control Register A (OCRA) is programmed as follows.

```
****************************************************************
* Configure function of IO/MUXed shared pins
****************************************************************
ldp     #DP_PF2
splk    #0009h,OPCRA  ;ADCIN0 and ADCIN8 selected, all other are GPIO
```

Interrupt Module

Once that the AtoD Converter is correctly set, it will be used during every PWM interrupt to sample and convert the stator phase currents. A conversion is done as follows.

```
****************************************************************
* Current sampling - AD conversions
* only the 10 Least Significant bits are relevant
****************************************************************
    ldp     #DP_PF1
    splk    #1801h,ADC_CNTL1 ;ia and ib conversion start
                             ;ADCIN0 selected for ia A/D1
                             ;ADCIN8 selected for ib A/D2
conversion
    bit     ADC_CNTL1,8
    bcnd    conversion,tc   ;wait end of Conversion
    lacc    ADC_FIFO1,10
    ldp     #ia
    sach    ia
    ldp     #DP_PF1
    lacc    ADC_FIFO2,10
    ldp     #ib
    sach    ib
```

The TMS320F240 integrated ADC converts simultaneously ia and ib. The result of this conversion lays in the 10 upper bits of the ADC FIFOs. Therefore a left shift of 10 bit is performed to obtain the result of the conversion in the upper word of the accumulator. Care must be taken when the sign extension mode is on (SXM = 1), the FIFO values greater than 512 (bit b15 of FIFO equal to 1) will propagate a negative sign to the upper accumulator bits. Therefore, in the Current Scaling module, the upper bits of the accumulator are masked to keep the binary representation of the variables as follows.

```
      ldp     #ia
      lacc    ia
      and     #3ffh       ;mask upper bits
```

Adaptation to Specific Cases

In this particular application, a LEM was used to measure the phase current. A lower cost solution would consist in a simple shunt resistor as current sensor. This possibility has been studied in the application report [7].

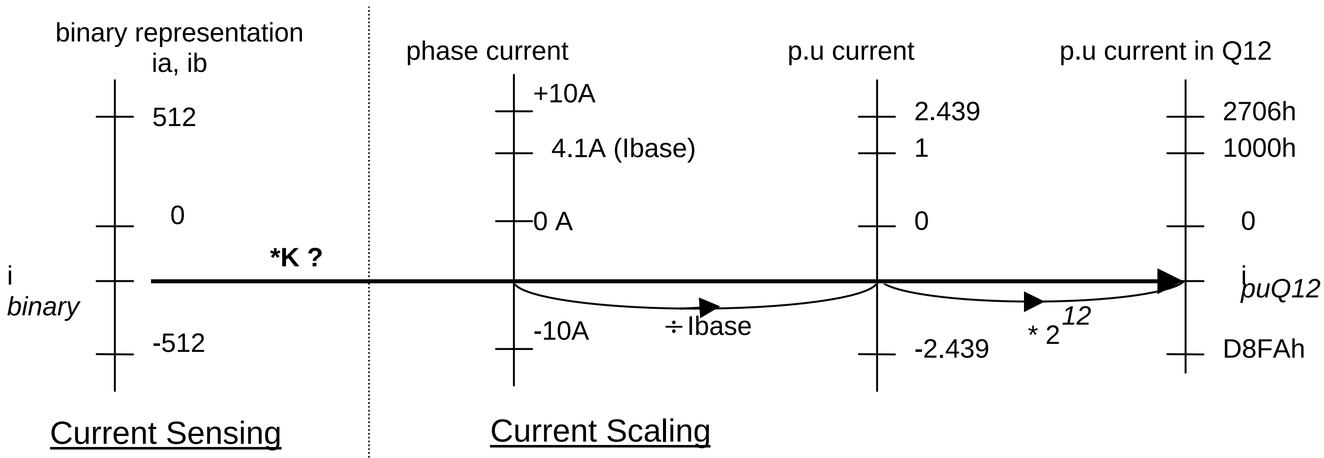
Current Scaling Module

The problem is to find a scaling factor K that makes the correspondence between the binary representation of the currents and their Q12 representation associated to the PU model of the currents.

Scale Change

Figure 21 depicts the scale changes needed to translate the binary representation of a current into its Per Unit Q12 representation.

Figure 21. Scaling Factor Representation



First, the binary representation of the current is modified in this module. An offset of 512 has been subtracted to contradict the analog offset of 2.5V that was previously introduced. +10A is now represented in binary by +512 and –10A by -512.

The problem here is the opposite of the one from the Sensing module. Now, given a binary representation of a current, the goal is to find a real number corresponding to the Per Unit value of the current.

In other words, the aim of the translation is to find a factor K such as :

$i_{puQ12} = i_{binary} * K$

For $i_{binary} = 512$, $i_{puQ12} = (I_{max}/I_{base}) * 2^{12} = 2.439*4096 = 9990$

Then : $K = i_{puQ12} / i_{binary} = 9990/512 = 19.51$

K has been determined by knowing the maximal value (10A corresponds to 512). It can also be determined from base values as follow:

Ibase= 4.1A corresponds to the binary representation 210 (D2h)

For $i_{binary} = 210$, $i_{puQ12} = (I_{base}/I_{base}) * 2^{12} = 4096$

Then $K = 4096/210 @ 19.51$

This method will be preferred to calculate translation factors knowing the base values.

Note that K is outside the Q12 dynamic range. The most appropriate format to accommodate this constant is the Q8 format.

In the application software, the constant K is called $K_{current}$ and its representation in Q8 is given by: $K_{current}$ = 19.51 ⇔1383h (Q8).

Translation Routine

The following routine performs the translation from the binary representation of the currents into their Per Unit Q12 format.

```
*********************************************
* Sampled current scaling
*
*********************************************
    ldp     #ia
    lacc    ia
    and     #3ffh  ;mask upper bits
    sub     #512   ;subtract the offset (2.5V) to have
                    ;positive and negative values of the current
    sacl    tmp
    spm     3
    lt      tmp
    mpy     Kcurrent
    pac
    sfr
    sfr
    sacl    ia              ;current ia, f 4.12 in PU
```

As previously mentioned, an offset of 512 is subtracted to the representation coming from the sensing. The result of this subtraction is stored in a temporary variable called *tmp*.

One of the most important point is to correctly tune this subtracted offset. In the FOCPMSM.ASM program, a SUB #440 is done instead of the SUB #512 instruction.

The next step is to multiply to multiply *tmp* by the scaling factor $K_{current}$.

The multiplication performed here is not as obvious as in base 10 representation of numbers. The problem is that the DSP is not able to perform directly the multiplication of a binary by a real number (19.51) as it is a fixed-point device.

The real operation performed is to multiply *tmp* by the Q8 fixed-point representation of $K_{current}$.

For example, when the binary number *tmp* is 210 (corresponding to the nominal current), the multiplication in base 10 would be:

$210 * 19.51 \approx 4097$ (0x1001h) represents 1 in Q12 format

As it is not possible to multiply directly by 19.51, the real multiplication performed is:

$210 * (19.51 * 2^8) = 210 * 4994$

The last operation is to retrieve the value of i_{puQ12} from this multiplication. To do so, the result of the multiplication is right shifted eight times which corresponds to a division by 2^8 (the instruction ”spm 3” performs 6 right shifts and two sfr complete the 2 right shifts).

Adaptation to Specific Cases

According to your specific motor characteristics (Nominal phase current) or the specific precision wanted (Qk format), it might be necessary to adapt the $K_{current}$ scaling factor.

For example, if the nominal phase current of the machine is 3.6A, the base value is 5.1A (and 261 its binary representation) then $K_{current} = 4096 / 261 = 15.69 \Leftrightarrow$ 0FB1h (Q8). If the Q12 precision doesn’t fit the user’s specific application (transient currents not greater than two times the nominal currents), a more precise format can be chosen. For example in the case of a 10bit ADC with a 3.6A of nominal current and a Q13 representation, $K_{current}$ would be:

$K_{current} = 8192 / 210 = 39.00$.

The other important point already mentioned is to tune the offset of the current measurement. You must adjust this offset to obtain sinusoidal stator currents. If the stator currents are wrongly interpreted in the software, the performance of the drive will be poor.

Mechanical Position Sensing and Scaling Module

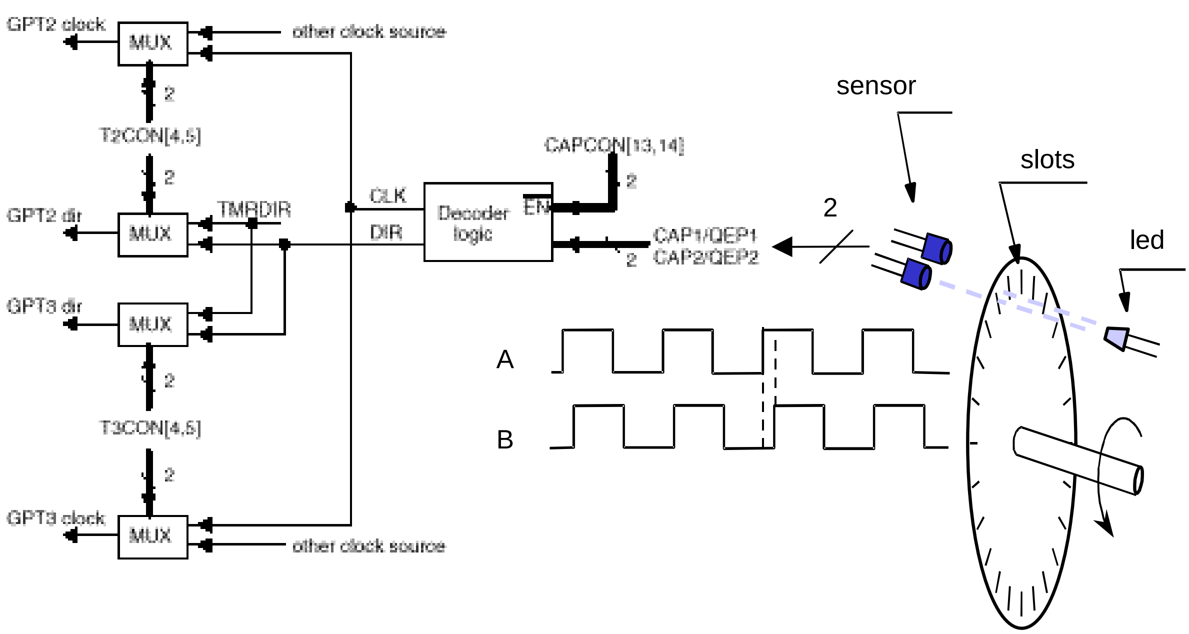
This module converts the number of pulses sent by the incremental encoder into an absolute mechanical position of the rotor shaft. The absolute mechanical position will be stored in the variable θ_m. It is possible to obtain an absolute mechanical position with the incremental encoder by physically locking the rotor in a known position. This stall is done in the start-up procedure. A zero is written in the encoder counter register thereby referencing the mechanical position to the locked position.

The number of encoder pulses detected between two PWM period is stored in the variable called “*encincr*”. These variables will be used afterwards to determine the electrical position of the rotor and the mechanical speed of the rotor in dedicated scaling modules.

Hardware Solution

The photo sensors of the encoder (Figure 22) are activated by the light of an internal LED. When the light is hidden, the sensor sends a logical “0”. When the light passes through one of the 1024 slots of the encoder, a logical “1” is sent. Two photo sendlogical information on Channel A and Channel B. The TMS320F240 on-chip QEP (Quadrature Encoder Pulse) detects the rising and falling edges of both channels. The count of the edges detected by the QEP is stored in the counter T3CNT. This counter is in fact related to the timer T3 that is automatically clocked by the QEP pulses when the QEP mode is selected.

Figure 22. Incremental Optical Encoder



The embedded encoder of this application generates 1024 pulses per mechanical revolution. Every slot generates 4 edges : 1 rising and 1 falling edge for both channels A and B. These edges are detected by the QEP, meaning that 4096 edges are detected per mechanical revolution. The QEP detects also the sense of rotation of the rotor shaft depending on the leading sequence (if Channel A signal are in advance or delayed compared to Channel B).

The number of edges is stored in T3CNT. Depending on the sense of revolution, T3CNT is incremented or decremented. Once that the QEP mode is selected, the Timer T3 wraps automatically around a period of FFFFh.

Sensing Scale Translation

The relative mechanical angular displacement calculated between two sampling period is equal to $\Delta q_m = \frac{\mathrm{T3CNT(t) - T3CNT(t - \Delta}t)}{EncPulses} * 360^\circ$, where Encpulses is here equal to 4096.

Accordingly, the absolute mechanical position is computed every sampling period as follow:

$\theta_m(t)=\theta_m(t-\Delta t)+\Delta\theta_m$

It has been chosen here to represent 360° mechanical by 1000h (EncPulses). The above equation is then simplified:

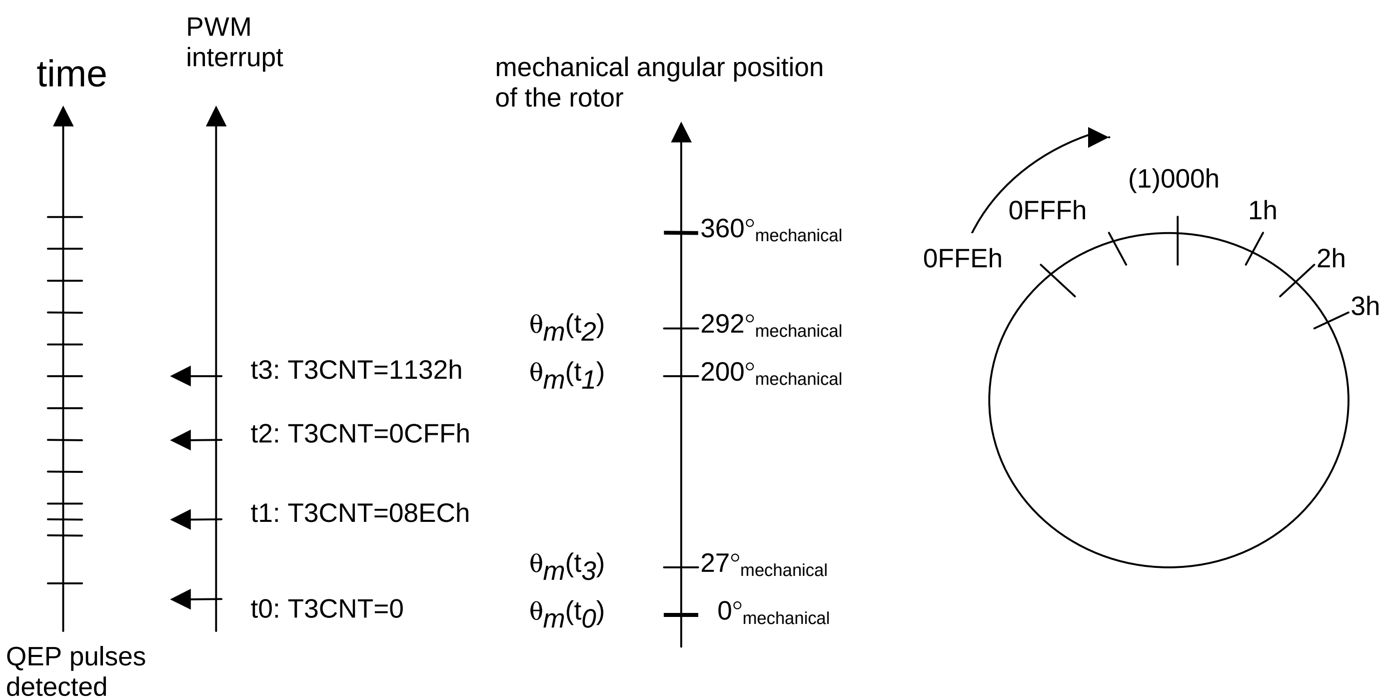
$\theta_m(t)=\theta_m old+encincr$

with encincr = T3CNT(t)-encoderold and encoderold=T3CNT(t-Δt).

A software rollover is also foreseen in case that the calculated angle exceeds 360°.

This sensing scale translation can be represented by the following diagram:

Figure 23. Sensing Scale



Associated low-level software routines

Initialization module

Both Timer3 Control register (T3CON) and Capture Unit Control register (CAPCON) are configured to enable the QEP functionality:

```
****************************************************************
* QEP initialization
****************************************************************
    ldp     #DP_EV
    splk    #0000h,T3CNT     ;reset counter register
    splk    #0ffffh,T3PER    ;configure period register
    splk    #9870h,T3CON     ;configure for QEP and enable Timer T3
    splk    #0E2F0h,CAPCON   ;T3 is selected as Time base for QEP
```

As the QEP pins are also shared with capture pins, it is necessary to set up the output control register (OCRB) to enable the QEP pins:

```
splk    #0038h,OPCRB    ;QEP pins selected and IOPC3
```

Interrupt Module

The following variables are used in the interrupt module:

- encincr : increment of T3CNT between two PWM interrupts
- θm : absolute mechanical position
- encoderold : last T3CNT value
- Encpulses : This constant is equal to four times the number of encoder pulses/mechanical rotation.

```
*** Encoder pulses reading
    ldp     #DP_EV
    lacc    T3CNT           ;read the encoder pulses
;   neg     ; if the encoder channels are plugged in the
            ; negative counting direction;
    ldp     #ia
    sacl    tmp
    subs    encoderold      ;increment T3CNT(k)-T3CNT(k-1)
    sacl    encincr
    add     teta_m          ;old mechanical position
    sacl    teta_m           ;new one
    sub     #Encpulses      ;soft rollover
    bcnd    encminmax,LT    ;
    sacl    teta_m
encminmax
    lacc    tmp
    sacl    encoderold      ;for next PWM ISR
```

In the above code, a software rollover is performed when the new position is greater than Encpulses to keep the value of θ_m in the range [0;1000h]. Notice that no software detection of the sense of rotation has been implemented. Depending on how the user wraps his two wires to the QEP inputs, a NEG instruction has to be added in order to get a positive increment in T3CNT.

Adaptation to Specific Cases

In this particular application, a 1024 incremental encoder was used to measure position and speed of the rotor. In the case of an encoder with a different resolution, the modifications are: change the value of the variable Encpulses and adapt the representation of 360 mechanical degrees.

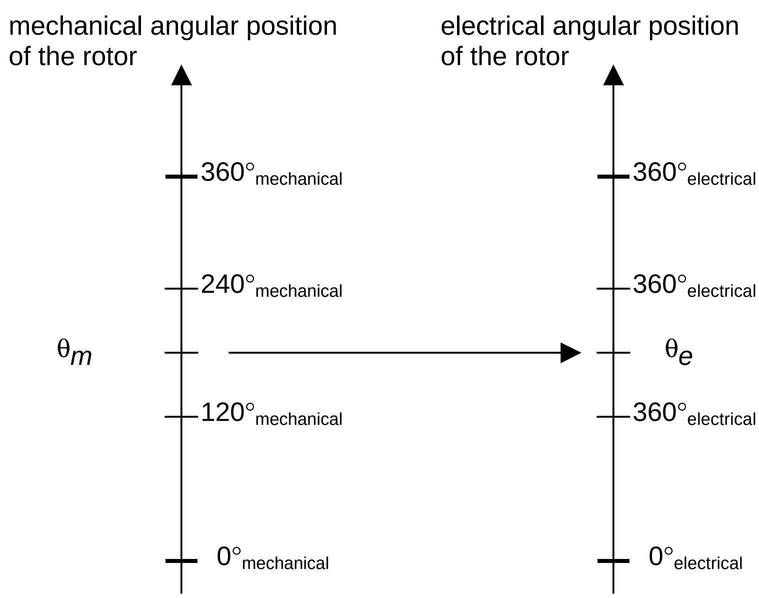
As previously mentioned, the user will have to add the NEG instruction, depending on how the channel A and B wires are connected to the QEP input pins.

Electrical Position Scaling Module

This module makes the correspondence between the relative mechanical position and the relative electrical position of the rotor.

Scale Change

Figure 24. Electrical Position Scaling



The aim of the translation is to find a scaling factor K such as $K^*\theta_m = \theta_e$

Given the relationship $\theta_e = \theta_m{}^* p$ (p is the number of pole pair), we obtain K=3 for a three pole pairs motor. In this application, K is called $K_{encoder}$.

Translation Routine

The following routine performs the translation from the mechanical position to the electrical.

```
*******************
* Teta calculation
*******************
    lt      teta_m         ;multiply mechanical pos by Kencoder
    mpyu    Kencoder
    pac
    and     #0fffh
    sacl    teta_e
```

The result of the multiplication is masked in order to have a software rollover of the electrical position when this position completed 360 electrical degrees.

Adaptation to Specific Cases

According to the Number of pole pairs of the motor, it might be necessary to adapt the $K_{encoder}$ coefficient as follow : $K_{encoder}=p$ where p is the number of pole pairs.

Mechanical Speed Scaling Module

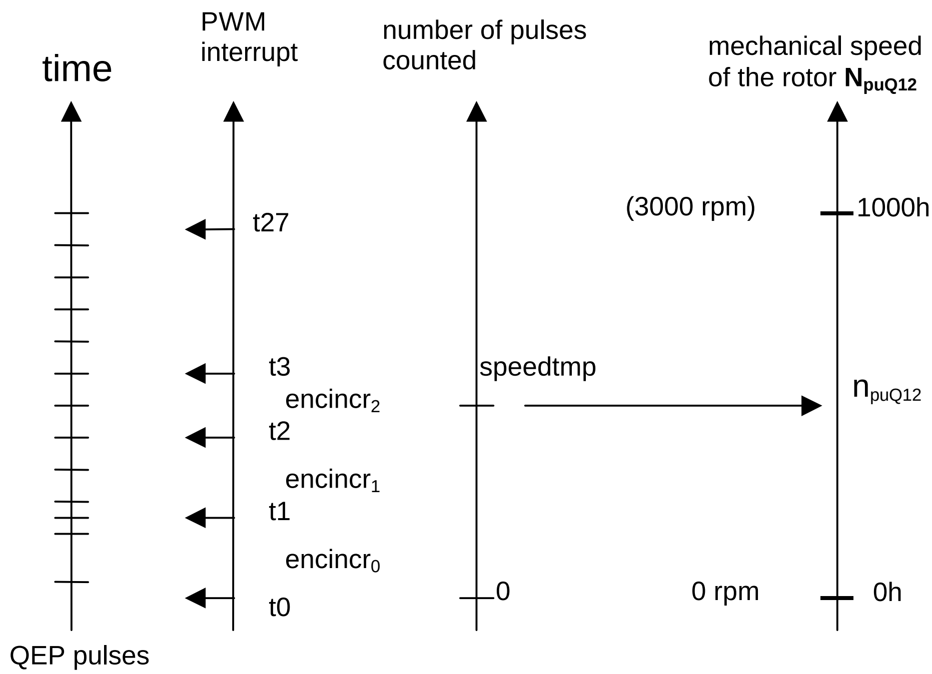
This section presents how to relate the increment of QEP pulses that appear between two speed sampling period to the Q12 representation of the mechanical speed associated its PU model.

The mechanical speed is computed periodically to provide a feedback to the PI speed regulator. The update of the speed information is not as critical as the update of the currents. The reason is that the mechanical response time constant of the system is very slow compared to the electrical one. Therefore, the mechanical speed is updated on a lower time base than the electrical quantities (updated every PWM interrupt).

A software counter called speedstep is incremented by one every PWM interrupt. Once it has reached its period value SPEEDSTEP, the calculation of the mechanical speed is done, taking into account the number of QEP pulses received from the last speed computation.

Scale Change

Figure 25. Mechanical Speed Scale



The aim of the translation is to find K such as

K**speedtmp* = n_{puQ12}

With: $$speedtmp = \sum_{k=0}^{k=SPEEDSTEP} encincr_k$$

When the software counter reaches its period called *SPEEDSTEP (SPEEDSTEP=28),* the time elapsed is 1.68 ms (28*60us).

Base speed n_{base} is 3000rpm ⇔50 mechanical revolutions per second

At base speed, the number of pulses counted by the QEP per second is 50 * 4096 = 204800 pulses. It means that *speedtmp* =204800*1.68*10^{-3} = 344 pulses

Then:

K= 4096/344 = 11.9069 ⇔ BE7h in 8.8f

In this application the coefficient K is called K_{speed}.

Translation Routine

```
*******************************************************
* Calculate speed and update reference speed variables
*******************************************************
    lacc    speedstep       ;are we in speed control loop ?
    sub     #1              ;
    sacl    speedstep       ;
    bcnd    nocalc,GT       ;if we aren't, skip speed calculation

*** Speed calculation from encoder pulses
    lt      speedtmp        ;multiply encoder pulses by Kspeed
                            ;(8.8 format constant) to have the value
                            ;of speed
    mpy     #Kspeed         ;
    pac                     ;
    rpt     #7              ;
    sfr                     ;
    sacl    n               ; n in PU Q12
    lacc    #0              ;zero speedtmp for next calculation
    sacl    speedtmp        ;
    lacc    #SPEEDSTEP      ;restore speedstep to the value
                            ;SPEEDSTEP
    sacl    speedstep       ;for next speed control loop
```

The result is stored in Q12 and is related to the PU model value of 3000rpm.

Adaptation to Specific Cases

According to your specific motor characteristics (Nominal Speed) and the specific speed sensing hardware (Encoder resolution), it might be necessary to adapt the K_{speed} coefficient.

For instance, in the case of a motor with a nominal speed of 1000 rpm, we would obtain the following K_{speed}:

$K_{speed} = 4096 / (\mathbf{16{,}66}*4096*1.68*10^{-3}) = 35.61$

If the encoder resolution is 1000 pulses per revolution, we would obtain a new K_{speed} such as: $K_{speed} = 4096 / (50*\mathbf{4000}**10^{-3}) = 12.19$

Experimental Results

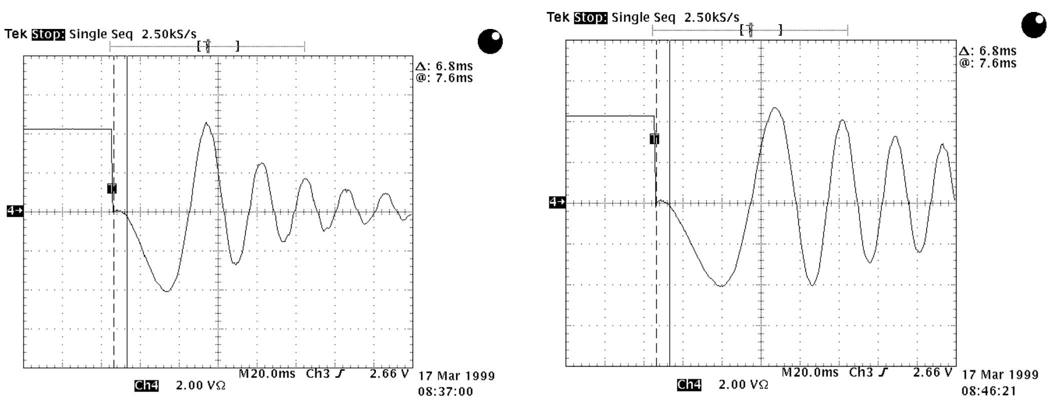
The motor has been mounted on a test bench with adjustable resistive torque in order to test the behavior of the drive in different configurations.

The DACs of the EVM are 12bit Digital to Analog converters. They have been used to output the evolution of the variables chosen by the user via the Graphic Interface. The DACs output are updated at the end of every PWM interrupt.

A written value of 0FFFh (4095) represents the maximal output voltage of +5V. The value 0800h is added in the assembly code to the outputted variables to provide a virtual ground at +2.5V in order to visualize positive and negative values.

Figure 26 shows the current in the stator phase A at start of rotation of the rotor. The left side of the figure is given without resistive torque applied whereas the right side of the figure is given with a resistive torque of 1Nm.

Figure 26. Transient Stator Phase A Current



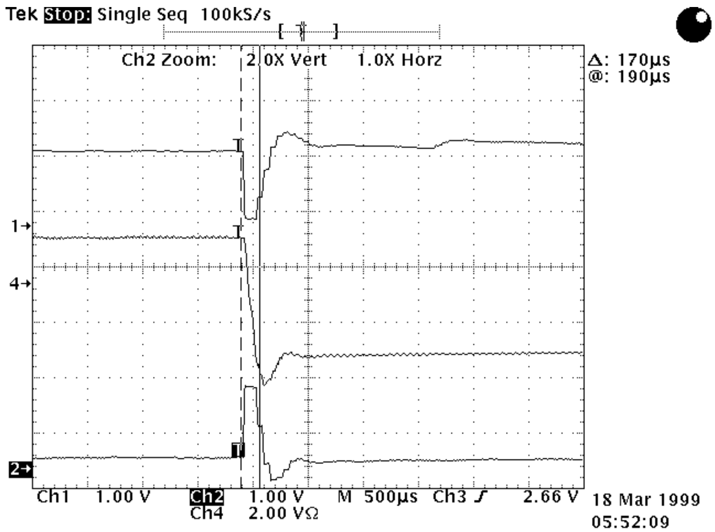
Ch4: stator phase current i_a

In Figure 26, it can be seen that the rotor has been first stalled by applying a constant stator reference current vector. The electrical angle chosen for this reference vector is –90 electrical degrees which corresponds physically to apply all the current to the stator phase a. The two other phases are the return paths for this current.

In this stalled mode, it can be seen that the current is correctly regulated to the nominal value, meaning that the fed current is controlled in amplitude. This control of the phase currents prevent the motor from heating.

Once the motion is started, the current fed to the phases depends on the resistive torque load applied to the rotor. In the case of no resistive torque, the steady-state currents are quasi null. In the case of the maximal resistive torque applied (2.2 Nm), the steady-state currents in the phases are equal to the nominal current (4.1A).

Figure 27. Transient Currents i_{sd}, i_{sq} at Start



Ch1: i_{sq} : current vector q projection (torque control)

Ch2: i_{sd} : current vector d projection (rotor flux control)

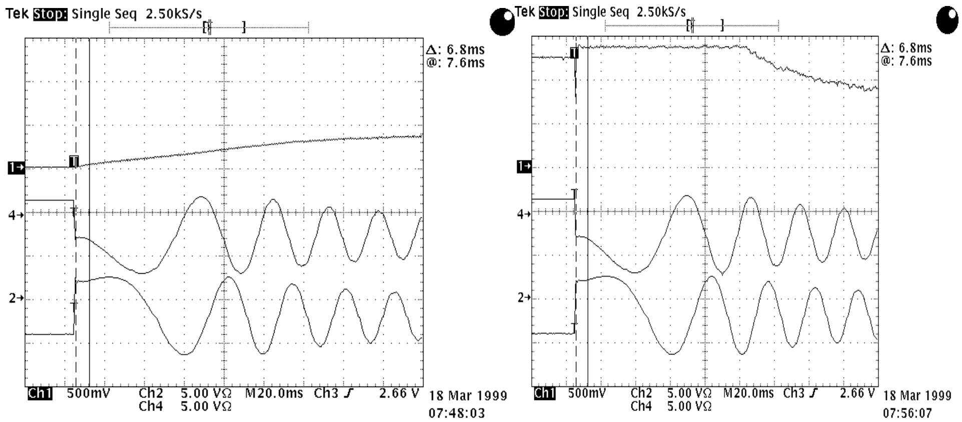
Ch4: i_c : real stator phase c current

Figure 27 shows the behavior of the calculated current projections at start. The reference speed is set to 500 rpm and no resistive torque is applied. The i_{sd} and i_{sq} projections are outputted on the DAC and their values are updated every PWM interrupt.

Current spikes appear in the picture. These spikes are due to the +90 electrical degrees shift of the stator current vector. The currents in the coils do not disappear instantaneously due to the electrical time constant of the motor. A certain amount of time is necessary to apply a new stator current vector that is +90° apart from the initial one. The i_{sd} and i_{sq} current regulators act to reduce the error between the projections and the reference currents.

Figure 28. Speed Transient from 0 to 1000 rpm

Figure 28 shows the behavior of the stator currents at start with a resistive torque load of 1 Nm.



Ch1 : n : mechanical speed *Ch1 : i_{sq} : torque control component*

Ch2 : i_b stator phase current *Ch2 : i_b stator phase current*

Ch4 : i_a stator phase current *Ch4 : i_a stator phase current*

It can be seen that when the motion starts, the torque control component i_{sq} is equal to the maximal authorized value. This maximal value is controlled by the i_{sq} PI current regulator and is equal to 1.1 of the PU model (10% more than the current base value is authorized). The amplitude of the stator phase currents are by the way always under control.

Table 4 relates the behavior of the motor under different loads at 500rpm and Table 5 at 1500rpm.

Table 4. Motor at 500 rpm

Torque Nm	0.1	0.5	1	1.5	2	2.2	2.3
Power W	5	25	50	76	102	113	119

The progression of the ratio power/torque is linear until the nominal torque. With a torque exceeding the nominal torque, the motor stalls.

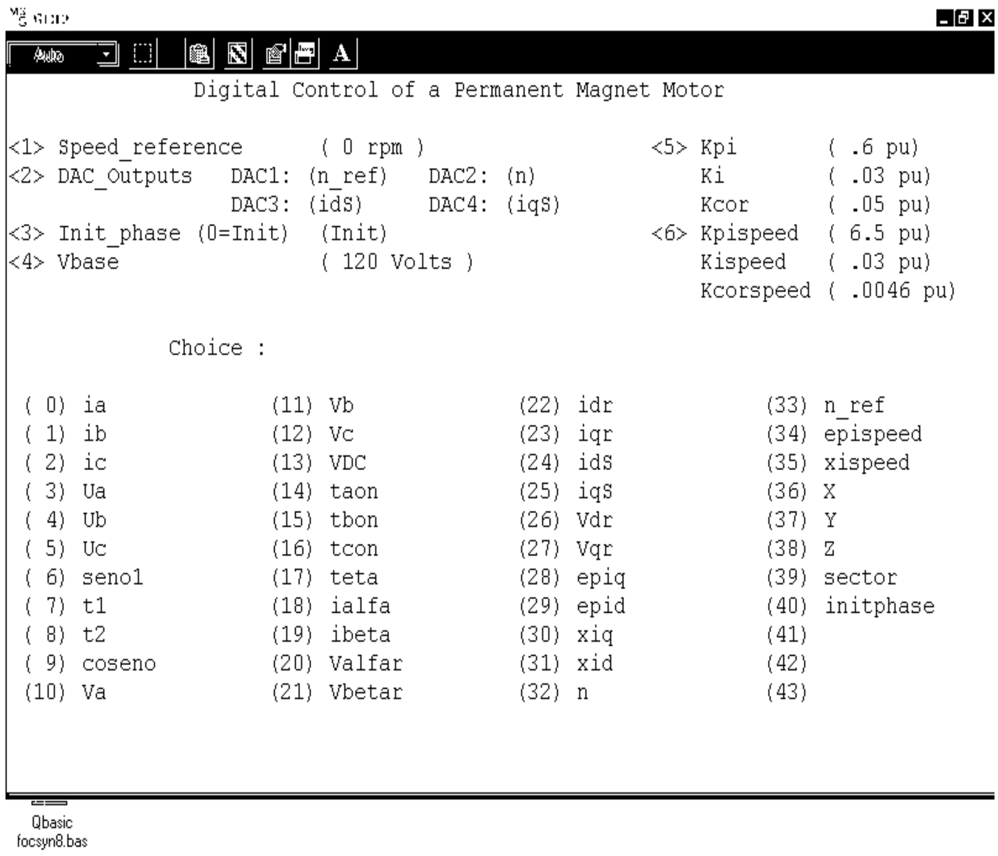
Table 5. Motor at 1500 rpm

Torque Nm	0.1	0.5	1	1.5	2	2.2	2.3
Power W	15	78	157	234	311	344	358

The progression of the ratio power/torque is also linear until the nominal torque. At nominal speed (3000rpm) the maximal power reached was 640W under nominal torque (2.2Nm).

User Interface

This section presents the screen picture that has been used as user interface. The corresponding Quick Basic program and the assembly communication software are given in Appendix. Below is a copy of the screen picture.



The top part of the Interface is dedicated to the six possible user commands.

- Selection of the mechanical speed reference <1>
- Selection of the outputted variables on the four DACS <2>
- Start the motion <3>
- Select the DC bus voltage (not used)
- Adjust the PI current controllers experimentally
- Adjust the PI speed controllers experimentally

The bottom part of the Interface corresponds to the choice of the variables to be outputted by the on-board 4 DACs. The DACs will output the real-time values of the variables every PWM interrupts. This feature is very useful during debugging or benchmarking phases.

Software Modularity

In this report, the software modules have been divided into blocks of codes that can be tuned individually. In order to ease the debug phase and individual module benchmarks, software switches have been added into the FOC algorithm. These switches consist of conditionnal assembly statements and thus induce no overhead in the execution time of the program.

The list of the individual module switches is given below:

```
*** Software switches ***
interrupt_module      .set  1
current_sensing       .set  1
current_scaling       .set  1
clarke                .set  1
park                  .set  1
inv_park              .set  1
isq_regulator         .set  1
isd_regulator         .set  1
speed_regulator       .set  1
svpwm                 .set  1
sine_table            .set  1
position_sensing      .set  1
position_scaling      .set  1
speed_scaling         .set  1
virtual_menu          .set  1
```

You can then select specific parts of the code to be assembled for test. When the switch is set to 0, the assembler doesn't take into account the block of code comprised between the .if *switch* and .endif statements. Therefore, the ADC conversion can be tested or the Space Vector Modulation without having to test the rest of the algorithm.

A special switch has been defined in order to allow the user to run the program without having to use the Graphical User Interface. This switch is called “virtual_menu”.

When virtual_menu is set to 0, a magnetic stall is performed until a software counter is decremented to 0 (stall_timer1, stall_timer2), then the motor is started with its nominal speed reference (3000 rpm).

Conclusion

The FOC control routine takes an average of 27.5 us for execution. The amount of program memory used for the whole program is lower than 1Kword. More details are given in the following tables:

The amount of memory used by this application is given in the following table:

Program Memory Used	Data Memory Used
Flash : 988 words (of the 16K)	B0 : 69 words (of the 256 available)
	B1 : 256 words (of 256)
	B2 : 25 words (of 32)

The timing benchmarks for the modules are given below:

Software Module	CPU cycles	Time
Current sensing	151	7.55 us
Current scaling	31	1.55 us
Park transform	9	0.45 us
Clarke transform	14	0.7 us
Clarke^{-1} transform	14	0.7 us
Motion sensing	19	0.95 us
Sine, Cosine calculation	32	1.6 us
Regulators (d,q, speed)	34 * 3	5.1 us
Space Vector PWM	166	8.3 us
TOTAL	538	**26.9 us**

The conversion time of the TMS320F240 is about 6.6 us (i_a and i_b are converted simultaneously). The new family of DSPs (TMS320F241, F243) have faster ADCs with 850ns conversion time for each current phase (1.7us to convert i_a and i_b).

It has been shown that the Field Oriented Control is a powerful algorithm that enables a real time control of the torque without ripples and stator phase currents amplitudes are always under control. The space vector algorithm is especially suited to generate the voltage references in co-ordination with the FOC.

References

1. Texas Instruments, *Field Orientated Control of Three phase AC-motors*, (BPRA073), December 1997.
2. Texas Instruments, *DSP Solution for Permanent Magnet Synchronous Motor*, (BPRA044), Nov. 1996.
3. Texas Instruments, *Clarke & Park Transforms on the TMS320C2xx*, (BPRA048), Nov. 1996.
4. T.J.E. Miller, *Brushless Permanent-Magnet and Reluctance Motor Drives*, Oxford Science Publications, ISBN 0-19-859369-4.
5. Riccardo Di Gabriele, *Controllo vettoriale di velocità di un motore asincrono mediante il Filtro di Kalman Esteso*, Tesi di Laurea, Università degli Studi di L'Aquila, Anno Accademico 1996-97
6. Roberto Petrella, *Progettazione e sviluppo di un sistema digitale basato su DSP e PLD per applicazione negli azionamenti elettrici*, Tesi di Laurea, Università degli Studi di L'Aquila, Anno Accademico 1995-96
7. Texas Instruments, *3-phase Current Measurements using a Single Line Resistor on the TMS320F240 DSP*, (BPRA077), May 1998.
8. Guy Grellet, Guy Clerc, *Actionneurs electriques*, Eyrolles, Nov 1996.
9. Jean Bonal, *Entrainements electriques a vitesse variable*, Lavoisier, Jan 1997.
10. Philippe Barret, *Regimes transitoires des machines tournantes electriques*, Eyrolles, Fev. 1987

Software Variables

The following list shows the different variables used in this control software and in the equations and schemes presented here.

i_a, i_b, i_c	stator phase currents
$i_{s\alpha}$, $i_{s\beta}$	stator current (α,β) components
i_{sd}, i_{sq}	stator current flux & torque components
i_{sdref}, i_{sqref}	flux and torque command
q_e	rotor flux electrical position
q_m	rotor flux mechanical position
V_{sdref}, V_{sqref}	(d,q) components of the reference stator voltage
$V_{s\alpha ref}$, $V_{s\beta ref}$	(α,β) components of the stator reference voltage
VDC	DC bus voltage
VDCinvT	constant using in the SVPWM
Va, Vb, Vc	(a,b,c,) components of the stator reference voltage
sector	sector variable used in SVPWM
t1, t2	time vector application in SVPWM
taon, tbon, tcon	PWM commutation instant
X, Y, Z	SVPWM variables
n, n_{ref}	speed and speed reference
$i_{sqrefmin}$, $i_{sqrefmax}$	speed regulator output limitation
V_{min}, V_{max}	d,q current regulator output limitation
K_i, K_{pi}, K_{cor}	current regulator parameters
K_{ispeed}, $K_{pispeed}$, $K_{corspeed}$	speed regulator parameters
x_{id}, x_{iq}, x_{ispeed}	regulator integral components
e_{pid}, e_{piq}, $e_{pispeed}$	d,q-axis, speed regulator errors
K_{speed}	4.12 speed formatting constant
$K_{current}$	4.12 current formatting constant
$K_{encoder}$	4.12 encoder formatting constant
SPEEDSTEP	speed loop period
speedstep	speed loop counter
encincr	encoder pulses storing variable
speedtmp	occurred pulses in SPEEDSTEP
sin, cos	sine and cosine of the rotor flux position

Appendix A. TMS320F240 FOC Software

```
***************************************************************
*                  TEXAS INSTRUMENTS                          *
*           Implementation of a Speed Field Orientated Control*
*           of 3phase PMSM  motor using the TMS320F240        *
*                                                             *
***************************************************************
*   File Name:  focpmsm.asm                                   *
*   Originator: Erwan SIMON                                   *
*   Description: PMSM Speed field oriented control            *
*   DSP development platform : TI TMS320F240 Evaluation Module *
*   Power board : IR2130 demo board          *
*   Motor :     Digiplan  MD3450                              *
*                                                             *
*   Last modified:   28/07/1999                               *
***************************************************************
* Auxiliary Register used                                     *
* ar4    pointer for context save stack                       *
* ar5    used as general purpose table pointer                *
***************************************************************


                .include ".\c240app.h"

***************************************************************
* Interrupt vector table                                      *
***************************************************************
      .global _c_int0

                .sect "vectors"
                b       _c_int0 ;reset vector
_c_int1         b       _c_int1
                b       _c_int2 ; PWM interrupt handler

stack           .usect "blockb2",15 ;space for ISR indirect context save
dac_val         .usect "blockb2",5  ;space for dac values in Page 0

                .sect "table"
sintab          .include        sine.tab
            ;sine wave look-up table for sine and cosine waves generation
            ;4.12 format

***************************************************************
* Variables and constants initializations
***************************************************************
                .data
*** current sampling constants
Kcurrent        .word   01383h  ;8.8 format (*19.5) sampled currents normalization
constant
                                ;ADCIN0 (ia current sampling)
                                ;ADCIN8 (ib current sampling)
*** axis transformation constants
SQRT3inv        .word   093dh   ;1/SQRT(3) 4.12 format
SQRT32          .word   0ddbh   ;SQRT(3)/2 4.12 format

*** PWM modulation constants
                .bss    _v_meas,1
PWMPRD          .set    258h    ;PWM Period=2*600 -> Tc=2*600*50ns=60us (50ns
                                ; resolution)
Tonmax          .set    0       ;minimum PWM duty cycle
MAXDUTY         .set    PWMPRD-2*Tonmax ;maximum utilization of the inverter
```

```
*** PI current regulators parameters
Ki              .word   07Ah    ;4.12 format = 0.03
Kpi             .word   999h    ;4.12 format = 0.60 (include period)
Kcor            .word   0cch    ;4.12 format = 0.05
                                ;Kcor = Ki/Kpi
*** PI speed regulators parameters
Kispeed         .word   7ah     ;4.12 format = 0.03
Kpispeed        .word   06800h  ;4.12 format = 6.5
Kcorspeed       .word   12h     ;4.12 format = 0.0046

*** Vqr and Vdr limitations
Vbase           .set    01000h  ;BEMF at base speed
Vmin            .set    0ec00h  ;4.12 format = -1.25 pu
Vmax            .set    01400h  ;4.12 format =  1.25 pu

*** Is and Idr limitations
ismax           .word   01199h  ;4.12 format = 4.51A Inominal+10%,
iSdrefmin       .set    0ee67h  ;4.12 format = -4.51A (1000h = Ibase)
iSdrefmax       .set    00000h  ;4.12 format =  0A   (1000h = Ibase)
zero            .word   0h

*** Initialization phase Iqr
iSqrefinit      .set    01000h  ;4.12 format = 4.1A (1000h = Ibase)

*** Encoder variables and constants
Kencoder        .word   3
                            ;this constant is used to convert encoder pulses
                            ;[0;4095] to an electric angle [0;360]=[0000h;1000h]
Encpulses       .set    4096    ;number of encoder pulses per mechanical
                                ;revolution

*** Speed and estimated speed calculation constants
Nbase           .set    1000h  ;Base speed
Kspeed          .set    0be7h  ;used to convert encoder pulses to a speed value
                               ;8.8 format = 11.9 (see manual for details about
                               ;this constant calculation)
                               ;base speed 3000rpm, PWMPR 258h
SPEEDSTEP       set    28      ;speed sampling period = current sampling period * 40

*** Speed and estimated speed calculation constants
    .bss    tmp,1              ;temporary variable (to use in ISR only !!!)
    .bss    option,1           ;virtual menu option number
    .bss    daout,1            ;address of the variable to send to the DACs
    .bss    daouttmp,1         ;value to send to the DACs

*** DAC displaying table starts here
    .bss    ia,1            ;phase current ia
    .bss    ib,1            ;phase current ib
    .bss    ic,1            ;phase current ic
    .bss    Ua,1            ; (not used)
    .bss    Ub,1            ; (not used)
    .bss    Uc,1            ; (not used)
    .bss    sin,1           ;generated sine wave value
    .bss    t1,1            ;SVPWM T1 (see SV PWM references for details)
    .bss    t2,1            ;SVPWM T2 (see SV PWM references for details)
    .bss    cos,1           ;generated cosine wave value
    .bss    Va,1            ;Phase 1 voltage for sector calculation
    .bss    Vb,1            ;Phase 2 voltage for sector calculation
    .bss    Vc,1            ;Phase 3 voltage for sector calculation
    .bss    VDC,1           ;DC Bus Voltage
    .bss    taon,1          ;PWM commutation instant phase 1
    .bss    tbon,1          ;PWM commutation instant phase 2
    .bss    tcon,1          ;PWM commutation instant phase 3
    .bss    teta_e,1        ;rotor electrical position in the range [0;1000h]
                            ;4.12 format = [0;360] degrees
```

```
    .bss    iSalfa,1        ;alfa-axis current
    .bss    iSbeta,1        ;beta-axis current
    .bss    vSal_ref,1      ;alfa-axis reference voltage
    .bss    vSbe_ref,1      ;beta-axis reference voltage
    .bss    iSdref,1        ;d-axis reference current
    .bss    iSqref,1        ;q-axis reference current
    .bss    iSd,1           ;d-axis current
    .bss    iSq,1           ;q-axis current
    .bss    vSdref,1        ;d-axis reference voltage
    .bss    vSqref,1        ;q-axis reference voltage
    .bss    epiq,1          ;q-axis current regulator error
    .bss    epid,1          ;d-axis current regulator error
    .bss    xiq,1           ;q-axis current regulator integral component
    .bss    xid,1           ;d-axis current regulator integral component
    .bss    n,1             ;speed
    .bss    n_ref,1         ;speed reference
    .bss    epispeed,1      ;speed error (used in speed regulator)
    .bss    xispeed,1       ;speed regulator integral component
    .bss    X,1             ;SVPWM variable
    .bss    Y,1             ;SVPWM variable
    .bss    Z,1             ;SVPWM variable
    .bss    sectordisp,1    ;SVPWM sector for display
    .bss    initphase,1     ;flag for initialization phase
    .bss    teta_m,1
    .bss    Vr,1            ;(not used)
    .bss    iSqrefmin,1     ;iSq min limitation
    .bss    iSqrefmax,1     ;iSq max limitation
*** END DAC displaying table

    .bss    sector,1        ;SVPWM sector
    .bss    serialtmp,1     ;serial communication temporary variable
    .bss    da1,1           ;DAC displaying table offset for DAC1
    .bss    da2,1           ;DAC displaying table offset for DAC2
    .bss    da3,1           ;DAC displaying table offset for DAC3
    .bss    da4,1           ;DAC displaying table offset for DAC4
    .bss    VDCinvT,1       ;used in SVPWM
    .bss    index,1         ;pointer used to access sine look-up table
    .bss    upi,1           ;PI regulators (current and speed) output
    .bss    elpi,1          ;PI regulators (current and speed) limitation error

    .bss    tmp1,1          ;tmp word
    .bss    accb,2          ;2 words buffer
    .bss    acc_tmp,2       ;2 words to allow swapping of ACC

    .bss    encoderold,1  ;encoder pulses value stored in the previous
                          ;sampling period
    .bss    encincr,1     ;encoder pulses increment between two
                          ;consecutive sampling periods
    .bss    speedtmp,1    ;used to accumulate encoder pulses increments
                          ;(to calculate the speed each speed sampling period)
    .bss    speedstep,1   ;sampling periods down counter used to
                          ;define speed
                          ;sampling period
*** END Variables and constants initializations


*** Software switches     ***
interrupt_module          .set 1
current_sensing           .set 1
current_scaling           .set 1
clarke                    .set 1
park                      .set 1
inv_park                  .set 1
isq_regulator             .set 1
isd_regulator             .set 1
```

```
speed_regulator          .set 1
svpwm                    .set 1
sine_table               .set 1
position_sensing         .set 1
position_scaling         .set 1
speed_scaling            .set 1

virtual_menu             .set 1

                         .bss stall_timer1,1
                         .bss stall_timer2,1
    .text
****************************************************************
*               Initialisation Module                          *
****************************************************************

_c_int0:
****************************
* C2xx core general settings
****************************
    clrc    CNF             ;set Block B0 as Data RAM (default)
    setc    OVM             ;saturate when overflow
    spm     0               ;no accumulator shift after multiplication
    setc    sxm             ;sign extension mode on
****************************************************************
* Initialize ar4 as the stack for context save
* space reserved: DARAM B2 60h-80h (page 0)
****************************************************************
    lar     ar4,#79h
    lar     ar5,#60h
****************************************************************
* Disable the watchdog timer                                   *
****************************************************************
    ldp     #DP_PF1
    splk    #006Fh, WD_CNTL
    splk    #05555h, WD_KEY
    splk    #0AAAAh, WD_KEY
    splk    #006Fh, WD_CNTL
*****************************************
* Initialization of the TMS320F240 Clocks
*****************************************
    splk    #00000010b,CKCR0;PLL disabled
                            ;LowPowerMode0
                            ;ACLK enabled
                            ;SYSCLK 5MHz
    splk    #10110001b,CKCR1;10MHz CLKIN
                            ;Do not divide PLL
                            ;PLL ratio x2 (CPUCLK=20MHz)
    splk    #10000011b,CKCR0;PLL enabled
                            ;LPM0
                            ;ACLK enabled
                            ;SYSCLK 10MHz
    splk    #40C0h,SYSCR    ;Set up CLKOUT to be SYSCLK
*****************************************
* F240 specific control register settings
*****************************************
    ; reset system control register
    lacc    SYSSR
    and     #69FFh
    sacl    SYSSR
****************************************************************
* A/D initialization
****************************************************************
    splk    #0003h,ADC_CNTL2;prescaler set for a 10MHz oscillator
    lacc    ADC_FIFO1       ;empty FIFO
```

```
    lacc    ADC_FIFO1
    lacc    ADC_FIFO2
    lacc    ADC_FIFO2
****************************************************************
* Serial communication initialization
****************************************************************
    splk    #00010111b,SCICCR   ;one stop bit, no parity, 8bits
    splk    #0013h,SCICTL1      ;enable RX, TX, clk
    splk    #0000h,SCICTL2      ;disable SCI interrupts
    splk    #0000h,SCIHBAUD     ;MSB |
    splk    #0082h,SCILBAUD     ;LSB |9600 Baud for sysclk 10MHz
    splk    #0022h,SCIPC2       ;I/O setting
    splk    #0033h,SCICTL1      ;end initialization

****************************************************************
* PWM Channel enable
* 74HC541 chip enable connected to IOPC3 of Digital input/output
****************************************************************
    ; Configure IO\function MUXing of pins
    ldp     #DP_PF2         ;Enable Power Security Function
    splk    #0009h,OPCRA    ;Ports A/B all IO except ADCs
    splk    #0038h,OPCRB    ;Port C as non IO function except IOPC0&3
    splk    #0FF08h,PCDATDIR;bit IOPC3

****************************************************************
* Incremental encoder initialization
****************************************************************
    ldp     #DP_EV
    splk    #0000h,T3CNT    ;configure counter register
    splk    #0ffffh,T3PER   ;configure period register
    splk    #9870h,T3CON    ;configure for QEP and enable Timer T3
    splk    #0E2F0h,CAPCON  ;T3 is selected as Time base for QEP
********************************************
* Wait state generator init
********************************************
    ldp     #ia
    splk    #04h,tmp
    out     tmp,WSGR
***************************************************************
* Variables initialization
****************************************************************
    ldp     #ia
    lacc    ismax
    sacl    iSqrefmax
    neg
    sacl    iSqrefmin
    zac
    sacl    iSqref
    sacl    iSdref
    sacl    n_ref
    sacl    iSdref
    sacl    index
    sacl    xid
    sacl    xiq
    sacl    xispeed
    sacl    upi
    sacl    elpi
    sacl    Va
    sacl    Vb
    sacl    Vc
    sacl    initphase
    sacl    da1
    lacc    #1
    sacl    da2
    lacc    #2
```

```
    sacl    da3
    lacc    #3
    sacl    da4
    splk    #015Ch,VDCinvT

    splk #07FFFh,stall_timer1
    splk #07FFFh,stall_timer2

*****************************************
* Event manager settings
*****************************************
    ldp     #DP_EV
    splk    #0666h,ACTR      ;Bits 15-12 not used, no space vector
                             ;PWM compare actions
                             ;PWM5/PWM6 - Active Low/Active High
                             ;PWM3/PWM4 - Active Low/Active High
                             ;PWM1/PWM2 - Active Low/Active High
    splk    #300,CMPR1       ;no current sent to the motor
    splk    #300,CMPR2
    splk    #300,CMPR3
    splk    #0000h,DBTCON    ;no dead band
    splk    #0207h,COMCON    ;Reload Full Compare when T1CNT=0
                             ;Disable Space Vector
                             ;Reload Full Compare Action when T1CNT=0
                             ;Enable Full Compare Outputs
                             ;Disable Simple Compare Outputs
                             ;Select GP timer1 as time base
                             ;Full Compare Units in PWM Mode
    splk    #8207h,COMCON    ;enable compare operation

    splk    #PWMPRD,T1PER    ;Set PWM interrupt period
    splk    #0,T1CNT
    splk    #0A800h,T1CON    ;Ignore Emulation suspend
                             ;Up/Down count mode
                             ;x/1 prescalar
                             ;Use own TENABLE
                             ;Disable Timer
                             ;Internal Clock Source
                             ;Reload Compare Register when T1CNT=0
                             ;Disable Timer Compare operation
    ; Enable Timer 1 operation
    lacc    T1CON
    or      #40h
    sacl    T1CON

****************************************************************
* Enable PWM control Interrupt
****************************************************************
    ; Clear EV IFR and IMR regs
    splk    #07FFh,IFRA
    splk    #00FFh,IFRB
    splk    #000Fh,IFRC

    ; Enable T1 Underflow Int
    splk    #0200h,IMRA
    splk    #0000h,IMRB
    splk    #0000h,IMRC

    ;Set IMR for INT2
    ldp     #0h
    lacc    #0FFh
    sacl    IFR              ;clear interrupt flags
    lacc    #0000010b
    sacl    IMR
    clrc    INTM             ;enable all interrupts
```

```
    b       menu             ;branch to menu loop
*******************************************************
* _c_int2 Interrupt Service Routine
* synchronization of the control algorithm with the PWM
* underflow interrupt
*******************************************************
_c_int2:
************************
* Context Saving
************************
    mar     *,ar4            ;AR4 active auxiliary reg (stack pointer)
    mar     *-
    sst     #1,*-            ;save status register 1
    sst     #0,*-            ;save status register 0
    sach    *-               ;save MS word of accu
    sacl    *-               ;save LS word of accu
* END Context Saving *
    mar     *,ar5            ;AR5 active auxiliary reg
    ldp     #DP_EV           ;DP points to Event Manager control reg page
    lacc    IVRA             ;read the interrupt vector


  .if   interrupt_module

ControlRoutine

  .if   current_sensing
****************************************************************
* Current sampling - AD conversions
* N.B. we will have to take only 10 bit (LSB)
****************************************************************
    ldp     #DP_PF1
    splk    #1801h,ADC_CNTL1;ia and ib conversion start
                             ;ADCIN0 selected for ia A/D1
                             ;ADCIN8 selected for ib A/D2
conversion
    bit     ADC_CNTL1,8
    bcnd    conversion,tc    ;wait approximatly 6us
    lacc    ADC_FIFO1,10
    ldp     #ia
    sach    ia
    ldp     #DP_PF1
    lacc    ADC_FIFO2,10
    ldp     #ib
    sach    ib
    .endif


*** Initialization phase
    lacl    initphase        ;are we in initialization phase ?
    bcnd    Run,NEQ
    lacc    #0fc00h          ;if yes, set teta = 0fc00h 4.12 format = -90
                             ;degrees
                             ;(align rotor with phase 1 flux)
    sacl    teta_e           ;
    lacc    #iSqrefinit      ;q-axis reference current = initialization
                             ;q-axis reference current
    sacl    iSqref           ;
    lacc    #0               ;zero some variables and flags
    sacl    iSdref           ;
    sacl    teta_m           ;
    sacl    encoderold       ;
    sacl    n                ;
    sacl    speedtmp         ;
    lacc    #SPEEDSTEP       ;restore speedstep to the value SPEEDSTEP
```

```
                          ; for next speed
                          ;control loop
    sacl    speedstep     ;
    ldp     #DP_EV
    splk    #0,T3CNT      ;zero Incremental Encoder value if
                          ;initialization step
    ldp     #initphase
    b       Init          ;there is no need to do position and
                          ;calculation
                          ;in initialization phase (the rotor is locked)

Run
 .if  position_sensing

*** Encoder pulses reading
    ldp     #DP_EV
    lacc    T3CNT           ;we read the encoder pulses and ...
    neg                     ;encoder plug in the opposite direction ?
    ldp     #ia
    sacl    tmp
    sub     encoderold      ;subtract the previous sampling period value
                            ;to have the increment that we'll
                            ;accumulate in encoder
    sacl    encincr         ;
    add     teta_m          ;
    bcnd    encmagzero,GT,EQ;here we start to normalize teta_m
                            ; value to the range [0;Encpulses-1]
    add     #Encpulses      ;the value of teta_m could be negative
                            ;it depends on the rotating direction
                            ;(depends on motor windings
                            ;to PWM Channels connections)
encmagzero
    sacl    teta_m          ;now teta_m value is positive but could be
                            ;greater than Encpulses-1
    sub     #Encpulses      ;we subtract Encpulses and we check whether
                            ;the difference is negative. If it is we
                            ;already have the right value in teta_m
    bcnd    encminmax,LT    ;
    sacl    teta_m          ;otherwise the value of teta_m is greater
                            ;than Encpulses and so we have to store the
encminmax                   ;right value ok, now teta_m contains the
                            ; right value in the range
    lacc    tmp             ;[0,Encpulses-1]
                            ;the actual value will be the old one during
                            ; the next sampling period
    sacl    encoderold
    .endif

    .if  position_scaling
*******************
* Teta calculation
*******************
    lt      teta_m    ;multiply teta_m pulses by Kencoder (4.12
                      ;format constant) to have the rotor
                      ;electrical position
    mpyu    Kencoder  ;encoder pulses = 0    -> teta = 0fffh = 0 degrees
    pac               ;encoder pulses = 1600 -> teta = 1fffh = 1*360
                      ;encoder pulses = 3200 -> teta = 2fffh = 2*360
    and     #0fffh
    sacl    teta_e
    .endif


    .if  speed_scaling
*******************************************************
```

```
* Calculate speed and update reference speed variables
*******************************************************
    lacc    speedstep       ;are we in speed control loop ? (SPEEDSTEP
                            ;times current control loop)
    sub     #1              ;
    sacl    speedstep       ;
    bcnd    nocalc,GT       ;if we aren't, skip speed calculation

*** Speed calculation from encoder pulses
    lt      speedtmp        ;multiply encoder pulses by Kspeed (8.8
                            ; format constant)
                            ;to have the value of speed
    mpy     #Kspeed         ;
    pac                     ;
    rpt     #7              ;
    sfr                     ;
    sacl    n
    lacc    #0              ;zero speedtmp for next calculation
    sacl    speedtmp        ;
    lacc    #SPEEDSTEP      ;restore speedstep to the value SPEEDSTEP
    sacl    speedstep       ;for next speed control loop
    .endif

    .if   speed_regulator
*****************************************************
* Speed regulator with integral component correction
*****************************************************
    lacc    n_ref
    sub     n
    sacl    epispeed
    lacc    xispeed,12
    lt      epispeed
    mpy     Kpispeed
    apac
    sach    upi,4
                            ;here start to saturate
    bit     upi,0
    bcnd    upimagzeros,NTC ;If value +ve branch
    lacc    iSqrefmin
    sub     upi
    bcnd    neg_sat,GT      ;if upi<iqrmin then branch to saturate
    lacc    upi             ;value of upi is valid
    b       limiters
neg_sat
    lacc    iSqrefmin       ;set acc to -ve saturated value
    b       limiters

upimagzeros                 ;Value is positive
    lacc    iSqrefmax
    sub     upi             ;
    bcnd    pos_sat,LT      ;if upi>iqrmax then branch to saturate
    lacc    upi             ;value of upi valid
    b       limiters
pos_sat
    lacc    iSqrefmax       ;set acc to +ve saturated value

limiters
    sacl    iSqref          ;Store the acc as reference value
    sub     upi
    sacl    elpi
    lt      elpi
    mpy     Kcorspeed
    pac
    lt      epispeed
    mpy     Kispeed
```

```
    apac
    add     xispeed,12
    sach    xispeed,4
    .endif


    .if   speed_scaling
****************************************************
* Encoder update
****************************************************
nocalc                       ;branch here if we don't have to calculate
                             ; the speed
    lacc    speedtmp         ;use the actual encoder increment to update
                             ;the increments accumulator used to
                             ; calculate the speed
    add     encincr          ;
    sacl    speedtmp         ;
    .endif

Init

  .if   current_scaling
*********************************************
* Sampled current scaling
* to nominal current 1000h <-> I_nominal
*********************************************
    ldp     #ia
    lacc    ia
    and     #3ffh
    sub     #440           ;then we have to subtract the offset (2.5V) to
                           ; have positive and negative values of the
                           ; sampled current
    sacl    tmp
    spm     3
    lt      tmp
    mpy     Kcurrent
    pac
    sfr
    sfr
    sacl    ia             ;sampled current ia, f 4.12
    lacc    ib
    and     #3ffh
    sub     #440
    sacl    tmp
    lt      tmp
    mpy     Kcurrent
    pac
    sfr
    sfr
    sacl    ib
    add     ia
    neg
    sacl    ic             ;ic = -(ib+ia)
    spm     0
    .endif


    .if   clarke
*********************************************
* (a,b,c) -> (alfa,beta) axis transformation
* iSalfa = ia
* iSbeta = (2 * ib + ia) / sqrt(3)
*********************************************
    lacc    ia
    sacl    iSalfa
```

```
    lacc    ib,1            ;iSbeta = (2 * ib + ia) / sqrt(3)
    add     ia
    sacl    tmp
    lt      tmp
    mpy     SQRT3inv        ;SQRT3inv = (1 / sqrt(3)) = 093dh
                            ;4.12 format = 0.577350269
    pac
    sach    iSbeta,4
    .endif


  .if   sine_table
*********************************************
* Sine and cosine wave calculation from
* teta values using sine look-up table
*********************************************
    lacc    teta_e        ;teta range is [0;1000h] 4.12 format = [0;360]
                          ;so we have a pointer (in the range [0;0ffh])
                          ;to the sine look-up table in the second and
                          ;third nibble
    rpt     #3
    sfr
    and     #0ffh         ;now ACC contains the pointer to access the table
    sacl    index
    add     #sintab
    sacl    tmp
    lar     ar5,tmp
    nop                   ; prevent pipeline conflict
    nop
    mar     *,ar5
    lacl    *
    nop
    sacl    sin            ;now we have sine value

    lacl    index         ;the same thing for cosine ... cos(teta)
                          ;sin(teta+90°)
    add     #040h         ;90 degrees = 40h elements of the table
    and     #0ffh
    sacl    index         ;we use the same pointer (we don't care)
    add     #sintab
    sacl    tmp
    lar     ar5,tmp
    lacc    *
    sacl    cos           ;now we have cosine value
    .endif


  .if   park
*********************************************
* d-axis and q-axis current calculation
* (alfa,beta) -> (d,q) axis transformation
* iSd = iSalfa * cos(teta_e) + iSbeta * sin(teta_e)
* iSq =-iSalfa * sin(teta_e) + iSbeta * cos(teta_e)
*********************************************
    lacc    #0
    lt      iSbeta          ;TREG0=iSbeta
    mpy     sin             ;PREG=iSbeta*sin(teta_e)
    lta     iSalfa          ;ACC+=PREG ; TREG0=iSalfa
    mpy     cos             ;PREG=iSalfa*cos(teta_e)
    mpya    sin             ;ACC+=PREG ; PREG=iSalfa*sin(teta_e)
    sach    iSd,4
    lacc    #0              ;ACC=0
    lt      iSbeta          ;TREG0=ibeta
    mpys    cos             ;ACC-=(PREG=iSalfa*sin(teta_e))
```

```
    apac                    ;ACC+=PREG
    sach    iSq,4
    .endif


  .if   isq_regulator
**************************************************************
* q-axis current regulator with integral component correction
* (iSq,iSqref)->(vSqref)
**************************************************************
iq_reg:
    lacc    iSqref
    sub     iSq
    sacl    epiq
    lacc    xiq,12
    lt      epiq
    mpy     Kpi
    apac
    sach    upi,4

    bit     upi,0
    bcnd    upimagzeroq,NTC
    lacc    #Vmin
    sub     upi
    bcnd    neg_satq,GT     ;if upi<Vmin branch to saturate
    lacc    upi             ;value of upi is valid
    b       limiterq
neg_satq
    lacc    #Vmin           ;set ACC to neg saturation
    b       limiterq

upimagzeroq                 ;Value was positive
    lacc    #Vmax
    sub     upi             ;
    bcnd    pos_satq,LT     ;if upi>Vmax branch to saturate
    lacc    upi             ;value of upi is valid
    b limiterq
pos_satq
    lacc    #Vmax           ;set ACC to pos saturation

limiterq
    sacl    vSqref          ;Save ACC as reference value
    sub     upi
    sacl    elpi
    lt      elpi
    mpy     Kcor
    pac
    lt      epiq
    mpy     Ki
    apac
    add     xiq,12
    sach    xiq,4
    .endif


  .if   isd_regulator
**************************************************************
* d-axis current regulator with integral component correction
* (iSd,iSdref)->(vSdref)
**************************************************************
    lacc    iSdref
    sub     iSd
    sacl    epid
    lacc    xid,12
    lt      epid
```

```
    mpy     Kpi
    apac
    sach    upi,4
    bit     upi,0
    bcnd    upimagzerod,NTC
    lacc    #Vmin
    sub     upi
    bcnd    neg_satd,GT      ;if upi<Vmin branch to saturate
    lacc    upi              ;value of upi is valid
    b       limiterd
neg_satd
    lacc    #Vmin            ;set ACC to neg saturation
    b       limiterd

upimagzerod                  ;Value was positive
    lacc    #Vmax
    sub     upi              ;
    bcnd    pos_satd,LT      ;if upi>Vmax branch to saturate
    lacc    upi              ;value of upi is valid
    b       limiterd
pos_satd
    lacc    #Vmax            ;set ACC to pos saturation
limiterd
    sacl    vSdref           ;Save ACC as reference value
    sub     upi
    sacl    elpi
    lt      elpi
    mpy     Kcor
    pac
    lt      epid
    mpy     Ki
    apac
    add     xid,12
    sach    xid,4
    .endif

  .if   inv_park
***********************************************
* alfa-axis and beta-axis voltages calculation
* (d,q) -> (alfa,beta) axis transformation
* vSbe_ref = vSqref * cos(teta_e) + vSdref * sin(teta_e)
* vSal_ref =-vSqref * sin(teta_e) + vSdref * cos(teta_e)
***********************************************
    lacc    #0
    lt      vSdref           ;TREG0=vSdref
    mpy     sin              ;PREG=vSdref*sin(teta_e)
    lta     vSqref           ;ACC+=PREG ; TREG0=vSqref
    mpy     cos              ;PREG=vSqref*cos(teta_e)
    mpya    sin              ;ACC+=PREG ; PREG=vSqref*sin(teta_e)
    sach    vSbe_ref,4
    lacc    #0               ;ACC=0
    lt      vSdref           ;TREG0=vSdref
    mpys    cos              ;ACC-=(PREG=vSqref*sin(teta_e))
    apac                     ;ACC+=PREG
    sach    vSal_ref,4
  .endif


  .if   svpwm
**********************************************
* Phase 1(=a) 2(=b) 3(=c) Voltage calculation
* (alfa,beta) -> (a,b,c) axis transformation
* modified exchanging alfa axis with beta axis
* for a correct sector calculation in SVPWM
* Va = vSbe_ref
```

```
* Vb = (-vSbe_ref + sqrt(3) * vSal_ref) / 2
* Vc = (-vSbe_ref - sqrt(3) * vSal_ref) / 2
***********************************************
    lt      vSal_ref        ;TREG0=vSal_ref
    mpy     SQRT32          ;PREG=vSal_ref*(SQRT(3)/2)
    pac                     ;ACC=PREG
    sub     vSbe_ref,11     ;ACC-=vSbe_ref*2^11
    sach    Vb,4
    pac                     ;ACC=PREG
    neg                     ;ACC=-ACC
    sub     vSbe_ref,11     ;ACC-=vSbe_ref*2^11
    sach    Vc,4
    lacl    vSbe_ref        ;ACC=vSbe_ref
    sacl    Va              ;Va=ACCL
**************************************
* SPACE VECTOR Pulse Width Modulation
* (see SVPWM references)
**************************************
    lt      VDCinvT
    mpy     SQRT32
    pac
    sach    tmp,4
    lt      tmp
    mpy     vSbe_ref
    pac
    sach    X,4
    lacc    X               ;ACC = vSbe_ref*K1
    sach    accb
    sacl    accb+1          ;ACCB = vSbe_ref*K1
    sacl    X,1             ;X=2*vSbe_ref*K1
    lt      VDCinvT
    splk    #1800h,tmp
    mpy     tmp             ;implement mpy #01800h
    pac
    sach    tmp,4
    lt      tmp
    mpy     vSal_ref
    pac
    sach    tmp,4
    lacc    tmp             ;reload ACC with vSal_ref*K2
    add     accb+1
    add     accb,16
    sacl    Y               ;Y = K1 * vSbe_ref + K2 * vSal_ref
    sub     tmp,1
    sacl    Z               ;Z = K1 * vSbe_ref - K2 * vSal_ref

*** 60 degrees sector determination
    lacl    #0
    sacl    sector
    lacc    Va
    bcnd    Va_neg,LEQ      ;If Va<0 do not set bit 1 of sector
    lacc    sector
    or      #1
    sacl    sector          ;implement opl #1,sector
Va_neg      lacc    Vb
    bcnd    Vb_neg,LEQ      ;If Vb<0 do not set bit 2 of sector
    lacc    sector
    or      #2
    sacl    sector          ;implement opl #2,sector
Vb_neg      lacc    Vc
    bcnd    Vc_neg,LEQ      ;If Vc<0 do not set bit 3 of sector
    lacc    sector
    or      #4
    sacl    sector          ;implement opl #4,sector
Vc_neg
```

```
*** END 60 degrees sector determination

*** T1 and T2 (= t1 and t2) calculation depending on the sector number
    lacl    sector          ;(see SPACE VECTOR Modulation references for
                            ;details)
    sub     #1
    bcnd    no1,NEQ
    lacc    Z
    sacl    t1
    lacc    Y
    sacl    t2
    b       t1t2out
no1
    lacl    sector
    sub     #2
    bcnd    no2,NEQ
    lacc    Y
    sacl    t1
    lacc    X
    neg
    sacl    t2
    b       t1t2out
no2
    lacl    sector
    sub     #3
    bcnd    no3,NEQ
    lacc    Z
    neg
    sacl    t1
    lacc    X
    sacl    t2
    b       t1t2out
no3
    lacl    sector
    sub     #4
    bcnd    no4,NEQ
    lacc    X
    neg
    sacl    t1
    lacc    Z
    sacl    t2
    b       t1t2out
no4
    lacl    sector
    sub     #5
    bcnd    no5,NEQ
    lacc    X
    sacl    t1
    lacc    Y
    neg
    sacl    t2
    b       t1t2out
no5
    lacc    Y
    neg
    sacl    t1
    lacc    Z
    neg
    sacl    t2
t1t2out
    lacc    t1              ;t1 and t2 minumum values must be Tonmax
    sub     #Tonmax
    bcnd    t1_ok,GEQ       ;if t1>Tonmax then t1_ok
    lacl    #Tonmax
    sacl    t1
```

```
t1_ok
    lacc    t2
    sub     #Tonmax
    bcnd    t2_ok,GEQ       ;if t2>Tonmax then t2_ok
    lacl    #Tonmax
    sacl    t2
t2_ok
*** END t1 and t2 calculation

    lacc    t1              ;if t1+t2>2*Tonmax we have to saturate t1 and t2
    add     t2              ;
    sacl    tmp             ;
    sub     #MAXDUTY        ;
    bcnd    nosaturation,LT,EQ

*** t1 and t2 saturation
    lacc    #MAXDUTY,15     ;divide MAXDUTY by (t1+t2)
    rpt     #15             ;
    subc    tmp             ;
    sacl    tmp             ;
    lt      tmp             ;calculate saturate values of t1 and t2
    mpy     t1              ;t1 (saturated)=t1*(MAXDUTY/(t1+t2))
    pac                     ;
    sach    t1,1            ;
    mpy     t2              ;t2 (saturated)=t2*(MAXDUTY/(t1+t2))
    pac                     ;
    sach    t2,1            ;
*** END t1 and t2 saturation

nosaturation
*** taon,tbon and tcon calculation
    lacc    #PWMPRD         ;calculate the commutation instants taon,
                            ;tbon and tcon
    sub     t1              ;of the 3 PWM channels
    sub     t2              ;taon=(PWMPRD-t1-t2)/2
    sfr                     ;
    sacl    taon            ;
    add     t1              ;tbon=taon+t1
    sacl    tbon            ;
    add     t2              ;tcon=tbon+t2
    sacl    tcon            ;
*** END taon,tbon and tcon calculation

*** sector switching
    lacl    sector         ;depending on the sector number we have
    sub     #1             ;to switch the calculated taon, tbon and tcon
    bcnd    nosect1,NEQ    ;to the correct PWM channel
                           ;(see SPACE VECTOR Modulation references for
                           ; details)
    bldd    tbon,#CMPR1    ;sector 1
    bldd    taon,#CMPR2
    bldd    tcon,#CMPR3
    b       dacout
nosect1
    lacl    sector
    sub     #2
    bcnd    nosect2,NEQ
    bldd    taon,#CMPR1    ;sector 2
    bldd    tcon,#CMPR2    ;
    bldd    tbon,#CMPR3    ;
    b       dacout
nosect2
    lacl    sector
    sub     #3
    bcnd    nosect3,NEQ
```

```
    bldd    taon,#CMPR1     ;sector 3
    bldd    tbon,#CMPR2     ;
    bldd    tcon,#CMPR3     ;
    b       dacout
nosect3
    lacl    sector
    sub     #4
    bcnd    nosect4,NEQ
    bldd    tcon,#CMPR1     ;sector 4
    bldd    tbon,#CMPR2     ;
    bldd    taon,#CMPR3     ;
    b       dacout
nosect4
    lacl    sector
    sub     #5
    bcnd    nosect5,NEQ
    bldd    tcon,#CMPR1     ;sector 5
    bldd    taon,#CMPR2     ;
    bldd    tbon,#CMPR3     ;
    b       dacout
nosect5
    bldd    tbon,#CMPR1     ;sector 6
    bldd    tcon,#CMPR2     ;
    bldd    taon,#CMPR3     ;
*** END sector switching
*** END * SPACE VECTOR Pulse Width Modulation
    .endif

dacout
****************************************************************
* DAC output of channels 'da1', 'da2', 'da3' and 'da4'         *
* Output on 12 bit Digital analog Converter                    *
* 5V equivalent to FFFh                                        *
****************************************************************
    ldp     #sector
    lacc    sector,7        ;scale sector by 2^7 to have good displaying
    sacl    sectordisp      ;only for display purposes

*** DAC out channel 'da1'
    lacc    #ia             ;get the address of the first elements
    add     da1             ;add the selected output variable offset
                            ; 'da1' sent by the terminal
    sacl    daout           ;now daout contains the address of the
                            ; variable to send to DAC1
    lar     ar5,daout       ;store it in AR5

    lacc    *               ;indirect addressing, load the value to send out
                            ;the following 3 instructions are required to
                            ;adapt the numeric format to the DAC resolution
    sfr                     ;on a 12 bit DAC, +/- 2000h = [0,5] Volt
    sfr                     ;-2000h is 0 Volt
    add     #800h           ;0 is 2.5 Volt.
    sacl    daouttmp        ;to prepare the triggering of DAC1 buffer
    out     daouttmp,DAC0_VAL

*** DAC out channel 'da2'
    lacc    #ia             ;get the address of the first elements
    add     da2             ;add the selected output variable offset
                            ;'da1' sent by the terminal
    sacl    daout           ;now daout contains the address of the
                            ; variable to send to DAC1
    lar     ar5,daout       ;store it in AR5

    lacc    *               ;indirect addressing, load the value to send out
                            ;the following 3 instructions are required to
```

```
                                ;adapt the numeric format to the DAC resolution
    sfr                         ;we have 10 bit DAC, we want to have the
                                ;number 2000h = 5 Volt
    sfr
    add     #800h               ;
    sacl    daouttmp            ;to prepare the triggering of DAC1 buffer
    out     daouttmp,DAC1_VAL

*** DAC out channel 'da3'
    lacc    #ia                 ;get the address of the first elements
    add     da3                 ;add the selected output variable offset 'da1'
                                ;sent by the terminal
    sacl    daout               ;now daout contains the address of the variable
                                ;to send to DAC1
    lar     ar5,daout           ;store it in AR5

    lacc    *                   ;indirect addressing, load the value to send out
                                ;the following 3 instructions are required to
                                ;adapt the numeric format to the DAC resolution
    sfr                         ;we have 10 bit DAC, we want to have the number
                                ;2000h = 5 Volt
    sfr
    add     #800h
    sacl    daouttmp            ;to prepare the triggering of DAC1 buffer
    out     daouttmp,DAC2_VAL

*** DAC out channel 'da4'
    lacc    #ia                 ;get the address of the first elements
    add     da4                 ;add the selected output variable offset 'da1'
;sent by the terminal
    sacl    daout               ;now daout contains the address of the
                                ;variable to send to DAC1
    lar     ar5,daout           ;store it in AR5

    lacc    *                   ;indirect addressing, load the value to send out
                                ;the following 3 instructions are required
                                ;to adapt the numeric format to the DAC resolution
    sfr                         ;we have 10 bit DAC, we want to have the
                                ;number 2000h = 5 Volt
    sfr
    add     #800h
    sacl    daouttmp            ;to prepare the triggering of DAC1 buffer
    out     daouttmp,DAC3_VAL
*** END DAC out

    OUT     tmp,DAC_VAL         ;start D to A convertion


*** END: PWM enable

    b       ContextRestoreReturn
*END ControlRoutine


  .endif

ContextRestoreReturn
**************************************
* Context restore and Return
**************************************
    larp    ar4
    mar     *+
    lacl    *+                  ;Accu. restored for context restore
    add     *+,16
    lst     #0,*+
```

```
    lst     #1,*+
    clrc    INTM
    ret
* END Context Restore and Return *


***************
* Virtual Menu
***************
menu


 .if  virtual_menu

    ldp     #DP_PF1
    bit     SCIRXST,BIT6        ;is there any character available ?
    bcnd    menu,ntc            ;if not repeat the cycle (polling)
    lacc    SCIRXBUF
    and     #0ffh               ;only 8 bits !!!
    ldp     #option             ;if yes, get it and store it in option
    sacl    option              ;now in option we have the option number
                                ;of the virtual menu
    sub     #031h               ;is it option 1 ?
    bcnd    notone,neq          ;if not branch to notone

*****************************
* Option 1): Speed reference
*****************************
navail11
    ldp     #DP_PF1
    bit     SCIRXST,BIT6        ;is there any character available (8 LSB)?
    bcnd    navail11,ntc        ;if not repeat the cycle (polling)
    lacc    SCIRXBUF
    and     #0FFh               ;take the 8 LSB
    ldp     #serialtmp
    sacl    serialtmp           ;if yes, get it and store it in serialtmp
navail12
    ldp     #DP_PF1
    bit     SCIRXST,BIT6        ;8 MSB available ?
    bcnd    navail12,ntc        ;if not repeat the cycle (polling)
    lacc    SCIRXBUF,8          ;load ACC the upper byte
    ldp     #serialtmp
    add     serialtmp           ;add ACC with lower byte
    sacl    n_ref               ;store it
    b       menu                ;return to the main polling cycle
*** END Option 1): speed reference

notone
    lacc    option
    sub     #032h               ;is it option 2 ?
    bcnd    nottwo,neq          ;if not branch to nottwo

*****************************
* Option 2): DAC update
*****************************
navail21
    ldp     #DP_PF1
    bit     SCIRXST,BIT6        ;is there any character available (8 LSB)?
    bcnd    navail21,ntc        ;if not repeat the cycle (polling)
    lacc    SCIRXBUF
    and     #0FFh               ;take the 8 LSB
    ldp     #da1
    sacl    da1                 ;if yes, get it and store it in da1
navail22
    ldp     #DP_PF1
```

```
    bit     SCIRXST,BIT6      ;is there any character available (8 LSB)?
    bcnd    navail22,ntc      ;if not repeat the cycle (polling)
    lacc    SCIRXBUF
    and     #0FFh             ;take the 8 LSB
    ldp     #da1
    sacl    da2               ;if yes, get it and store it in da2
navail23
    ldp     #DP_PF1
    bit     SCIRXST,BIT6      ;is there any character available (8 LSB)?
    bcnd    navail23,ntc      ;if not repeat the cycle (polling)
    lacc    SCIRXBUF
    and     #0FFh             ;take the 8 LSB
    ldp     #da1
    sacl    da3               ;if yes, get it and store it in da3
navail24
    ldp     #DP_PF1
    bit     SCIRXST,BIT6      ;is there any character available (8 LSB)?
    bcnd    navail24,ntc      ;if not repeat the cycle (polling)
    lacc    SCIRXBUF
    and     #0FFh             ;take the 8 LSB
    ldp     #da1
    sacl    da4               ;if yes, get it and store it in da4
    b       menu              ;return to the main polling cycle
*** END Option 2): DAC update

nottwo
    lacc    option
    sub     #033h             ;is it option 3 ?
    bcnd    notthree,neq      ;if not branch to notthree

*****************************
* Option 3): initphase
*****************************
navail31
    ldp     #DP_PF1
    bit     SCIRXST,BIT6      ;is there any character available (8 LSB)?
    bcnd    navail31,ntc      ;if not repeat the cycle (polling)
    lacc    SCIRXBUF
    and     #0FFh             ;take the 8 LSB
    ldp     #serialtmp
    sacl    serialtmp         ;if yes, get it and store it in serialtmp
navail32
    ldp     #DP_PF1
    bit     SCIRXST,BIT6      ;8 MSB available ?
    bcnd    navail32,ntc      ;if not repeat the cycle (polling)
    lacc    SCIRXBUF,8        ;load ACC the upper byte
    ldp     #serialtmp
    add     serialtmp         ;add ACC with lower byte
    sacl    initphase         ;store it
    b       menu              ;return to the main polling cycle
*** END Option 3): initphase

notthree
    lacc    option
    sub     #034h             ;is it option 4 ?
    bcnd    notfour,neq       ;if not branch to notfour

*****************************
* Option 4): vDCinvTc
*****************************
navail41
    ldp     #DP_PF1
    bit     SCIRXST,BIT6      ;is there any character available (8 LSB)?
    bcnd    navail41,ntc      ;if not repeat the cycle (polling)
    lacc    SCIRXBUF
```

```
    and     #0FFh              ;take the 8 LSB
    ldp     #serialtmp
    sacl    serialtmp          ;if yes, get it and store it in serialtmp
navail42
    ldp     #DP_PF1
    bit     SCIRXST,BIT6       ;8 MSB available ?
    bcnd    navail42,ntc       ;if not repeat the cycle (polling)
    lacc    SCIRXBUF,8         ;load ACC the upper byte
    ldp     #serialtmp
    add     serialtmp          ;add ACC with lower byte
    sacl    VDCinvT            ;store it
    b       menu               ;return to the main polling cycle
*** END Option 4): vDCinvTc

notfour
    lacc    option
    sub     #035h              ;is it option 5 ?
    bcnd    notfive,neq        ;if not branch to notfive

*****************************
* Option 5): Kpi, Ki, Kcor
*****************************
navail51
    ldp     #DP_PF1
    bit     SCIRXST,BIT6       ;is there any character available (8 LSB)?
    bcnd    navail51,ntc       ;if not repeat the cycle (polling)
    lacc    SCIRXBUF
    and     #0FFh              ;take the 8 LSB
    ldp     #serialtmp
    sacl    serialtmp          ;if yes, get it and store it in serialtmp
navail52
    ldp     #DP_PF1
    bit     SCIRXST,BIT6       ;8 MSB available ?
    bcnd    navail52,ntc       ;if not repeat the cycle (polling)
    lacc    SCIRXBUF,8         ;load ACC the upper byte
    ldp     #serialtmp
    add     serialtmp          ;add ACC with lower byte
    sacl    Kpi                ;store it
navail53
    ldp     #DP_PF1
    bit     SCIRXST,BIT6       ;is there any character available (8 LSB)?
    bcnd    navail53,ntc       ;if not repeat the cycle (polling)
    lacc    SCIRXBUF
    and     #0FFh              ;take the 8 LSB
    ldp     #serialtmp
    sacl    serialtmp          ;if yes, get it and store it in serialtmp
navail54
    ldp     #DP_PF1
    bit     SCIRXST,BIT6       ;8 MSB available ?
    bcnd    navail54,ntc       ;if not repeat the cycle (polling)
    lacc    SCIRXBUF,8         ;load ACC the upper byte
    ldp     #serialtmp
    add     serialtmp          ;add ACC with lower byte
    sacl    Ki                 ;store it
navail55
    ldp     #DP_PF1
    bit     SCIRXST,BIT6       ;is there any character available (8 LSB)?
    bcnd    navail55,ntc       ;if not repeat the cycle (polling)
    lacc    SCIRXBUF
    and     #0FFh              ;take the 8 LSB
    ldp     #serialtmp
    sacl    serialtmp          ;if yes, get it and store it in serialtmp
navail56
    ldp     #DP_PF1
    bit     SCIRXST,BIT6       ;8 MSB available ?
```

```
    bcnd    navail56,ntc        ;if not repeat the cycle (polling)
    lacc    SCIRXBUF,8          ;load ACC the upper byte
    ldp     #serialtmp
    add     serialtmp           ;add ACC with lower byte
    sacl    Kcor                ;store it
    b       menu                ;return to the main polling cycle
*** END Option

notfive
    lacc    option
    sub     #036h               ;is it option 6 ?
    bcnd    notsix,neq          ;if not branch to notsix

*****************************
* Option 6): Kpispeed , Kispeed , Kcorspeed
*****************************
navail61
    ldp     #DP_PF1
    bit     SCIRXST,BIT6        ;is there any character available (8 LSB)?
    bcnd    navail61,ntc        ;if not repeat the cycle (polling)
    lacc    SCIRXBUF
    and     #0FFh               ;take the 8 LSB
    ldp     #serialtmp
    sacl    serialtmp           ;if yes, get it and store it in serialtmp
navail62
    ldp     #DP_PF1
    bit     SCIRXST,BIT6        ;8 MSB available ?
    bcnd    navail62,ntc        ;if not repeat the cycle (polling)
    lacc    SCIRXBUF,8          ;load ACC the upper byte
    ldp     #serialtmp
    add     serialtmp           ;add ACC with lower byte
    sacl    Kpispeed            ;store it
navail63
    ldp     #DP_PF1
    bit     SCIRXST,BIT6        ;is there any character available (8 LSB)?
    bcnd    navail63,ntc        ;if not repeat the cycle (polling)
    lacc    SCIRXBUF
    and     #0FFh               ;take the 8 LSB
    ldp     #serialtmp
    sacl    serialtmp           ;if yes, get it and store it in serialtmp
navail64
    ldp     #DP_PF1
    bit     SCIRXST,BIT6        ;8 MSB available ?
    bcnd    navail64,ntc        ;if not repeat the cycle (polling)
    lacc    SCIRXBUF,8          ;load ACC the upper byte
    ldp     #serialtmp
    add     serialtmp           ;add ACC with lower byte
    sacl    Kispeed             ;store it
navail65
    ldp     #DP_PF1
    bit     SCIRXST,BIT6        ;is there any character available (8 LSB)?
    bcnd    navail65,ntc        ;if not repeat the cycle (polling)
    lacc    SCIRXBUF
    and     #0FFh               ;take the 8 LSB
    ldp     #serialtmp
    sacl    serialtmp           ;if yes, get it and store it in serialtmp
navail66
    ldp     #DP_PF1
    bit     SCIRXST,BIT6        ;8 MSB available ?
    bcnd    navail66,ntc        ;if not repeat the cycle (polling)
    lacc    SCIRXBUF,8          ;load ACC the upper byte
    ldp     #serialtmp
    add     serialtmp           ;add ACC with lower byte
    sacl    Kcorspeed           ;store it
    b       menu                ;return to the main polling cycle
```

```
*** END Option

notsix
    b       menu

    .else

    ldp          #n_ref
    splk    #1000h,n_ref

    lacc    stall_timer1        ;cascaded timers to ensure correct stall
                                ;at start
    sub          #1             ;when no Graphic User's Interface is available
    sacl    stall_timer1
    bcnd    norun,GT

    lacc    stall_timer2
    sub          #1
    sacl    stall_timer2
    bcnd    norun,GT

    splk    #01000h,initphase
    splk    #0,stall_timer1
    splk    #0,stall_timer2

    b  menu
norun
    splk    #0,initphase
    b       menu
    .endif
```

Appendix B. Qbasic Graphic User's Interface

```
REM File name : FOC_PMSM.BAS

OPEN "COM1: 9600,N,8,1,CD0,CS0,DS0,OP0,RS,TB1,RB1" FOR OUTPUT AS #1
PRINT #1, "1"; CHR$(0); CHR$(0); : REM speed reference initialization to 0
PRINT #1, "2"; CHR$(23); CHR$(25); CHR$(41); CHR$(3); : REM dac initialization
PRINT #1, "3"; CHR$(0); CHR$(0); : REM initialization phase to 0

est = 0
speedref = 0
init = 0
VDC = 311
da1 = 33: da2 = 32
da3 = 24: da4 = 25
Ki = .03
Kpi = .6
Kcor = .05
Kispeed = .03
Kpispeed = 6.5
Kcorspeed = .0046

initphase$(0) = "Init"
initphase$(1) = "Run"

Tc = 896: REM PWM period in us
speedpu = 3000: REM base speed
ibase = 2: REM base current
Vbase = 120

DIM daout$(200)
daout$(0) = "ia"
daout$(1) = "ib"
daout$(2) = "ic"
daout$(3) = "Ua"
daout$(4) = "Ub"
daout$(5) = "Uc"
daout$(6) = "seno1"
daout$(7) = "t1"
daout$(8) = "t2"
daout$(9) = "coseno"
daout$(10) = "Va"
daout$(11) = "Vb"
daout$(12) = "Vc"
daout$(13) = "VDC"
daout$(14) = "taon"
daout$(15) = "tbon"
daout$(16) = "tcon"
daout$(17) = "teta"
daout$(18) = "ialfa"
daout$(19) = "ibeta"
daout$(20) = "Valfar"
daout$(21) = "Vbetar"
daout$(22) = "idr"
daout$(23) = "iqr"
daout$(24) = "idS"
daout$(25) = "iqS"
daout$(26) = "Vdr"
daout$(27) = "Vqr"
daout$(28) = "epiq"
daout$(29) = "epid"
daout$(30) = "xiq"
```

```
daout$(31) = "xid"
daout$(32) = "n"
daout$(33) = "n_ref"
daout$(34) = "epispeed"
daout$(35) = "xispeed"
daout$(36) = "X"
daout$(37) = "Y"
daout$(38) = "Z"
daout$(39) = "sector"
daout$(40) = "initphase"
REM daout$(41) = "Vr"
REM daout$(42) = "idrref"
daout$(43) = ""
daout$(44) = ""
daout$(45) = ""


nDA = 10

1 CLS
FOR i = 0 TO nDA
COLOR 11
LOCATE (12 + i), 2: PRINT "("; : PRINT USING "##"; i; : PRINT ") "; daout$(i)
LOCATE (12 + i), 22: PRINT "("; : PRINT USING "##"; i + nDA + 1; : PRINT ") "; daout$(i +
nDA + 1)
LOCATE (12 + i), 42: PRINT "("; : PRINT USING "##"; i + 2 * nDA + 2; : PRINT ") ";
daout$(i + 2 * nDA + 2)
LOCATE (12 + i), 62: PRINT "("; : PRINT USING "##"; i + 3 * nDA + 3; : PRINT ") ";
daout$(i + 3 * nDA + 3)
NEXT i
LOCATE 1, 15
COLOR 12: PRINT " Digital Control of a Permanent Magnet Motor"
PRINT
COLOR 10: PRINT "<1>"; : COLOR 2: PRINT " Speed_reference      ("; speedref; "rpm )"
COLOR 10: PRINT "<2>"; : COLOR 2: PRINT " DAC_Outputs   DAC1: ("; daout$(da1); ")"
LOCATE 4, 35: PRINT "DAC2: ("; daout$(da2); ")"
PRINT "                    DAC3: ("; daout$(da3); ")"
LOCATE 5, 35: PRINT "DAC4: ("; daout$(da4); ")"
COLOR 10: PRINT "<3>"; : COLOR 2: PRINT " Init_phase (0=Init)  ("; initphase$(init); ")"
COLOR 10: PRINT "<4>"; : COLOR 2: PRINT " Vbase                ("; Vbase; "Volts )"
COLOR 10: LOCATE 3, 50: PRINT "   <5>"; : COLOR 2: PRINT " Kpi       ("; Kpi; "pu)"
COLOR 10: LOCATE 4, 50: PRINT "      "; : COLOR 2: PRINT " Ki        ("; Ki; "pu)"
COLOR 10: LOCATE 5, 50: PRINT "      "; : COLOR 2: PRINT " Kcor      ("; Kcor; "pu)"
COLOR 10: LOCATE 6, 50: PRINT "   <6>"; : COLOR 2: PRINT " Kpispeed  ("; Kpispeed; "pu)"
COLOR 10: LOCATE 7, 50: PRINT "      "; : COLOR 2: PRINT " Kispeed   ("; Kispeed; "pu)"
COLOR 10: LOCATE 8, 50: PRINT "      "; : COLOR 2: PRINT " Kcorspeed ("; Kcorspeed; "pu)"

COLOR 10: LOCATE 10, 14: PRINT "Choice : ";
DO

a$ = INKEY$
LOOP UNTIL ((a$ <= "6") AND (a$ >= "1")) OR (a$ = "r") OR (a$ = "R")

SELECT CASE a$
CASE "1"
    REM 4.12 format
    PRINT a$; ") ";
    PRINT "Speed_Reference ("; speedref; "rpm ) : ";
    INPUT speedref$
    IF speedref$ = "" THEN 1
    speedrpu = VAL(speedref$) / speedpu
    IF (speedrpu >= 7.999755859#) THEN speedrpu = 7.999755859#
    IF (speedrpu <= -8) THEN speedrpu = -8
    speedrefpu = CLNG(speedrpu * 4096)
    IF (speedref < 0) THEN speedrefpu = 65536 + speedrefpu
```

```
    PRINT #1, "1"; CHR$(speedrefpu AND 255); CHR$((speedrefpu AND 65280) / 256)
    speedref = speedrpu * speedpu
    GOTO 1
CASE "2"
    REM standard decimal format
    PRINT a$; ") ";
    PRINT "DAC1, DAC2, DAC3 or DAC4 ? ";
2       dach$ = INKEY$
    IF dach$ = "" THEN 2
    IF dach$ = CHR$(13) THEN 1
    IF dach$ = "1" THEN
    PRINT "DAC1 Output ("; da1; ") : ";
    INPUT da$
    IF da$ = "" THEN 1
    da1 = VAL(da$)
    END IF
    IF dach$ = "2" THEN
    PRINT "DAC2 Output ("; da2; ") : ";
    INPUT da$
    IF da$ = "" THEN 1
    da2 = VAL(da$)
    END IF
    IF dach$ = "3" THEN
    PRINT "DAC3 Output ("; da3; ") : ";
    INPUT da$
    IF da$ = "" THEN 1
    da3 = VAL(da$)
    END IF
    IF dach$ = "4" THEN
    PRINT "DAC4 Output ("; da4; ") : ";
    INPUT da$
    IF da$ = "" THEN 1
    da4 = VAL(da$)
    END IF
    PRINT #1, "2"; CHR$(da1 AND 255); CHR$(da2 AND 255); CHR$(da3 AND 255); CHR$(da4 AND
255)
    GOTO 1
CASE "3"
    REM 8.8 format
    est = 0
    IF init = 1 THEN init = 0 ELSE init = 1
    IF (init >= 255.9960938#) THEN init = 255.9960938#
    IF (init < 0) THEN init = 0
    init88 = CLNG(init * 256)
    PRINT #1, "3"; CHR$(init88 AND 255); CHR$((init88 AND 65280) / 256)
    GOTO 1

CASE "4"
    REM 4.12 format
    PRINT a$; ") ";
    PRINT "Vbase ("; Vbase; "Volts ) : ";
    INPUT Vbase$
    IF Vbase$ = "" THEN 1
    IF (Vbase <= 0) THEN 1
    VDCpu = VDC / VAL(Vbase$)
    IF (VDCpu >= 7.999755859#) THEN VDCpu = 7.999755859#
    IF (VDCpu <= -8) THEN VDCpu = -8
    VDCinvTc = Tc / VDCpu
    PRINT #1, "4"; CHR$(VDCinvTc AND 255); CHR$((VDCinvTc AND 65280) / 256)
    Vbase = VDC / VDCpu
    GOTO 1

CASE "5"
    REM 4.12 format
    PRINT a$; ") ";
```

```
    PRINT "Kpi ("; Kpi; " ) : ";
    INPUT Kpi$
    IF Kpi$ = "" THEN 51
    Kpi = VAL(Kpi$)
    IF (Kpi >= 1) THEN Kpi = 1
    IF (Kpi <= -1) THEN Kpi = -1
51
    PRINT "                     Ki ("; Ki; " ) : ";
    INPUT Ki$
    IF Ki$ = "" THEN 52
    Ki = VAL(Ki$)
    IF (Ki >= 1) THEN Ki = 1
    IF (Ki <= -1) THEN Ki = -1
52
    Kpipu = 4096 * Kpi
    Kipu = 4096 * Ki
    Kcor = (Ki / Kpi)
    Kcorpu = 4096 * Kcor
    PRINT #1, "5"; CHR$(Kpipu AND 255); CHR$((Kpipu AND 65280) / 256); CHR$(Kipu AND 255);
CHR$((Kipu AND 65280) / 256); CHR$(Kcorpu AND 255); CHR$((Kcorpu AND 65280) / 256)
    GOTO 1

CASE "6"
    REM 4.12 format
    PRINT a$; ") ";
    PRINT "Kpispeed ("; Kpispeed; " ) : ";
    INPUT Kpispeed$
    IF Kpispeed$ = "" THEN 61
    Kpispeed = VAL(Kpispeed$)
    IF (Kpispeed >= 7.9) THEN Kpispeed = 7.9
    IF (Kpispeed <= 0) THEN Kpispeed = 0
61
    PRINT "                     Kispeed ("; Kispeed; " ) : ";
    INPUT Kispeed$
    IF Kispeed$ = "" THEN 62
    Kispeed = VAL(Kispeed$)
    IF (Kispeed >= 1) THEN Kispeed = 1
    IF (Kispeed <= 0) THEN Kispeed = 0
62
    Kpispeedpu = 4096 * Kpispeed
    Kispeedpu = 4096 * Kispeed
    Kcorspeed = (Kispeed / Kpispeed)
    Kcorspeedpu = 4096 * Kcorspeed
    REM Send "Option" - "LSB" - "MSB"
    PRINT #1, "6"; CHR$(Kpispeedpu AND 255); CHR$((Kpispeedpu AND 65280) / 256);
CHR$(Kispeedpu AND 255); CHR$((Kispeedpu AND 65280) / 256); CHR$(Kcorspeedpu AND 255);
CHR$((Kcorspeedpu AND 65280) / 256)
    GOTO 1

CASE ELSE
    PRINT #1, "1"; CHR$(speedrefpu AND 255); CHR$((speedrefpu AND 65280) / 256)
    PRINT #1, "2"; CHR$(da1 AND 255); CHR$(da2 AND 255); CHR$(da3 AND 255); CHR$(da4 AND
255)
REM    PRINT #1, "3"; CHR$(init88 AND 255); CHR$((init88 AND 65280) / 256)
REM    PRINT #1, "4"; CHR$(VDCinvTc AND 255); CHR$((VDCinvTc AND 65280) / 256)
REM    PRINT #1, "5"; CHR$(Kpipu AND 255); CHR$((Kpipu AND 65280) / 256); CHR$(Kipu AND
255); CHR$((Kipu AND 65280) / 256); CHR$(Kcorpu AND 255); CHR$((Kcorpu AND 65280) / 256)
REM    PRINT #1, "6"; CHR$(Kpispeedpu AND 255); CHR$((Kpispeedpu AND 65280) / 256);
CHR$(Kispeedpu AND 255); CHR$((Kispeedpu AND 65280) / 256); CHR$(Kcorspeedpu AND 255);
CHR$((Kcorspeedpu AND 65280) / 256)
    GOTO 1
END SELECT
CLOSE #1
```

Appendix C. Linker Command File

```
foc_pmsm.obj
-m foc_pmsm.map
-o foc_pmsm.out

MEMORY
{
    PAGE 0:
    FLASH_VEC    : origin =    0h, length =   40h
    FLASH        : origin =  040h, length =  00FC0h

    PAGE 1:
    REGS        : origin =     0h, length =    60h
    BLK_B22     : origin =    60h, length =    20h
    BLK_B0      : origin =   200h, length =   100h
    BLK_B1      : origin =   300h, length =   100h
    EXT_DATA    : origin =  8000h, length =  1000h
}

/*--------------------------------------------------------------------*/
/* SECTIONS ALLOCATION                                                */
/*--------------------------------------------------------------------*/
SECTIONS
{
    vectors      : { } > FLASH_VEC PAGE 0  /* INTERRUPT VECTOR TABLE   */
    .text        : { } > FLASH     PAGE 0   /* CODE                    */
    .stack       : { } > BLK_B22   PAGE 1   /* Data storage on DP 0    */
    .dacval      : { } > BLK_B22   PAGE 1
    .data        : { } > BLK_B0    PAGE 1
    .bss         : { } > BLK_B0    PAGE 1   /* GLOBAL VARS, STACK, HEAP*/
    table        : { } > BLK_B1    PAGE 1
}
```

TI Contact Numbers

INTERNET

TI Semiconductor Home Page
www.ti.com/sc

TI Distributors
www.ti.com/sc/docs/distmenu.htm

PRODUCT INFORMATION CENTERS

Americas
Phone +1(972) 644-5580
Fax +1(972) 480-7800
Email sc-infomaster@ti.com

Europe, Middle East, and Africa
Phone
Deutsch +49-(0) 8161 80 3311
English +44-(0) 1604 66 3399
Español +34-(0) 90 23 54 0 28
Francais +33-(0) 1-30 70 11 64
Italiano +33-(0) 1-30 70 11 67
Fax +44-(0) 1604 66 33 34
Email epic@ti.com

Japan
Phone
International +81-3-3344-5311
Domestic 0120-81-0026
Fax
International +81-3-3344-5317
Domestic 0120-81-0036
Email pic-japan@ti.com

Asia
Phone
International +886-2-23786800
Domestic
Australia 1-800-881-011
TI Number -800-800-1450
China 10810
TI Number -800-800-1450
Hong Kong 800-96-1111
TI Number -800-800-1450
India 000-117
TI Number -800-800-1450
Indonesia 001-801-10
TI Number -800-800-1450
Korea 080-551-2804
Malaysia 1-800-800-011
TI Number -800-800-1450
New Zealand 000-911
TI Number -800-800-1450
Philippines 105-11
TI Number -800-800-1450
Singapore 800-0111-111
TI Number -800-800-1450
Taiwan 080-006800
Thailand 0019-991-1111
TI Number -800-800-1450
Fax 886-2-2378-6808
Email tiasia@ti.com