Application Report
SPRA494

# Implementation of Vector Control for PMSM Using the TMS320F240 DSP

*Michel Platnic* *Digital Signal Processor Solutions*

## Abstract

This document presents a solution for controlling a permanent magnet synchronous motor using the Texas Instruments (TI™) TMS320C24x digital signal processor (DSP). The new TMS320C24x family of DSPs offers a cost-effective design of intelligent controllers for brushless motors that can fulfill enhanced operations consisting of fewer system components, lower system cost, and increased performances. The control method presented relies on the field orientated control (FOC) together with a field-weakening operation. This algorithm maintains efficiency in a wide range of speeds, above nominal speed, and takes into consideration torque changes with transient phases by controlling the flux directly from the rotor coordinates. This report describes a practical solution and corresponding results.

### Contents



### Figures

# Introduction

The Texas Instruments TMS320F240 DSP Controller is suitable for a wide range of motor drives. The TMS320F240 provides a single chip solution by integrating on-chip a high computational power along with all of the peripherals necessary for electrical motor control. The main effect of this combination is the possible implementation of advanced controls such as vector control. High range controls increase system performance, reliability, efficiency, and cost. This application report describes a speed control implemented on a TMS320F240 for a three-phase Permanent Magnet drive with sinewave currents.

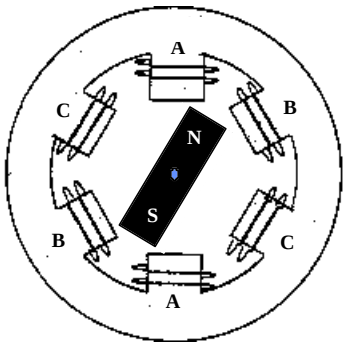
# The AC Permanent Magnet Motor

There are mainly two kinds of three-phase synchronous motors (SM). One uses rotor windings fed from the stator. The other one uses permanent magnets.

A motor fitted out with rotor windings requires brushes to obtain its current supply and generate the rotor flux. The contacts are, in this case, made of rings and do not have any commutator segment. The lifetime of both the brushes and the motor may be similar. The drawbacks of this structure – maintenance needs and lower reliability – are then limited.

Replacing common rotor field windings and pole structure with permanent magnets put the motor into the category of brushless motors. It is possible to build brushless permanent magnet motors with any even number of magnet poles. The use of magnets enables an efficient use of the radial space and replaces the rotor windings, therefore suppressing the rotor copper losses. Advanced magnet materials permit a considerable reduction in motor dimensions while maintaining a very high power density.

*Figure 1. Three-Phase Synchronous Motor with One Permanent Magnet Pair Pole Rotor*




The application studied in this report concerns the permanent magnet motor.

# The Sinewave Currents

Two configurations of permanent magnet brushless motor drives are usually considered, depending on the back-electromagnetic force (BEMF) waveform:

- Sinusoidal type
- Trapezoidal type

Different control strategies (and control hardware) are implemented for each.

The trapezoidal BEMF motor is usually called the DC brushless motor (BLDC). Its appropriate control turns the stator phases on and off using a coarse rotor position. This control is described in the application report, *Implementation of a Speed Controlled Brushless DC Drive Using TMS320F240*, literature number BPRA064.

Sinewave stator currents drive the sinusoidal BEMF drive called the three-phase permanent magnet synchronous motor (PMSM). The stator magnetic field is set in accordance to the rotor field. This application report describes the TMS320F240 DSP Controller together with system considerations that allow high performance to be extracted from this category of motor drives, also called BLDC 3 phases-on.

# Application Description

## Motor Characteristics

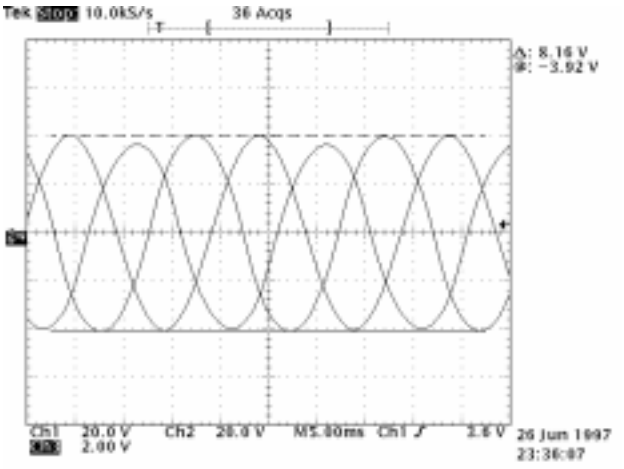
The synchronous machine with permanent magnets described in this application report is a three-phase (Y ) connected motor. The motor includes the following characteristics:

- 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
- Thermal time constant: 30min
- Torque constant: 0.76Nm/A rms
- Voltage constant: 65Vpk/krpm
- Magnet material: NdFeB

The above values are given at 20°C.

The back electromagnetic force has a sinusoidal shape and its stator phases are supplied with sinusoidal currents.

*Figure 2. Three BEMF Waveforms at 1000rpm*

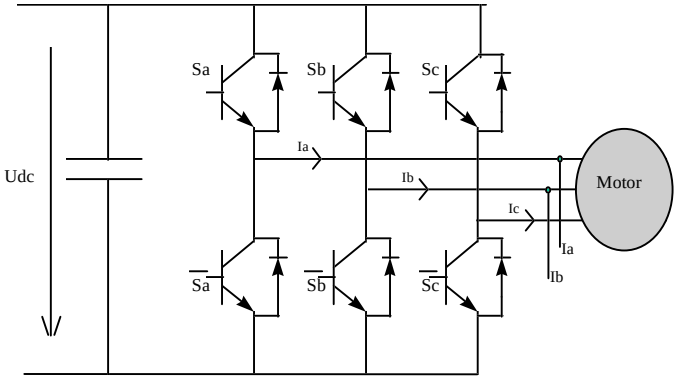



## The Power Electronics Hardware

The ACPM750E used in this application is built around the 750W POWIRTRAIN integrated power stage IRPT1056C from International Rectifier, which includes a rectifier bridge and a three-phase ultra-fast IGBT inverter.

The converter topology supports either sinusoidal currents (three phases ON operation) or direct currents (two phases ON operation). The first control is implemented in this application report. Figure 3 shows the inverter topology used. All of the power device securities are wired (Shutdown, Fault, Clearfault, Itrip, reverse battery diode, varistor peak current protection). The current sensing is insured by 2 L.E.M. directly interfaced with the TMS320F240. The power board also supports the voltage supply of an incremental encoder.

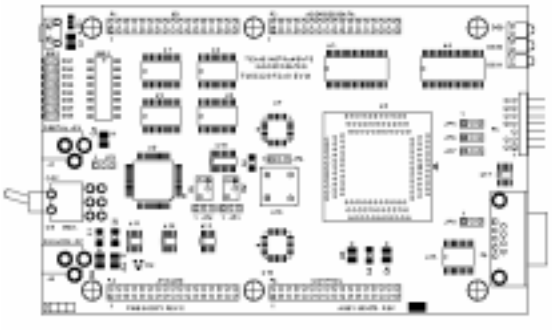
*Figure 3. Inverter Topology*




## The DSP Control Board

The control hardware is the Texas Instruments TMS320F240 Evaluation Module (EVM). It can be directly interfaced to the power electronics board. This evaluation board has a TMS320F240 DSP Controller with an oscillator, JTAG link, RS232 link, and the necessary output connectors. See Figure 4 depicting the EVM board.

*Figure 4. Top View of TMS320F240 EVM Board*



## An Integrated Solution

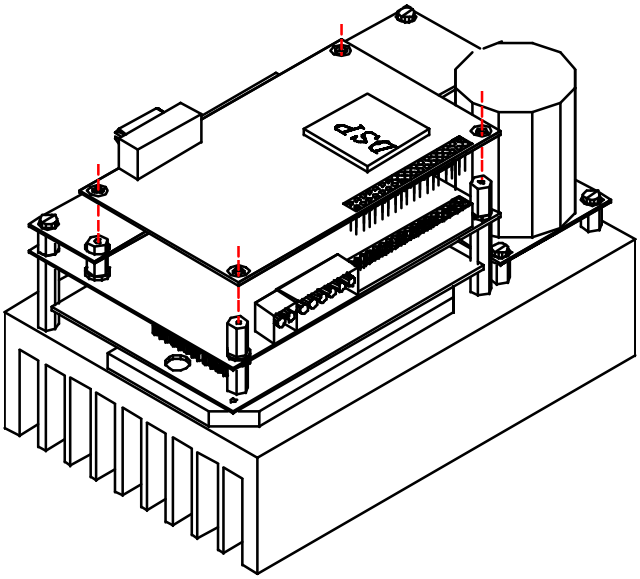
This part describes the features of the ACPM750E and provides an overview of the intelligent AC drive unit, which results when the ACPM750E is connected with Technosoft's Motion Control Kit MCK240 board.

The ACPM750E is a power module for three-phase AC motors, which can directly be controlled with the MCK240 board. Both devices use the universal motion control bus (MC-BUS). They can be connected by simply plugging the MCK240 on top of the ACPM750E.

The ACPM750E offers galvanic isolated feedback signals for two motor currents and the DC bus voltage. Motor speed provided by a tachometer can be measured through an adjustable-gain circuit. Motor position given by an incremental encoder and three Hall sensor signals can also be read through the MCK240.

When the ACPM750E is combined with the MCK240, it results in an intelligent AC drive unit. This unit represents an ideal development platform for design and implementation of high performance control algorithms for three-phase AC motors using the Texas Instruments TMS320F240 ('F240) DSP controller.

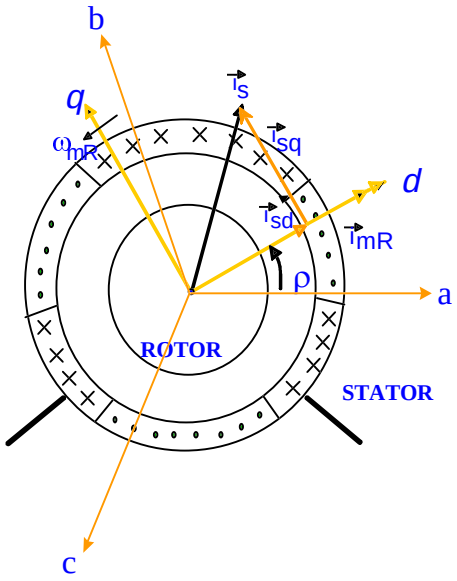
*Figure 5. ACPM750E with a MCK240*




## Field Orientated Control Principle

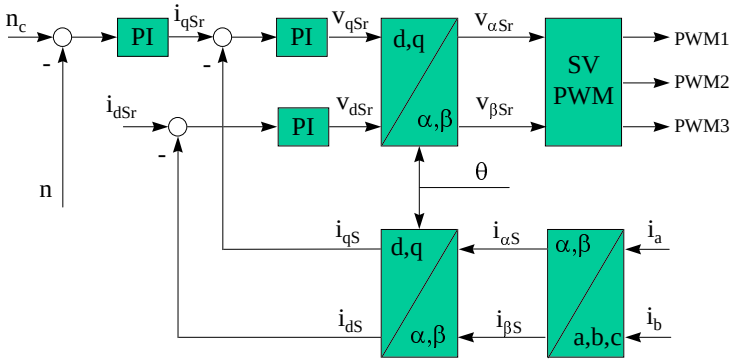
The vector control principle consists of controlling the angle and amplitude components of the stator field. For ease of motor equation representation, the components of the stator current are represented in a rotating reference frame *d,q* aligned with the rotor axis, i.e., with the magnet flux. The motor torque for a permanent magnet machine depends only on the quadrature (*q)* current component (torque component). In this case, the most convenient control strategy is to set to zero the direct (*d* ) current component to minimize the torque vs. current ratio and then increase the motor (and converter) efficiency. The control of current components requires the knowledge of the instantaneous rotor position.

*Figure 6. Stator Current and Magnet Flux Space Vectors in the d,q Rotating Reference Frame and its Relationship with the a, b, c Stationary Reference Frame*




The control scheme proposed for the PM synchronous motor drive is shown in Figure 7. It is based on the vector control principle arranged in the *d,q* rotating frame introduced in the TI application report, *DSP Solution for Permanent Magnet Asynchronous Motor*, literature number BPRA044. Two of three motor phase currents are measured with current sensor, the Clarke transform is applied and then modifies a three-phase system into a two-phase orthogonal system. The output of this transformation is indicated as $i_{\alpha S}$ and $i_{\beta S}$. These two components of the stator current are the input of the Park transform that gives the stator current in the d,q rotating reference frame. Note that this second transformation needs the rotor flux position. The quadrature current component is regulated to the reference value given by the speed controller, while the direct current component is set to zero to minimize the current vs. torque ratio of the motor. The outputs of the current controllers, representing the voltage references, are then impressed to the motor using the Space Vector Modulation technique, once an inverse transformation from the rotating to the fixed stator reference is performed. An outer speed control loop completes the scheme. All of the controllers used are standard PI regulators. Figure 7 shows this basic scheme.

*Figure 7. PMSM Control with Field Orientation*



# Convention

## Software Variables

| | |
|---|---|
| $i_a$, $i_b$, $i_c$ | phase currents |
| $i_{S\alpha}$, $i_{S\beta}$ | stator current ($\alpha$,$\beta$) components |
| $i_d$ | flux component of the stator current |
| $i_q$ | torque component of the stator current |
| $i_{dr}$, $i_{qr}$ | flux and torque command |
| $\theta$ | rotor flux position |
| $v_d$, $v_q$ | (d,q) components of the stator voltage |
| $v_{S\alpha ref}$, $v_{S\beta ref}$ | ($\alpha$,$\beta$) components of the stator voltage (input of the SVPWM) |
| $v_{DC}$ | DC bus voltage |
| $v_{DCinvTc}$ | constant using in the SVPWM |
| $v_a$, $v_b$, $v_c$ | voltage reference used for sector determination in the SVPWM |
| *sector* | sector variable used in SVPWM |
| $t_1$, $t_2$ | time vector application in SVPWM |
| $t_{aon}$, $t_{bon}$, $t_{con}$ | PWM commutation instant |
| *X, Y, Z* | SVPWM variables |
| $n$, $n_{ref}$ | speed and speed reference |
| $I_{idrmin}$, $I_{idrmax}$ | voltage regulator output limitation |
| $I_{smax}$ | phase current limitation |
| $i_{qrmin}$, $i_{qrmax}$ | 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 |
| $K_{iweak}$, $K_{piweak}$, $K_{corweak}$ | field weakening regulator parameters |
| *SPEEDSTEP* | speed loop period |
| *speedstep* | speed loop counter |
| *encincr*, | encoder pulses storing variable |
| *speedtmp* | number of pulses in SPEEDSTEP |

## Base Values

Since the TMS320F240 is a fixed-point DSP, a per unit (p.u.) model is used for the motor variable representations. In this model all quantities are referred to base values. The advantage of this method is that it can be used for any motor (different parameters, power, user requirements, etc.) by simply changing the base values without changing any part of the software.

The base values are determined from the nominal values with the following equations:

$$I_{base} = \sqrt{2} I_n$$
$$V_{base} = \sqrt{2} BEMF$$
$$\omega_{base} = 2\pi f_n$$

Where $I_n$, the BEMF are rms values. The BEMF is measured at nominal speed.

The base RMS values of the motor used in this synchronous drive are as follows:

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

$$V_{base} = 3 * 65 \cong 195V$$

$$\omega_{base} = 2\pi f_n = 2\pi \cdot 50 = 314.15 \frac{rad}{\sec}$$

The quantities in p.u. are defined:

$$i = \frac{I}{I_{base}}$$

$$v = \frac{V}{V_{base}}$$

$$speed = \frac{synchronous\ speed\ (\omega)}{\omega_{base}}$$

## Numerical Consideration

The p.u. model has been developed so that the software representation of speed, currents, and voltages are equal to one when the drive has reached its nominal speed under nominal load. Knowing that during transients, the current might reach higher values than the nominal current ($I_{base}$) to achieve a short response time. Assuming also that the motor speed range might be extended above the nominal speed ($\omega_{base}$), every per unit value may be greater than one. This fact forces the implementation to handle these situations and thus determine the best suited numerical format.

## The Numeric Format

The numeric format used is 4 bit for integer number and 12 bit for fractional number. This numeric format is noted: 4.12 format. The resolution of this format is:

$$\frac{1}{2^{12}} = 0.00024414$$

The correspondence from rated magnitude to 16 bit DSP variable is the following:

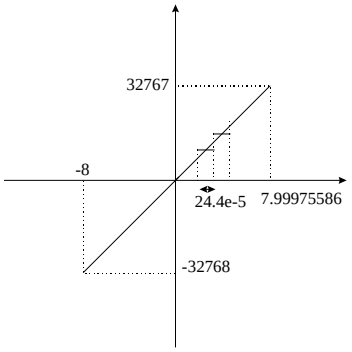
(-8 ; 7.99975586) ⇔ (-32768 ; 32767)

This format has been chosen because the drive control quantities are (most of the time) not greater than four times their nominal values (in other words not greater than four when the per unit model is considered). Otherwise, a different format will be chosen. So using a representation range of [-8;8] ensures that the software values can handle each drive control quantity not only during steady state operation but during transient operation as well.

In this format if the p.u. variable is 1, the correspondent word is 01000h ($2^{12}$=4096). This representation allows for both of the above mentioned variables to be eight times bigger than the base quantities.

With sign extension mode set, the link between the real quantity and its 4.12 representation is given by Figure 8.

*Figure 8. Format Correspondence Diagram*



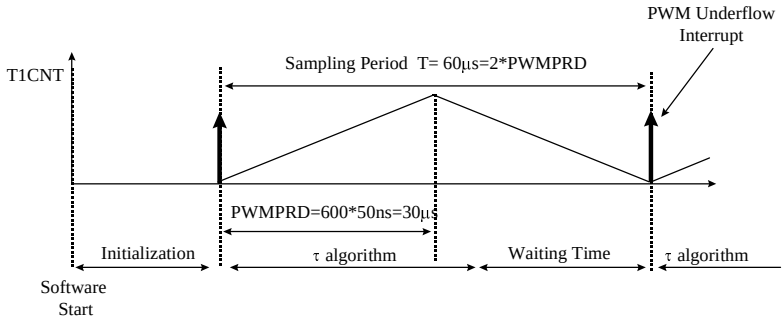

# FOC Software Organization

## Software Synchronization

This software is based on two modules: the initialization and control modules. The first one is performed only once at the beginning. The second module is based on a waiting loop interrupted by the PWM underflow event. When this interrupt flag is set the corresponding Interrupt Service Routine (ISR) is acknowledge and serviced.

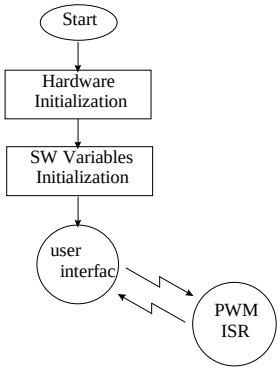
Figure 9 shows the time diagram for the initialization and the operating system.

*Figure 9. FOC Software Initialization and Operating System*



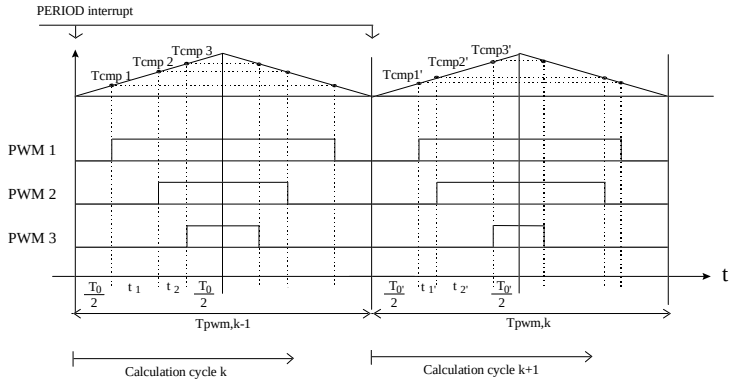

The complete FOC algorithm is computed within the PWM ISR and thus runs at the same frequency as the PWM frequency. The waiting loop could be easily replaced by a human machine interface. The interface software presentation is beyond the scope of this report but is useful to fit the control code and to monitor the control variables. The overview of the software is given in the flow chart below:

*Figure 10. General Software Flowchart*




The DSP Controller Full Compare Unit is handled to generate the necessary pulsed signals to the power electronics board. It is set to generate symmetrical complementary PWM signals at the frequency of 16.6kHz with TIMER1 as time base and with the DEADBAND unit disabled. The sampling period (T) of 60μs can be achieved by setting the timer period T1PER to 600 (PWMPRD=258h).

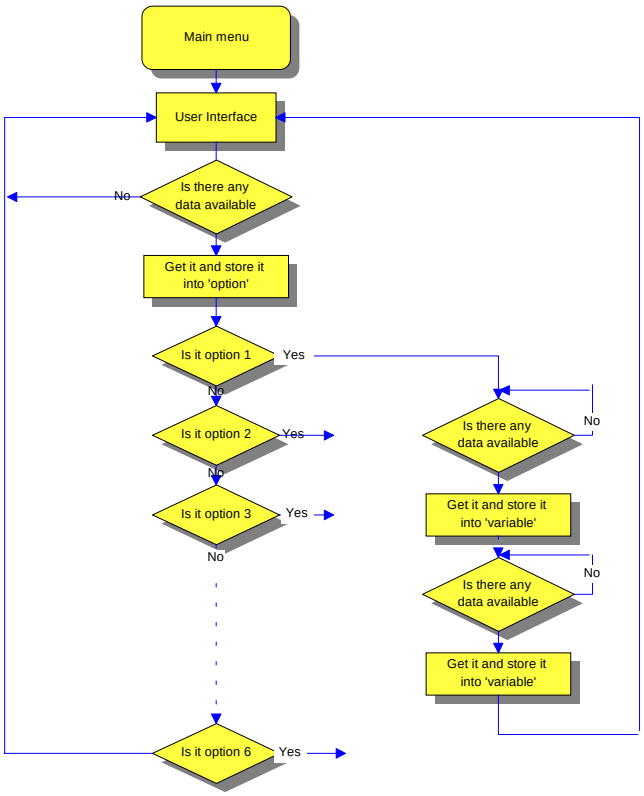
*Figure 11. Control Algorithm Timing*




## Flow Chart

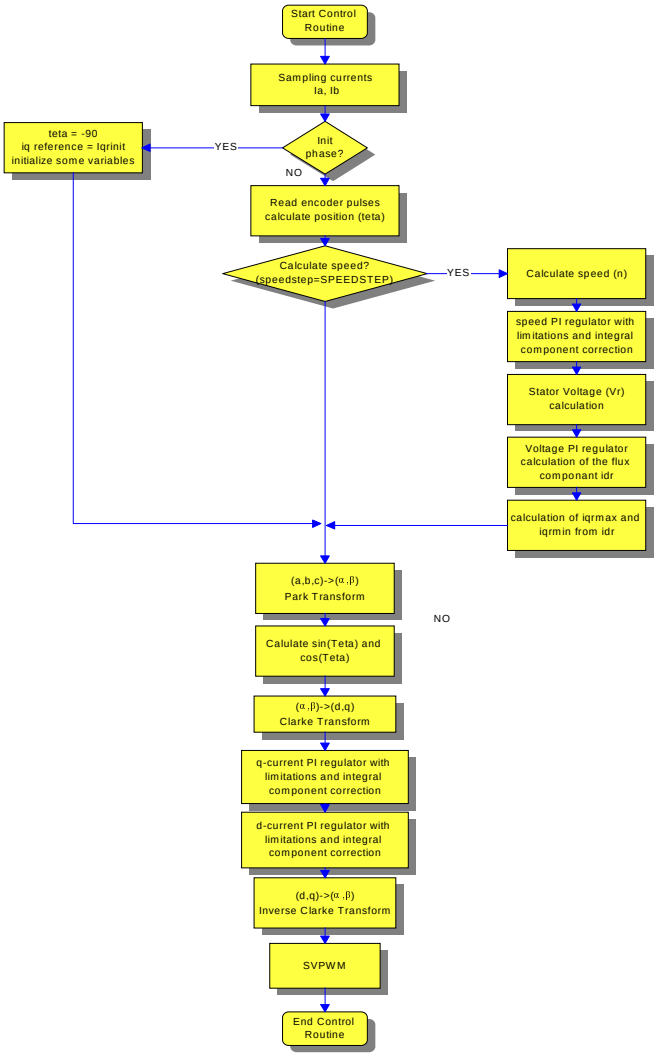
After the F240 features and variable initializations, the software jumps to the waiting loop. Below is presented the user interface. It is interrupted every time an interrupt occurs to start the control. This algorithm is asynchronous from the control and uses the MIPS not used by the control code. It behaves like a waiting loop.

*Figure 12. Waiting Loop/User Interface*



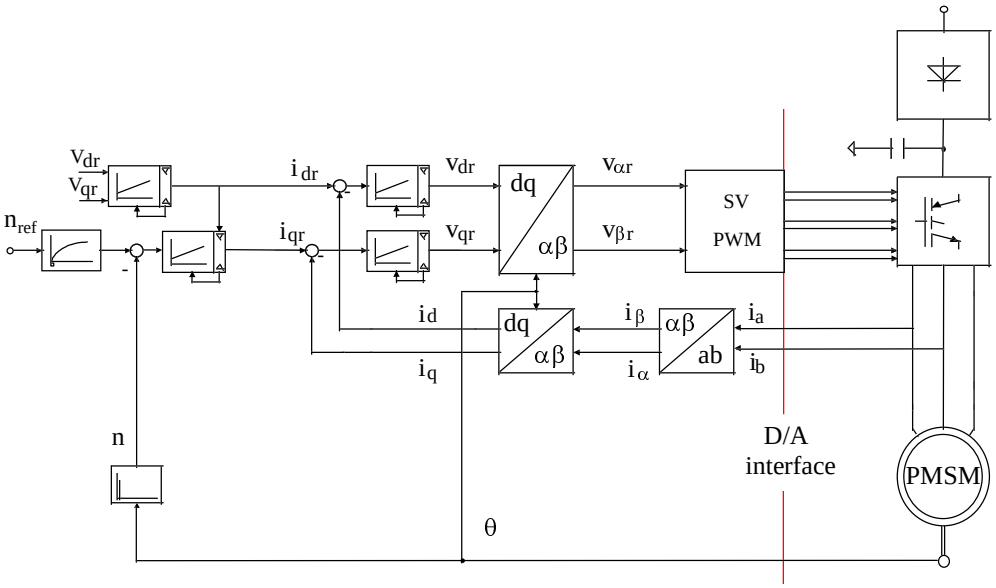

The next block diagram shows the control interrupt service routine. It is executed at the same frequency as the PWM.

*Figure 13. Control Routine Block Diagram*



## Block Diagram

*Figure 14. Block Diagram of the FOC Including Closed Loop Field Weakening Control*




## Parameter Adaptation

This section deals with parameter adaptation depending on the motor specifications. Changes will be done depending on motor poles, rated speed, encoder resolution and current sensor scaling. The proportional integral regulators will be explained for the user to handle the coefficient optimization. All of the parameter adaptation described here needs to be done in the initialization part of the software.

## Motor Poles

Depending on the motor poles, the pulse increments have to be multiplied by Kencoder so that the number of QEP counts is equal to 360° after an electrical period (the angle 360° is equivalent to 1000h). In the general case Kencoder is calculated as follow:

$$Kencoder = N * \frac{1024}{Encoder\_resolution}$$

N is the number of pair poles.

In the case of a three-pair pole motor having a 1024 pulse encoder, set the variable Kencoder to 3 and make a filter as follows:

```
Kencoder        .word   0c000h  ;equivalent to 3.0 in 4.12 format

        lt      encoder         ;multiply encoder pulses by Kencoder to have therotor electrical
                                position
        mpyu    Kencoder        ;encoder pulses = 1365 -> theta = 0fffh = 360 degrees
        pac                     ;encoder pulses = 2731 -> theta = 1fffh = 1*360 degrees
                                ;encoder pulses = 4096 -> theta = 2fffh = 2*360 degrees
        sach    theta,2         ;shift right by 12 for 4.12 format then by 2 to be scaled
                                ;between 0 and 2fffh

        lacl    theta
        and     #0fffh          ;filter to get an angle between 0 and 0fffh
        sacl    theta
```

## Initialization Vector

Idr and iqr are the reference values of id and iq. In permanent state, these values correspond to a current generating a constant flux perpendicular to the magnet's flux and produce a constant torque. In the initialization phase, the stator flux is not rotating, theta must be set to place the rotor aligned with phase A. The rotor will be aligned with iq being different from 0, id angle equals -90°.

```
INITANGLE       .set  fc00h    ;set to -90°
```

At this point the motor should be able to rotate until it reaches a position aligned with phase A. It is then able to generate some torque if forced to another position. The amplitude of the starting current is fixed in amplitude, the value capable of driving the maximum load torque at open loop. Id and iq are respectively equal to zero and Iqrinit.

```
Iqrinit     .set      1000h ;set to nominal current
```

## DC Voltage

The voltage constant is usually found as a motor characteristic Ke in V/rpm

The maximum inverter voltage is equal to VDC 310V

The rated voltage is determined by the BEMF voltage measurement at nominal speed. VDCpu is the variable that sets this magnitude.

VDC is determined as the maximum inverter voltage (ex: 310V) divided by the normalizing factor Vbase, scaled in format 4.12, e.g., VDC = (310/195) $*2^{12}$.

```
VDC . set  196fh
```

VDCinvTc is the invert of VDC multiplied by the PWM period PWMPRD, VDCinvTc = (195/310)*PWMPRD

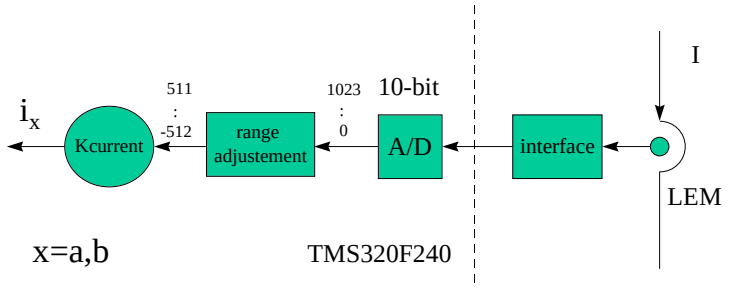
```
PWMPRD  .set  258h   ;equivalent to 60us

VDCinvTc  .set  179h
```

# Current Sensing and Scaling

## Hardware Interface

The FOC structure needs as input two-phase currents. In this application, the two currents are sensed by current-voltage transducers (LEM). The current sensor output needs then to be rearranged and scaled so that they can be used by the control software as 4.12 format values. The complete structure of the current obtained is depicted in Figure 15.

*Figure 15. Current Measurement Chain*




In this application the LEM output signal can be either positive or negative. This signal thus needs to be translated in the range of (0;5V) by the analog interface to allow the single voltage ADC module to read positive and negative values.

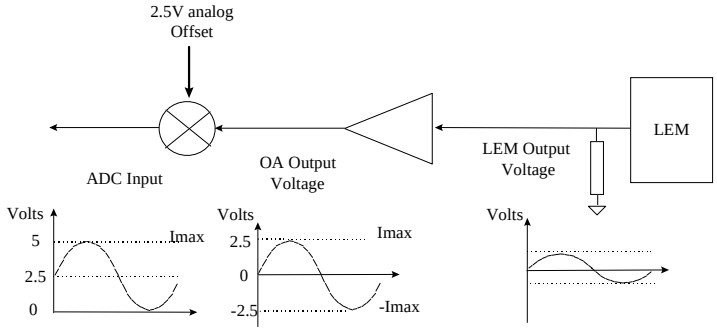
$$(-I_{\max}; I_{\max}) \Leftrightarrow (0;5V)$$

The resolution for the current measurements is:

$$resolution = \frac{2I_{\max}}{2^{10}}$$

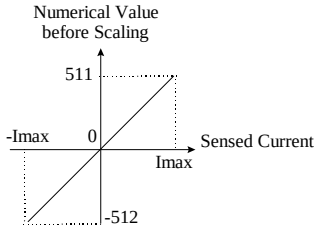
The integrated DSP A/D converters have a 10 bit resolution. The corresponding voltage step is: $\frac{5}{2^{10}} = 4.88 \cdot 10^{-3} V$. Figure 16 shows the different stages of the implemented current sensing:

*Figure 16. Current Sensing Interface Block Diagram*



Note that $I_{max}$ represents the maximum measurable current, which is not necessarily equal to the maximum phase current. The 2.5V analog offset is digitally subtracted from the conversion result, giving thus a signed integer value of the sensed current. The result of this process is represented below:

*Figure 17. Sensed Current Values before Scaling*




## Scaling the Sensed Currents

Like every other quantity in this application, the sensed phase currents must now be expressed with the p.u. model and then be converted into the 4.12 format. Knowing that the p.u. representation of the current is defined as the ratio between the measured current and the base current and that the maximum current handled by the hardware is represented by 512, the p.u. current representation into the 4.12 format is performed by multiplying the sensed current by the following constant:

$$K_{current} = \frac{4096}{(\frac{512 \cdot I_{base}}{I_{\max}})}$$

This constant can be evaluated in one single calculation, not only the p.u. modeling but also the numerical conversion into 4.12 format. When nominal current flows in the motor running at nominal speed the current sensing and scaling block output is 1000h (equivalent to 1pu). You may change the numerical format by simply changing the numerator value. You may adapt this constant to its own current sensing range by simply recalculating $K_{current}$ with its own $I_{max}$ value.

Below is given the dedicated code to scale the current stored in ADCFIFO1. For each current measured, one A/D converter input is used. In this way two currents are converted at the same time (the conversion time for one channel is 6.6μs). In the application, channel 1 for i1 and channel 9 for i2 were selected.

```
*******************************************
* Calculate Ib from LEM measurement *
*******************************************
   ldp  #0e0h
NoConv bit   ADCTRL1,1000b
   bcnd  NoConv,TC ;wait the EOC
   laccADCFIFO1,10
   ldp  #Ib
   sach  ISR_Temp  ;temporary variable
   lacl ISR_Temp
   and   #03ffh
   sub #512     ;the range is [0fe00;01ffh]
   sacl ISR_Temp
              spm   2     ;PM=10,
              lt ISR_Temp
   mpy   Kcurrent     ;Kcurrent is defined like .word
   pac
   sach  Ib    ;Ib f 4.12 (1pu=1000h)
   spm   0     ;PM=00
```

As an example, we consider an input of 0 to 5V representing a current from -10A to 10A. The register range is then:

| Input Voltage | Related Current | ADC_FIFO hexa. Value | ADC_FIFO Binary Value |
|---|---|---|---|
| 0v | -10A | 0000h | 0000 0000 0000 0000b |
| 5v | +10A | FFC0h | 1111 1111 1100 0000b |

All the variables in the software are normalized. The currents are set as 1000h, which represents Ibase (4.1Amps). To give better flexibility, the constant Kcurrent is defined with a format 8.8. This does not influence the format of the currents i1 and i2, which remain in format 4.12.

The scaling factor required is:

Kcurrent=(4096*10)/(512*4.1)*$2^8$=1383h. In format 8.8

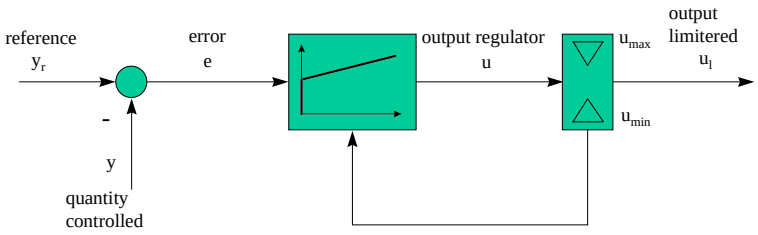
*Kcurrent .set 1383h*

The next step is to verify that the currents sampled i1 and i2 are equal to 0 after scaling. To see this, emulate the software until you step into the label ‘go’. Set a breakpoint after sacl I1, display the memory where I1 is contained or look at the F240 accumulator. If the memory is not exactly 0, it is possible to adjust it directly in the software with the addition of an offset (see comment DC offset in the software).

## Current Regulation

The currents *id* and *iq* are controlled with a PI regulator to match their reference values *idr* and *iqr*. The output *upi* from the regulator is limited between *Vmin* and *Vmax*. This is the anti-windup reset. *Xiq*, the PI integral component, is then adjusted from *upi* limitation.

The following figure illustrates the block scheme and the algorithm:




*INPUT* $y_r, y$

$e=y-y_r$

$u=x_i+K_{pi}e$

$u_l=u$

*IF* $u > u_{max}$ *THEN* $u_l=u_{max}$

*IF* $u < u_{min}$ *THEN* $u_l=u_{min}$

*OUTPUT* $u_l$

$e_l=u-u_l$

$x_i=x_i+K_ie+K_{cor}e_l$

The proportional component is named *Kpi*, the Integral is *Ki* and the PI output limitation error *elpi* is fed back in the *xiq* integral component with a coefficient *Kcor* and the relation *Kcor=Ki/Kpi.*

Below is the PI regulator used to control *iq*. The same scheme is used for *id*.

```
        lacc    iqr             ;epiq = iqr - iq
        sub     iq
        sacl    epiq
        lacc    xiq,12          ;upi = xiq + Kpi*(iqr - iq)
        lt      epiq
        mpy     #Kpi
        apac
        sach    upi,4           ;scale in 4.12
        bit     upi,0           ;test if upi is negative
        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 Vmin
        b       limiterq

upimagzeroq                     ;upi 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
                                ;at this point: Vmin < Vqr < Vmax
limiterq
```

```
sacl    Vqr             ;Save ACC as reference value
sub     upi
sacl    elpi            ;elpi = Vqr - upi
lt      elpi
mpy     #Kcor
pac
lt      epiq
mpy     #Ki
apac
add   xiq,12            ;calculation of the integral coefficient xiq
sach    xiq,4           ;xiq = Kcor*(Vqr - upi) + Ki*(iqr - iq) + xiq
```

If Kcor is set as *Kcor=Ki/Kpi* and upi replaced by *upi = xiq + Kpi*(iqr - iq*), the last equation

*xiq = Kcor*(Vqr - upi) + Ki*(iqr - iq) + xiq*

becomes

*xiq = Kcor*(Vqr - xiq) + xiq*

## Speed Regulation

The speed is given by a 1024 point incremental encoder. The two sensor output channels (A and B) are directly wired to the QEP unit of the TMS320F240 DSP Controller . The QEP assigned timer counts the number of pulses, given by the timer counter register (T3CNT). At each sampling period this value is stored in a variable named *encincr*. As the mechanical time constant is much lower than the electrical one, the speed regulation loop frequency might be lower than the current loop frequency. The speed regulation loop frequency is realized in this application using a software counter. This counter increments on PWM interrupts. Its period is the software variable called *SPEEDSTEP*. The counter variable is named *speedstep*. When *speedstep* is equal to SPEEDSTEP, the number of counted pulses is stored in another variable called *speedtmp* and multiplied by KSPEED to get the motor speed. SPEEDSTEP multiplied by the current cycle time determines the speed cycle time

## Encoder Constant

The TMS320F 240 QEP needs two input signals to count the pulses generated by the encoder. They consist of two pulse sequences with variable frequency shifted of a quarter of a period (90 degrees) apart. The QEP circuit counts both edges of these quadrature-encoded input pulses. Therefore, the frequency of the generated clock to the GP timer is four times the one of each input sequence. For example, if the encoder has 1024 pulses per revolution, the edges counted by the QEP circuit are 4096 for one revolution. This constant is indicated with “Encpulses” and the number of pulses is normalized in the range [0; Encpulses].

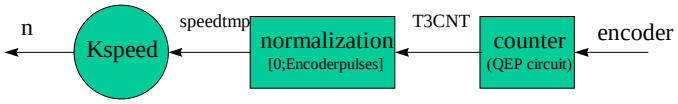
```
Encpulses       .set  4096
```

In the case of an encoder with a number of steps not being a power of 2, the process remains the same.

```
************************************************************
*Read Encoder Pulses and Normalize to [0;EncoderPulses] *
************************************************************
        ldp     #0e8h
        lacc    T3CNT
        ldp     #ISR_Temp
        sacl    ISR_Temp
        sub     encoderold      ;... subtract the previous sampling period value to have
                                ;the increment that we'll accumulate in encoder
        sacl    encincr
        add     encoder
        bcnd    encmagzero,GT,EQ        ;here we start to normalize encoder value to the
                                        ;range [0;Encpulses-1]
        add     #Encpulses      ;the value of encoder could be negative, it depends on
                                ;the rotating direction (depends on motor windings to
                                ;PWM Channels connections)
encmagzero
        sacl    encoder   ;now encoder 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 encoder
        bcnd    encminmax,LT
        sacl    encoder         ;otherwise the value of encoder is greater than
                                ;Encpulses and so we have to store the right value
encminmax               ;ok, now encoder contains the right value in the range
        lacc    tmp             ;[0,Encpulses-1]
                                ;the actual value will be the old one during the next
        sacl    encoderold      ;sampling period
```

## Speed Calculation

*The speedtmp* variable holds the number of pulses in SPEEDSTEP. The following figure explains all of the steps in the speed calculation.




*Kspeed* is the constant that multiplies the encoder increment to calculate the real speed named in the assembly software *speed*. In case the encoder detects only one increment during a speed control time, the motor speed in 4.12 format is equal to:

*Kspeedpu = (60 / (speed_cycle_time*Encpulses*speedpu))*4096*

The encoder constant in 8.8 format is equal to:

*Kspeed=Kspeedpu*256*

To determine the speed with *speedtmp* as the number of encoder steps during a speed cycle time, the code is:

```
lt      speedtmp        ;multiply encoder pulses by Kspeed (8.8 format constant)
mpy     Kspeed          ;to have the value of speed
pac
sfl                     ;shift by 8 to recover from Kspeed being in 8.8 format
sach    speed,7         ;speed is in 4.12 format
```

As an example, to calculate *Kspeed*:

Control cycle time = SPEEDSTEP * PWMPRD = 28*60 us
Encpulses = 4096 incr
speedpu = 3000 rpm

*Kspeed = (60/Encpulses) * (1/*SPEEDSTEP *PWMPRD) * (4096/speedpu) * $2^8$

*Kspeed = (60/4096) * (1/(28*60*$10^{-6}$)) * (4096/ 3000))*256 = 11.9*256*

*Kspeed = 3048 = 0be7h*

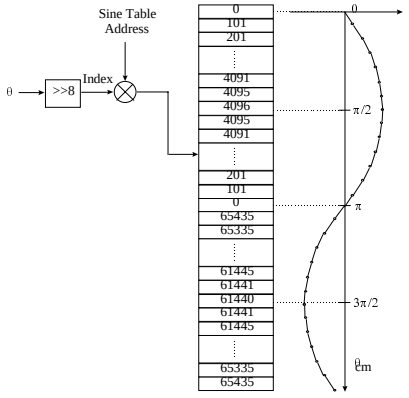
*Kspeed .set 0be7h ;14.28 in 8.8 format*

# Coordinate Transformation

## Generation of Sine and Cosine Values

To generate sine and cosine values a sine look-up table and indirect addressing mode by auxiliary register AR5 have been implemented. As a compromise between the position accuracy and the used memory minimization, this table contains $2^8$=256 words to represent the [0;2π] range. The above computed position (16 bits integer value) thus needs to be shifted 8 positions to the right. This new position (8 bits integer value) is used as a pointer (named *Index*) to access this table. The output of the table is the $\sin\theta$ value represented in 4.12 format. Figure 18 shows *theta,* the *Index* and the sine look-up table.

*Figure 18. Sinθ Calculation Using the Sine Look-Up Table*




Note that to have the cosine value, 256/4=40h must be added to the sine *Index*. The assembly code to address the sine look-up table is given below:

```
*********************************************
* sinTheta, cosTheta calculation
*********************************************
        mar     *,ar5
        lt      theta
        mpyu    SR8BIT
        pac
        sach    Index
        lacl    Index
        and     #0ffh
        add     #sintab
        sacl    tmp
        lar     ar5,tmp
        lacl *
        sacl    sine            ;sine Theta value, 4.12 format

        lacl    Index           ;The same for Cos ...
                ;cos(theta)=sin(theta+90ø)
        add     #40h            ;90ø = 40h elements of the table
        and     #0ffh
        add     #sintab
        sacl    tmp
        lar     ar5,tmp
        lacc*
        saclcosine              ;cosine Theta value, 4.12 format
*********************************************
* END sinTheta, cosTheta calculation
*********************************************
```

## The (a,b,c)->(α,β) Transform (Clarke Transform)

For more details, refer to the TI application report, *Clarke & Park Transforms on the TMS320C2xx*, literature number BPRA048, where the theory is explained. The format used is also 4.12 format.

```
  ***************************************
  * Clarke transform
  * (a,b,c)->(alfa,beta)
  * ialfaS=Ia
  * ibetaS=(2*Ib+Ia)/sqrt(3)
  ***************************************
        lacc    Ia
        sacl    ialfaS          ;ialfaS=Ia
        lacc    Ib,1
        add     Ia
        sacl    ISR_Temp
        spm     2               ;shift 4 places after multiplication
        lt      ISR_Temp
        mpy     #SQRT3inv
        pac
        sach    ibetaS          ;ibetaS=(2*Ib+Ia)/sqrt(3)
        spm     0               ;no shift after multiplication
  END Clarke transform *
```

where SQRT3inv is the following constant (.set):

$$SQRT3inv = \frac{1}{\sqrt{3}} = 0.577 \Leftrightarrow 093dh$$

## The (α,β)->(d,q) Transform (Park Transform)

For more details refer to TI application report, *Clarke & Park Transforms on the TMS320C2xx*, literature number BPRA048, where the theory is explained. The format used is also 4.12 format. With the quantity Theta we consider the rotor flux position.

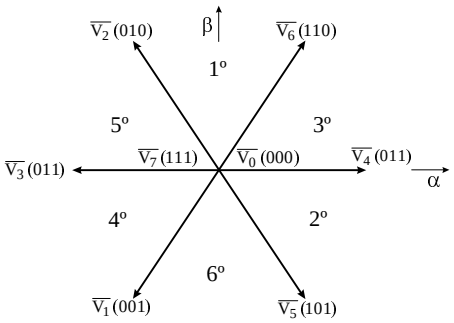
```
*************************************************************
* Park transform
* (alfa, beta)->(d,q)
* id=ialfaS*cos(theta)+ibetaS*sin(theta)
* iq=-ialfaS*sin(theta)+ibetaS*cos(theta)
*************************************************************
        spm     2
        lt      ibetaS
        mpy     sinTheta
        lta     ialfaS
        mpy     cosTheta
        mpya    sinTheta
        sach    idS         ; shift 4 places after multiplication
        lacc    #0
        lt      ibetaS
        mpys    cosTheta
        apac
        sach    iqS
        spm     0
* END Park transform *
```

Inverse matrixes are used to perform the back transformation from currents [id, iq] to [i1, i2, i3].

## Space Vector Modulation

Using a three-leg inverter, eight transistor configurations are possible. These configurations generate eight vectors two of which are ‘zero’ vectors. The remaining vectors divide the plane into six sectors.

*Figure 19. SVPWM, Vectors and Sectors*

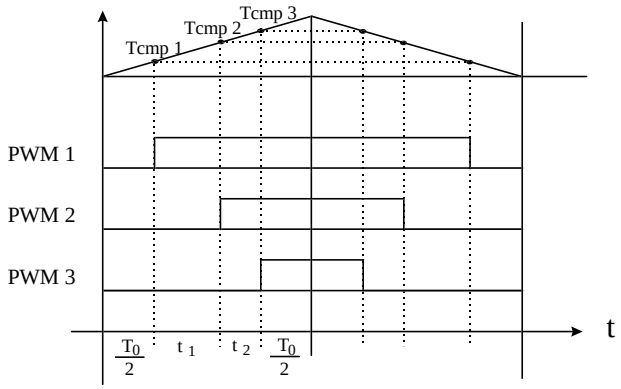



The first step in generating software PWM is to determine the sector where the reference voltage Vref is located. To do this, Vref is transformed into the stator coordinate system Va, Vb and Vc, and the following criteria is applied:

$$IF\ v_a > 0 \quad THEN\ A{:=}1, \quad ELSE\ A{:=}0$$
$$IF\ v_b > 0 \quad THEN\ B{:=}1, \quad ELSE\ B{:=}0$$
$$IF\ v_c > 0 \quad THEN\ C{:=}1, \quad ELSE\ C{:=}0$$
$$\text{sector}{:=}\ A+2B+4C$$

Depending on the sector, two adjacent vectors are chosen. The binary representations of two adjacent basic vectors are different by only one bit, so that only one of the upper transistor switches when the switching pattern switches from one vector to the adjacent one. The two vectors are time weighted by [t1, t2] in a sample period T to produce the desired output voltage[1].

The PWM pattern retained for the output signal is symmetrical PWM where t1 and t2 are centered. T0, the remaining time, is shared equally on each side of t1 and t2 within a half PWM period as shown in the following graph.




The following equations give the method to determinate the right value for the compare register depending on [t1,t2] and the sector.

The first step is to perform saturation control of t1 and t2:

$$IF\ (t_1 + t_2) > PWMPRD\ THEN$$
$$t_{1SAT} = t_1 \frac{PWMPRD}{t_1 + t_2}$$
$$t_{2SAT} = t_2 \frac{PWMPRD}{t_1 + t_2}$$

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

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

[1] More details about space vectors are given in the application note BPRA073.

The last step is to give the right duty cycle (txon) to the right motor phase (in other words to the correct CMPRx) according to the sector. Figure 20 depicts how this is determined.

*Figure 20 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 |

# Field Weakening

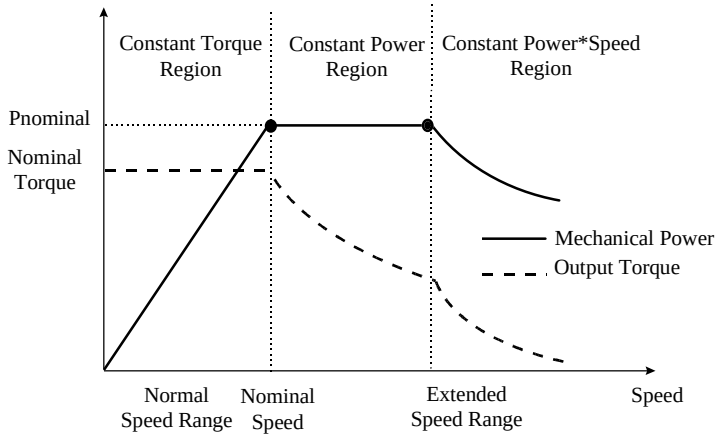
## The Field Weakening

Under certain assumptions it is possible to extend the control speed range beyond the motor's nominal speed. This section explains one possible process to perform this speed range extension.

### Field Weakening Principles

Under nominal load the mechanical power increases as a linear function of speed, up to the nominal power (reached when speed is equal to its nominal value). Knowing that mechanical power is proportional to the torque T times the speed n and that its nominal value has been reached when speed is equal to 3000rpm (nominal value), the torque production must be reduced if the desired speed is to be greater than 3000rpm. This is shown in Figure 21.

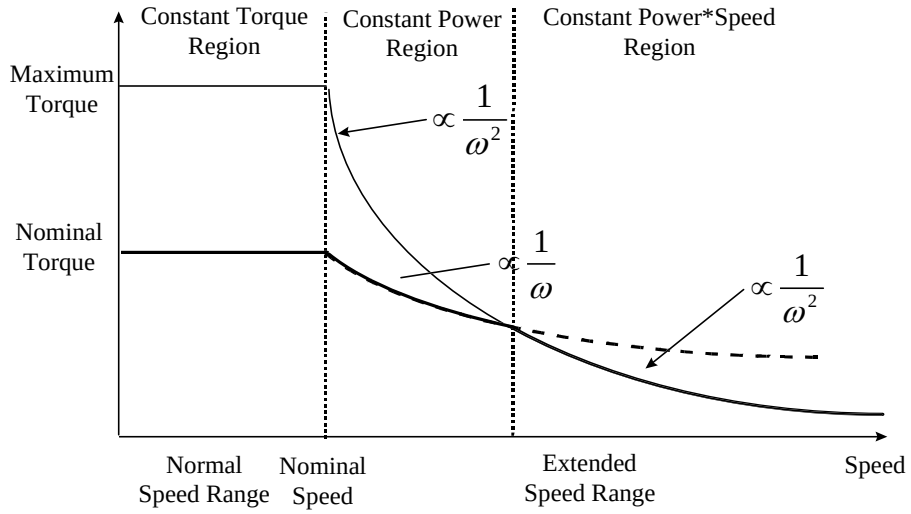
*Figure 21. Field Weakening Real Operation*




Note the three different zones. In the constant power region the nominal torque production behaves like the inverse function of speed, allowing thus constant power production ($P=T\omega$). In the constant Power*Speed region the nominal torque production behaves as the inverse function of the squared speed.

The maximum torque function is equal to a constant in the first region as V (the phase voltage) increases linearly with speed. Above the nominal speed the phase voltage is maintained constant and equal to its nominal value, thus making the maximum torque an inverse function of the squared speed. This results in the curve shown in Figure 22.

*Figure 22. Maximum and Nominal Torque vs Speed*




Note that the nominal torque curve crosses the maximum torque curve. This cross point is the brake point, delimiting the constant power region and the constant power*speed region. Note also that the nominal torque curve crosses the depicted steady state torque curves in the stability zone (making nominal torque to be smaller than the maximum torque) until the brake point. Once this point has been crossed the nominal torque is forced equal to the maximum torque, thus making the power behave as the inverse function of speed.

These characteristics are only related to the motor capabilities. Good control will enable the full speed, torque and efficiency of the drive to be exploited.

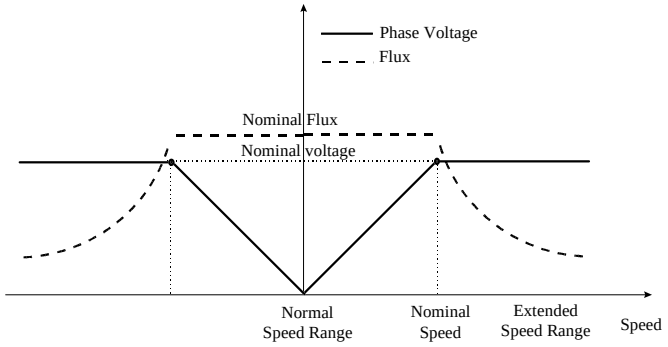
## Field Weakening Constraints

The drive constraints for the extended speed range are first the phase voltages and second the phase currents. Knowing that the phase voltage references increase with speed and as their value can not exceed the nominal value, the flux component must then be reduced down to a value that allows the nominal phase voltage to be maintained and the desired speed to be reached.

Knowing that phase currents increase with load, the maximum resistive torque put on the drive during the extended speed range operation must be set to a value that keeps the phase currents not greater than their nominal value. The maximum resistive torque decreases then as a function of speed.

Both the maximum phase voltage and flux references are given for normal and extended speed range in the following scheme (see Figure 23).

*Figure 23. Field Weakening Voltage Constraints*

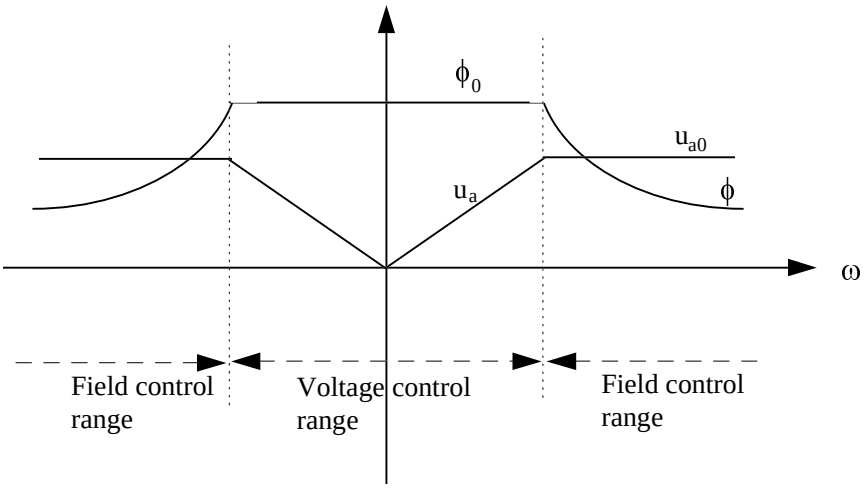



Note that both voltage and current constraints must be respected in steady state operation. In fact, during transient operation the phase current might reach several times its nominal value without any risk for the drive. This only assumes that the resulting drive overheating can be dissipated before performing another transient operation.

## Field Weakening of the Motor at High Speeds with Closed Loop Scheme

The stator current frequency is increased to achieve high speeds. The stator voltage is directly proportional to the motor flux and the angular speed. In normal condition the motor flux is kept constant. It is then obvious that a maximum stator speed is reached with the limit output voltage of the power converter. To reach a higher speed, the flux is reduced as an inverse of the angular speed to keep the stator voltage constant and equal to its maximum.

*Figure 24. Control Range of a PMSM in Steady State*



Practically, if we consider the stator current in the d,q rotating reference frame and its relationship with the a,b stationary reference frame, below the speed where the maximum output voltage is reached, the best choice is $\delta=\pm\pi/2$ and $i_d=0$. The effect of the field weakening can be achieved by advancing the current vector beyond $\delta=\pi/2$, which means introducing a current component in the negative d-axis. As a consequence, $i_q$ and then the torque are reduced so as not to exceed the maximum output current $i_{smax}$:

$$i_s = \sqrt{i_d^2 + i_q^2} \le i_{s\max}$$

Two schemes are possible to implement the field weakening operation. The simplest is the standard **open loop** control for the d axis current reference. Despite the relative simplicity of this realization, it has the following drawbacks:

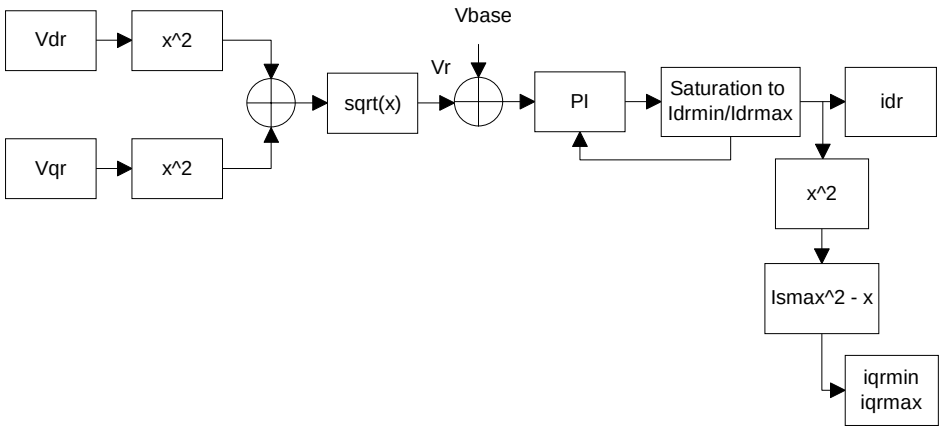
- The reference current equation must be set in the worst-case condition of operation because it corresponds to the lower line voltage and gives a low utilization of the inverter with higher voltages.
- High-speed reliability: to guarantee the correct operation of control at high speeds, it is necessary to reduce further the voltage capability of the inverter.
- The reference current equation depends on the motor electrical characteristics, and it is necessary to consider these parameter variations in the determination.

A **closed loop** control avoids these negative effects. It consists of feeding back a proportional integral (PI) regulator with the motor d and q axis voltages applied to the motor and calculating a new reference for the d-current component. This diagram allows us to exploit the full voltage capability of the inverter independently of the line voltage and the motor characteristics. Idr being determined, the new iqr limitation range set by [iqrmin,iqrmax] is then calculated to not exceed $i_{smax}$, as explained before.

## TMS320F240 Field Weakening Implementation

This section describes in detail the field weakening closed-loop control system. All calculations are performed in real time. No look-up table or approximation of steady-state characteristics is used for this implementation.

*Figure 25. Field Weakening Function Structure*



Vdr and Vqr are used to determine the neutral-phase voltage.

$$Vr = \sqrt{Vdr^2 + Vqr^2}$$

The square root is calculated in real time using the Newton-Raphson method with the recursive equation:

*X(n) = 0.5 * ( X(n-1) + N/X(n-1) )*

```
*********************************************************
* Macro
*********************************************************
isqrt .macro
   lacc   *+          ;lacc Vr, X(0)
   sfr                ;Initial approximate root=N/2
   sacl   *+          ;(AR5+1) = X(0)/2
   splk   #10,*       ;(AR5+2) = 10 (iterations for square root)
isqrt?:               ;X(n) = 0.5 * ( X(n-1) + N/X(n-1) ):
   sbrk   #2
   lacc   *+
   rpt    #15
   subc   *
   and    #0FFFFh
   add    *
   sfr
   sacl   *+

   lacc   *          ;Repeat until iterations completed:
   sub    #1
   sacl   *
   bcnd   isqrt?,NEQ

   mar    *-         ;Restore AR5 & put result in acclow:
   lacc   *
   mar    *-
   .endm
```

Vr and the reference value Vbase are controlled with a PI regulator to generate the output idr. In the field weakening region, |idr| increases. In order not to go over the nominal stator current ismax, iqmax is decreased with the relation:

$$iq\max = \sqrt{is\max^2 - idr^2}$$

The new iqmax calculated serves as limitation for iqr, the result of the speed controller. As iqr is directly proportional to the motor torque, modifying iqmax also changes the torque available on the motor.

# Results

# Software Implementation

The proposed control scheme has been implemented on the TMS320F240 EVM. All of the control routines are implemented using assembler language with fixed-precision numerical representation.

The calculation time of the whole control algorithm is described below:

| | PWM (us) | Control (us) | MIPS |
|---|---|---|---|
| FOC | 60 | 32,6 | 10,9 |
| FOC+speed control | 60 | 35 | 11,7 |
| FOC+field weakening | 60 | 35,6 | 11,9 |
| FOC+speed control+ field weakening | 60 | 38 | 12,7 |

The inverter switching frequency is 16.6 kHz. The timing of the control algorithm is shown in Figure 4. The control algorithm is run by an interrupt (PERIOD interrupt) generated by the event manager PWM unit to synchronize the control and the PWM generating ramp. The speed reference and the main parameters of the control scheme can be changed in real time by means of a host PC linked to the evaluation board via a RS232 link.

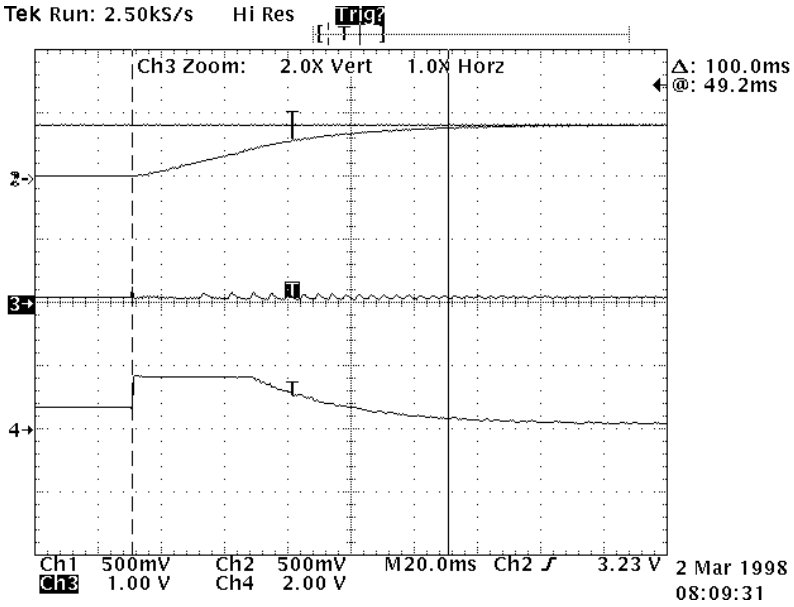
# Experimental Results

This section handles the results of the different drive operations. The motor has been mounted on a test bench with adjustable resistive torque.

At power up, a constant flux is imposed to the drive so that the shaft aligns to a position that is defined as zero angle (index mark). This initialization phase is performed so that a non-absolute encoder may be used as position sensor. This phase does not take into account the direction of shaft at start up. The current amplitude during the initialization phase is the nominal current so that the shaft can be aligned even at nominal torque.

## Results until Nominal Speed

*Figure 26. Speed Transient from 0 rpm to 1000 rpm*




This speed transient picture is given with no torque. The values observed on the oscilloscope are directly derived from control variables through four digital-to-analog converters included in the F240 EVM.

- plot1: n_ref the reference speed
- plot2: n the sensed speed
- plot3: id the magnetizing current
- plot4: iq the current proportional to the torque

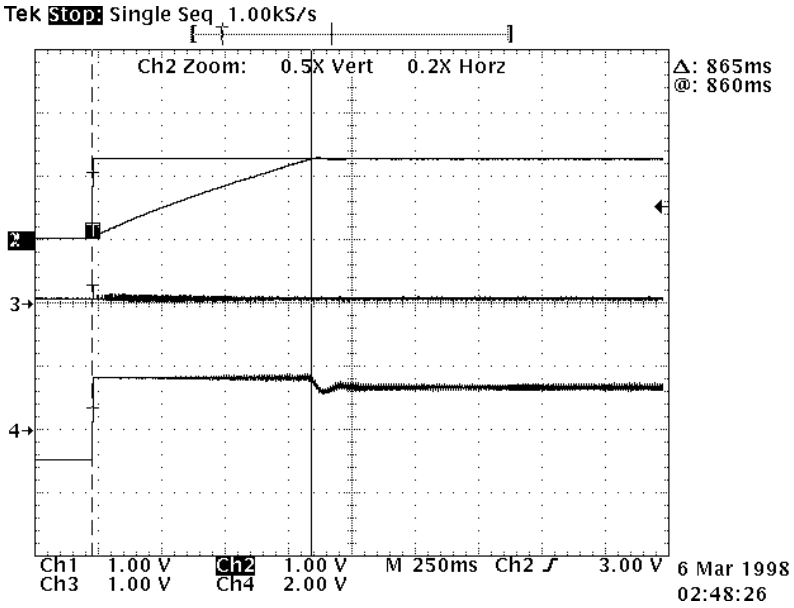
The scaling factor for the variables are 1.25 volts for 1 pu (1000h). In this case:

- 415mV is equivalent to 1000rpm.
- 1.56V is equivalent to 5.12Amps, the maximum current.

The first part of the plot illustrates the PI behavior in steady-state at initialization phase. Id is maintained to 0 and iq to Iqrinit = 1.5Amps, the starting phase current.

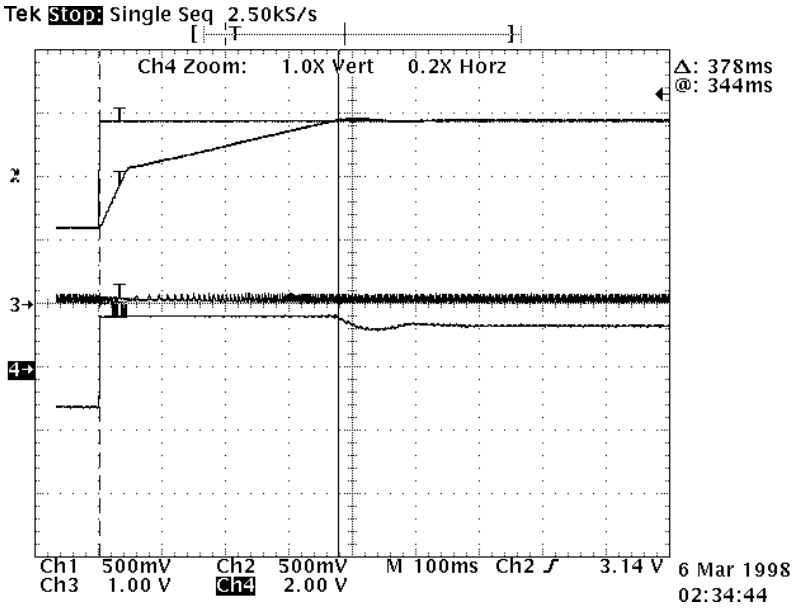
After the speed reference step, iq is set to its maximum value for a better dynamic response, then goes back to its steady-state value related to the low torque implied.

*Figure 27. Speed Transient from 0 rpm to 3000 rpm at Nominal Torque*




This plot is a speed transient from zero speed to nominal speed (3000rpm) at nominal torque (2.2Nm). Id is always kept at zero, iq during transient reaches the maximum current which is 25% higher (5.12Amps) than the nominal current (4.1Amps). In steady state iq is equal to its nominal current as the conditions are nominal power.

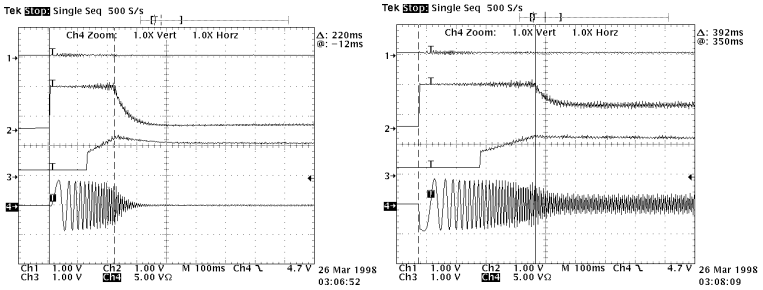
*Figure 28. Speed Transient from -1000 rpm to 1000 rpm at Nominal Torque*




The speed transient from -1000rpm to 1000rpm is done in two phases. The first step from -1000rpm to zero speed is helped by the resistive torque and the acceleration phase from 0 to 1000rpm. All the transient times can be increased by increasing the maximum iq value to more than 25% of the base value.

## Field-Weakening Operation

*Figure 29. Speed Transient from 0 rpm to 3000 rpm Graph1 without Torque, Graph2 with 1.1Nm*



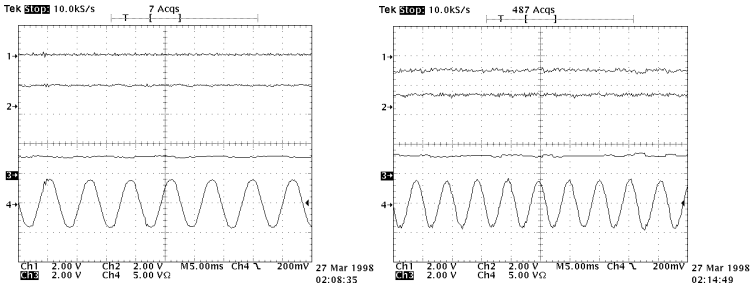

Plot1 is id. It remains constant even during transients, as the voltage Vr (plot3) does not exceed its maximum value equal to Vbase, the BEMF at nominal speed.

Iq (plot2) increases to its maximum value iqrmax and goes back to a value related to the torque imposed when the speed reference is reached.

The break observed in Vr comes from the fact that this variable is only calculated for voltage above Vbase/2.

The last plot is the phase current. It illustrates the control performance never losing the motor position.

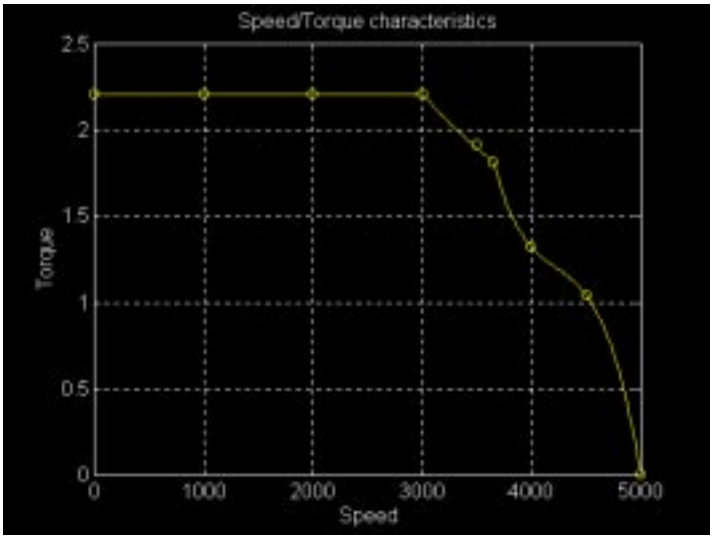
*Figure 30. Steady-State Speed at Nominal Speed and 33% Above at Maximal Torque*




The first graph is done at nominal conditions 3000rpm and 2.2Nm. Id is equal to zero and iq to its maximum value iqrmax. Vr is equal to Vbase.

The second graph is done at 4000rpm with 1.3Nm. Id is controlled negative. 2.2Nm can't be achieved, as iqr is limited to iqrmax, which is updated so that the stator current $i_s = \sqrt{i_d^2 + i_q^2}$ remains constant and equal to its maximum ismax during field weakening operation.

*Figure 31. Speed/Torque Plot*



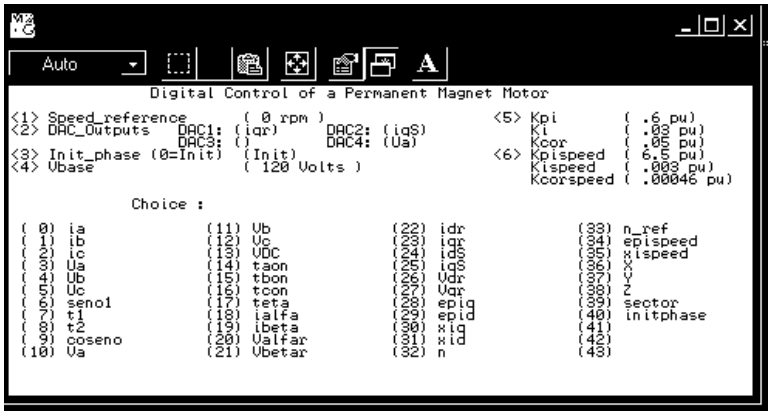

The maximum speed reached is 60% above the nominal speed. The table below gives the results in term of torque and efficiency achieved depending on the speed.

| Speed (rpm) | Torque (Nm) | Efficiency (%) | Power (W) |
|---|---|---|---|
| 1000 | 2,21 | 74 | 690 |
| 2000 | 2,21 | 82 | 690 |
| 3000 | 2,21 | 85 | 690 |
| 3500 | 1,91 | 85 | 690 |
| 3650 | 1,81 | 85 | 690 |
| 4000 | 1,31 | 80 | 540 |
| 4500 | 1,04 | 78 | 490 |
| 5000 | 0 | 0 | 0 |

# User Interface

The user interface is implemented to help the programmer while optimizing some of the control parameters. It allows changing the speed, or selection of any of the control variables to be visualized through any of the four DAC outputs, to run and stop the motor or to change in real-time the PI parameters. Many other options may be implemented. The main program is written in Qbasic. Below is the appearance of the User screen:

*Figure 32. User Screen*




The Qbasic code is given in the appendix. Another part is needed to perform the data exchange from the DSP serial communication interface to the PC. The code is found at the end of the assembly code.

# Conclusion

This document presented a field orientated control scheme for a three-phase permanent magnet drive based on the Texas Instruments TMS320F240 DSP Controller. It has been shown how the real-time processing capabilities of this DSP controller can lead to a highly reliable and effective drive. Not only is drive reliability and efficiency improved, but so is the motor and drive cost effectiveness.

This document also described the speed variation capability, direct torque and flux control, and excellent dynamic behavior. This level of performance has been reached by utilizing only 10.6 MIPS from the 20 MIPS available with the code size not exceeding 1.1Kword of program memory with 300 words of data memory.

# References

Texas Instruments, *Field Orientated Control of Three-Phase AC-Motors*, Literature number: BPRA073, December 1997.

Texas Instruments, *DSP Solution for Permanent Magnet Asynchronous Motor*, Literature number: BPRA044, Nov. 1996.

Werner Leonard, *Control of Electrical Drives*, 2nd Completely Revised and Enlarged Edition, Springer, ISBN 3-540-59380-2.

Texas Instruments, *Current Measurement Using a Single Shunt Resistor*, Literature number: BPRA077, December 1997.

T.J.E. Miller, *Brushless Permanent-Magnet and Reluctance Motor Drives*, Oxford Science Publications, ISBN 0-19-859369-4.

Texas Instruments, *Implementation of a Speed Controlled Brushless DC Drive Using TMS320F240*, Literature number BPRA064, July 1997,

Texas Instruments, *Clarke & Park Transforms on the TMS320C2xx*, literature number BPRA048, June 1998,

# Appendix A. TMS320F240 FOC Software

```
;*****************************************************************
;*                    TEXAS INSTRUMENTS                          *
;*             Field Orientated Control for PMSM                 *
;*                    with field weakening                       *
;*****************************************************************
;*   File Name:  focmck7.asm                                     *
;*   Originator: Michel Platnic                                  *
;*   Description:The software includes                           *
;*                   -PMSM field oriented control                *
;*                   -2 phase current measurement                *
;*                   -PDPINT routine for fault                   *
;*                   -Current closed-loop initialization phase   *
;*                   -Vr BEMF real-time calculation              *
;*                   -iqr real-time limit calculation / idr      *
;*                   -idr PI calculation                         *
;*                   -User Interface                             *
;*                                                               *
;*   Function list: -c_int0                                      *
;*                                                               *
;*   Target:     TMS320F240, MCK240                              *
;*               Code can be Flashed                             *
;*               ACPM750E Power board                            *
;*               Digiplan Motor MD3450                           *
;*                                                               *
;*   status:     Working                                         *
;*                                                               *
;*   History:    Completed on 26 March 98                        *
;*****************************************************************

                .include ".\c240app.h"
                .mmregs

;*****************************************************************
* Start
;*****************************************************************
                .globl  _c_int0 ;set _c_int0 as global symbol

                .sect "vectors"
                b       _c_int0 ;reset interrupt handler
_c_int1         b       _c_int1 ;RTI,SPI,SCI,Xint interrupt handler
```

```
                b       _c_int2 ;PWM interrupt handler
_c_int3         b       _c_int3 ;
_c_int4         b       _c_int4 ;
_c_int5         b       _c_int5 ;
_c_int6         b       _c_int6 ;capture/ encoder Interrupts
                .space 16*6     ;reserve 6 words in interrupt table


*************************************************************
* Auxiliary Register used
* ar4    pointer for context save stack
* ar5    used in the interruption c_int2 for control calculation
* ar6    for main program
*************************************************************
stack           .usect "blockb2",15    ;space for Status Register context save in Page 0
dac_val         .usect "blockb2",5     ;space for dac values in Page 0

*** Motor Digiplan ***
*** Numeric formats: all 4.12 fixed point format twos complement for negative values
*** (4 integer & sign + 12 fractional) except otherwise specified
* - Nominal current 4.1 Amps max= 2.2/0.76 *sqrt(2) (max value)
* - Nominal Torque 2.2 Nm
* - Rated Power 1150W
* - Currents: 1000h (4.12)= 4.1 A = Ibase (max value)
* - Voltages: 1000h (4.12)= 325 V = Vbase (max value) phase-neutral
* - Angles  : [0;ffffh]  = [0;360] degrees
* - Speed   : [0;1000h] (4.12) = [0;3000] rpm
*** END Numeric formats

*************************************************************
* Look-up tables .includes
* N.B. all tables include 256 elements
*************************************************************
sintab          .usect "table",256  ;space to copy sine table in RAM

                .sect "table_f"     ;sine table in ROM
sintab_flash    .include        sine.tab
            ;sine wave look-up table for sine and cosine waves generation
            ;generated by the BASIC program "SINTAB.BAS"
            ;4.12 format

*** END look-up tables .includes

*************************************************************
* Variables and constants initializations
* To program the flash array all variable are set to bss
* A 'V' is added in front of the variable initialized later
*************************************************************
                .data

*** current sampling constants
VKcurrent       .set    00b38h  ;8.8 format (*11.22) sampled currents normalization
constant
                                ;ADCIN6 (i1 current sampling)
                                ;ADCIN14 (i2 current sampling)
                                ;+/- 5.75 Amps for Ibase = 4.1 Amps
*** axis transformation constants
VSQRT3inv       .set    093dh   ;1/SQRT(3) 4.12 format
VSQRT32         .set    0ddbh   ;SQRT(3)/2 4.12 format

*** PWM modulation constants
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
VKi             .set    07Ah    ;4.12 format = 0.03
```

```
VKpi            .set    999h    ;4.12 format = 0.60 (include period)
VKcor           .set    0cch    ;4.12 format = 0.05
                                ;Kcor = Ki/Kpi
*** PI speed regulators parameters
VKispeed        .set    7ah     ;4.12 format = 0.03
VKpispeed       .set    06800h  ;4.12 format = 6.5
VKcorspeed      .set    12h     ;4.12 format = 0.0046

*** PI field-weakening regulators parameters
VKiweak         .set    07Ah    ;4.12 format = 0.03
VKpiweak        .set    999h    ;4.12 format = 0.60 (include period)
VKcorweak       .set    0cch    ;4.12 format = 0.05

*** 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
Vismax          .set    0bb5h   ;4.12 format = 3Amps Limitation for
                                ;ACPM750E
Idrmin          .set    0f44bh  ;4.12 format = -3Amps limit. for ACPM750E
Idrmax          .set    00000h  ;4.12 format =  0A   (1000h = Ibase)

*** Initialization phase Iqr
Iqrinit         .set    009c1h  ;4.12 format = 2.5A (1000h = Ibase)

*** Encoder variables and constants
VKencoder       .set    0c000h  ;this constant is needed only with
                    encoder !
                ;it is used to convert encoder pulses [0;4095] to an
                    electric angle [0;360]=[0000h;1000h]
                ;2.14 format unsigned = 3.0 (see "Theta calculation" block
                    for details)
Encpulses       .set    4096    ;this constant is needed only with encoder !
                ;number of encoder pulses per round (mechanical)

*** Speed and estimated speed calculation constants
Nbase           .set    1000h   ;Base speed
Kspeed          .set    0be7h   ;this constant is needed only with encoder !
                ;it is 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 samplig period = current sampling period * 10

*** 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    i1,1            ;phase current i1
    .bss    i2,1            ;phase current i2
    .bss    i3,1            ;phase current i3
    .bss    Ua,1            ;Phase 1 voltage
    .bss    Ub,1            ;Phase 2 voltage
    .bss    Uc,1            ;Phase 3 voltage
    .bss    seno1,1         ;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    coseno,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    theta,1          ;rotor electrical position in the range [0;1000h]
                            ;4.12 format = [0;360] degrees
    .bss    ialfa,1         ;alfa-axis current
    .bss    ibeta,1         ;beta-axis current
    .bss    Valfar,1        ;alfa-axis reference voltage
    .bss    Vbetar,1        ;beta-axis reference voltage
    .bss    idr,1           ;d-axis reference current
    .bss    iqr,1           ;q-axis reference current
    .bss    id,1            ;d-axis current
    .bss    iq,1            ;q-axis current
    .bss    Vdr,1           ;d-axis reference voltage
    .bss    Vqr,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    encoder,1
    .bss    Vr,1            ;Phase max voltage
    .bss    iqrmin,1        ;iqr min limitation
    .bss    iqrmax,1        ;iqr 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    vDCinvTc,1      ;VDCinv*(Tc/2) (used in SVPWM)
    .bss    epvr,1          ;PI error for field weakening
    .bss    xvr,1           ;PI integral term for field weakening
    .bss    indice1,1       ;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    faultreset,1    ;Used to re-enable the hardware protection
    .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
    .bss    Kcurrent,1      ;Cf explanation given above
    .bss    SQRT3inv,1      ;Cf explanation given above
    .bss    SQRT32,1        ;Cf explanation given above
    .bss    Ki,1            ;Cf explanation given above
    .bss    Kpi,1           ;Cf explanation given above
    .bss    Kcor,1          ;Cf explanation given above
    .bss    Kispeed,1       ;Cf explanation given above
    .bss    Kpispeed,1      ;Cf explanation given above
    .bss    Kcorspeed,1     ;Cf explanation given above
    .bss    Kiweak,1        ;Cf explanation given above
    .bss    Kpiweak,1       ;Cf explanation given above
    .bss    Kcorweak,1      ;Cf explanation given above
```

```
    .bss    ismax,1         ;Cf explanation given above
    .bss    Kencoder,1      ;Cf explanation given above


*** END Variables and constants initializations


    .text                   ;link in "text section

*******************************************************
* Macro
* Inputs: argument in *AR5 !MUST BE POSITIVE!
* Outputs:result in ACClow
* Notes:  this function uses the Newton-Raphson method:
*         X(n) = 0.5 * ( X(n-1) + N/X(n-1) )
* this function uses *(AR5+1) & *(AR5+2) locations!
*******************************************************
isqrt .macro
    clrc sxm
    lacc    *+              ;lacc Vr, X(0)
    sfr                     ;Initial approximate root = N/2
    sacl    *+              ;(AR5+1) = X(0)/2
    splk    #10,*           ;(AR5+2) = 10 (iterations for square root)
isqrt?:                     ;X(n) = 0.5 * ( X(n-1) + N/X(n-1) ):
    sbrk    #2
    lacc    *+              ;
    rpt     #15
    subc    *
    and     #0FFFFh
    add     *
    sfr
    sacl    *+

    lacc    *               ;Repeat until iterations completed:
    sub     #1
    sacl    *
    bcnd    isqrt?,NEQ

    mar     *-              ;Restore AR5 & put result in acclow:
    lacc    *
    mar     *-
    setc sxm
    .endm

*******************************************************
* _c_int2 Interrupt Service Routine
* synchronization of the control algorithm with the PWM
* underflow interrupt
*******************************************************
_c_int2:
************************
* Context Saving
************************
    larp    ar4             ;context save
    mar     *-
    sst     #1,*-           ;status register 1
    sst     #0,*-           ;status register 0
    sach    *-              ;Accu. low saved for context save
    sacl    *-              ;Accu. high saved
* END Context Saving *

    mar     *,ar5           ;used later for DACs output
    ldp     #DP_EV
    lacc    IVRA
    ldp     #tmp
    sacl    tmp
    sub     #20H
    bcnd    PDPRoutine,EQ
    sub     #9
```

```
    bcnd    ControlRoutine,EQ
    b       ContextRestoreReturn
* END Int2 Interrupt Service Routine *

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 *


PDPRoutine
    ldp     #IFRA>>7
    splk    #001h,IFRA  ;Clear all flags, may be change
                        ;with only T1 underflow int.
    ldp     #DP_EV
    splk    #0fffh,ACTR
    splk    #0207h,COMCON   ;FIRST enable PWM operation
    splk    #8207h,COMCON   ;THEN enable Compare operation
    ldp     #DP_PF2
    splk    #0FF00h,PBDATDIR;IOPB 1 conf. as output, set to logic 0
                            ;to re-enable protection circuitry
                            ;after a fault on ACPM750E
    rpt     #200            ;wait minimum 2us (needed by circuitry)
    nop                     ;here 10us
    splk    #0FF02h,PBDATDIR;IOPB 1 conf. as output, set to logic 1
    b       ContextRestoreReturn
* END PDPRoutine

ControlRoutine
****************************************************************
* Current sampling - AD conversions
* N.B. we will have to take only 10 bit (LSB)
****************************************************************
    ldp     #DP_PF2
    splk    #0FF08h,PCDATDIR;bit IOPC7 set to 0 for test purposes

    ldp     #DP_PF1
    splk    #186Dh,ADC_CNTL1;i2 and i3 conversion start
                            ;ADCIN6 selected for i2 A/D1 110
                            ;ADCIN14 selected for i3 A/D2 110
                            ;01101101
*** current sampling
conversion
    bit     ADC_CNTL1,8
    bcnd    conversion,tc   ;wait approximatly 6.6us
    lacc    ADC_FIFO1,10
    ldp     #i2
    sach    i2
    ldp     #DP_PF1
    lacc    ADC_FIFO2,10
    ldp     #i3
    sach    i3

*** fault enable and test EVM LED On/Off
    lacc    faultreset
    bcnd    initcontrol,EQ
    ldp     #DP_EV
    splk    #0999h,ACTR
    ldp     #faultreset
    splk    #0,faultreset
```

```
*** Initialization phase
initcontrol
    lacl    initphase       ;are we in initialization phase ?
    bcnd    noinit1,NEQ
    setc    xf
    lacc    #0fc00h         ;if yes, set theta = 0fc00h 4.12 format = -90 degrees
                            ;(align rotor with phase 1 flux)
    sacl    theta            ;
    lacc    #Iqrinit        ;q-axis reference current = initialization q-axis
                            ;reference current
    sacl    iqr             ;
    lacc    #0              ;zero some variables and flags
    sacl    idr             ;
    sacl    encoder         ;
    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    #1,T3CNT        ;zero Incremental Encoder value if initialization step
    ldp     #initphase
    b       go              ;there is no need to do position and speed calculation
                            ;in initialization phase (the rotor is locked)
*** End Initialization phase

noinit1
*** Encoder pulses reading
    clrc    xf
    ldp     #DP_EV
    lacc    T3CNT           ;we read the encoder pulses and ...
    neg                     ;encoder plug in the opposite direction for ACPM
    ldp     #i1
    sacl    tmp
    sub     encoderold      ;subtract the previous sampling period value to have
                            ;the increment that we'll accumulate in encoder
    sacl    encincr         ;
    add     encoder         ;
    bcnd    encmagzero,GT,EQ;here we start to normalize encoder value to the
                            ;range [0;Encpulses-1]
    add     #Encpulses      ;the value of encoder could be negative, it depends on
                            ;the rotating direction (depends on motor windings to
                            ;PWM Channels connections)
encmagzero
    sacl    encoder         ;now encoder 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 encoder
    bcnd    encminmax,LT    ;
    sacl    encoder         ;otherwise the value of encoder is greater than
                            ;Encpulses and so we have to store the right value
encminmax                   ;ok, now encoder contains the right value in the range
    lacc    tmp             ;[0,Encpulses-1]
                            ;the actual value will be the old one during the next
    sacl    encoderold      ;sampling period
*** END Encoder pulses reading

******************
* Theta calculation
******************
    lt      encoder         ;multiply encoder pulses by Kencoder (4.12 format
                            ;constant) to have the rotor electrical position
    mpyu    Kencoder        ;encoder pulses = 0    -> theta = 0fffh = 0 degrees
    pac                     ;encoder pulses = 1600 -> theta = 1fffh = 1*360 degrees
                            ;encoder pulses = 3200 -> theta = 2fffh = 2*360 degrees
    sach    theta,2
```

```
    lacl    theta
    and     #0fffh
    sacl    theta
*** END theta calculation

*****************************************************
* 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
*** END Speed calculation from encoder pulses

*****************************************************
* 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    iqrmin
    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    iqrmin          ;set acc to -ve saturated value
    b       limiters

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

limiters
    sacl    iqr             ;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
*** END Speed regulator with integral component correction

*************************************************
* Field-weakening algorithm with PI regulator
* Calculation of sqrt(Vdr^2 + Vqr^2)
* only if n>Nbase/2
*************************************************
    lacc    #Nbase
    sfr
    sub     n
    bcnd    nocalc,GEQ      ;calculate field-weakening if n>Nbase/2

    lar     ar5,#60h
    zac
    mpy     #0
    mar     *,ar5
    spm     2               ;4.12 multiplication format
    sqra    Vdr
    sqra    Vqr
    apac
    sach    *
    isqrt                   ;calculate the square root
    spm     0

    sacl    Vr,6

************************************************************
* Voltage regulator with integral component correction
* (Vbase,Vr)->(idr)
************************************************************
    lacc    #Vbase
    sub     Vr
    sacl    epvr
    lacc    xvr,12
    lt      epvr
    mpy     Kpi
    apac
    sach    upi,4

    bit     upi,0
    bcnd    upimagzerov,NTC
    lacc    #Idrmin
    sub     upi             ;
    bcnd    neg_satv,GT     ;if upi<Vmin branch to saturate
    lacc    upi             ;value of upi is valid
    b       limiterv
neg_satv
    lacc    #Idrmin         ;set ACC to neg saturation
    b       limiterv

upimagzerov                 ;Value was positive
    lacc    #Idrmax
    sub     upi             ;
    bcnd    pos_satv,LT     ;if upi>Vmax branch to saturate
    lacc    upi             ;value of upi is valid
    b       limiterv
pos_satv
    lacc    #Idrmax         ;set ACC to pos saturation

limiterv
    sacl    idr             ;Always negative
    sub     upi
    sacl    elpi
    lt      elpi
    mpy     Kcor
    pac
```

```
    lt      epvr
    mpy     Ki
    apac
    add     xvr,12
    sach    xvr,4
*** END voltage regulator with integral component correction

************************************************
* Field-weakening algorithm iqr limitation
* for PI regulator
* Calculation of sqrt(ismax^2 - idr^2)
* Output iqrmax
************************************************
    lar     ar5,#60h
    zac
    mpy     #0
    mar     *,ar5
    spm     2               ;4.12 multiplication format
    sqra    idr
    sqrs    ismax           ;substract
    apac
    sach    *
    isqrt                   ;calculate the square root
    spm     0
    sacl    iqrmax,6
    neg
    sacl    iqrmin,6
*** END field weakening routines

************************************************
* 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        ;
*** END Measured speed and reference speed variables updating

go
*****************************************
* Sampled current scaling
* to nominal current 1000h <-> I_nominal
*****************************************
    ldp     #i1
    lacc    i2
    add     #045h           ;then we compensate the DC offset (that should be zero, but
it isn't
    and     #3ffh
    sub     #512            ;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
    neg                     ;needed for ACPM
    sacl    i2              ;sampled current i2, f 4.12

    lacc    i3
    add     #03ch           ;then we compensate the DC offset (that should be
                             zero, but it isn't)
    and     #3ffh
    sub     #512
    sacl    tmp
    lt      tmp
    mpy     Kcurrent
```

```
    pac
    sfr
    sfr
    neg                     ;needed for ACPM
    sacl    i3

    add     i2
    neg
    sacl    i1              ;i1 = -(i3+i2)
    spm     0
*** END current sampling - AD conversions

*******************************************
* Sampled current scaling and
* (a,b,c) -> (alfa,beta) axis transformation
* ialfa = i1
* ibeta = (2 * i2 + i1) / sqrt(3)
*******************************************
    lacc    i1
    sacl    ialfa

    lacc    i2,1            ;ibeta = (2 * i2 + i1) / sqrt(3)
    add     i1              ;
    sacl    tmp             ;
    lt      tmp             ;
    mpy     SQRT3inv        ;SQRT3inv = (1 / sqrt(3)) = 093dh
                            ;4.12 format = 0.577350269
    pac                     ;
    sach    ibeta,4
*** END Sampled current scaling and (a,b,c) -> (alfa,beta) axis transformation

*******************************************
* Sine and cosine wave calculation from
* theta values using sine look-up table
*******************************************
    lacc    theta            ;theta range is [0;1000h] 4.12 format = [0;360] degrees
                            ;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    indice1         ;
    add     #sintab         ;
    sacl    tmp             ;
    lar     ar5,tmp         ;
    nop
    nop                     ;
    mar     *,ar5
    lacl    *               ;
    nop
    sacl    seno1           ;now we have sine value

    lacl    indice1         ;the same thing for cosine ... cos(theta) = sin(theta+90°)
    add     #040h           ;90 degrees = 40h elements of the table
    and     #0ffh           ;
    sacl    indice1         ;we use the same pointer (we don't care)
    add     #sintab         ;
    sacl    tmp             ;
    lar     ar5,tmp         ;
    lacc    *               ;
    sacl    coseno          ;now we have cosine value
*** END Sine and cosine wave calculation from theta values using sine look-up table

*******************************************
* d-axis and q-axis current calculation
* (alfa,beta) -> (d,q) axis transformation
* id = ialfa * cos(theta) + ibeta * sin(theta)
* iq =-ialfa * sin(theta) + ibeta * cos(theta)
*******************************************
```

```
    lacc    #0
    lt      ibeta           ;TREG0=ibeta
    mpy     seno1           ;PREG=ibeta*sin(theta)
    lta     ialfa           ;ACC+=PREG ; TREG0=ialfa
    mpy     coseno          ;PREG=ialfa*cos(theta)
    mpya    seno1           ;ACC+=PREG ; PREG=ialfa*sin(theta)
    sach    id,4
    lacc    #0              ;ACC=0
    lt      ibeta           ;TREG0=ibeta
    mpys    coseno          ;ACC-=(PREG=ialfa*sin(theta))
    apac                    ;ACC+=PREG
    sach    iq,4
*** END d-axis and q-axis current calculation

*************************************************************
* q-axis current regulator with integral component correction
* (iq,iqr)->(Vqr)
*************************************************************
    lacc    iqr
    sub     iq
    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    Vqr             ;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
*** END q-axis current regulator with integral component correction

*************************************************************
* d-axis current regulator with integral component correction
* (id,idr)->(Vdr)
*************************************************************
    lacc    idr
    sub     id
    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    Vdr             ;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
*** END d-axis current regulator with integral component correction

*********************************************
* alfa-axis and beta-axis voltages calculation
* (d,q) -> (alfa,beta) axis transformation
* Vbetar = Vqr * cos(theta) + Vdr * sin(theta)
* Valfar =-Vqr * sin(theta) + Vdr * cos(theta)
*********************************************
    lacc    #0
    lt      Vdr             ;TREG0=Vdr
    mpy     seno1           ;PREG=Vdr*sin(theta)
    lta     Vqr             ;ACC+=PREG ; TREG0=Vqr
    mpy     coseno          ;PREG=Vqr*cos(theta)
    mpya    seno1           ;ACC+=PREG ; PREG=Vqr*sin(theta)
    sach    Vbetar,4
    lacc    #0              ;ACC=0
    lt      Vdr             ;TREG0=Vdr
    mpys    coseno          ;ACC-=(PREG=Vqr*sin(theta))
    apac                    ;ACC+=PREG
    sach    Valfar,4
*** END alfa-axis and beta-axis voltages calculation

*********************************************
* Phase 1(=a) 2(=b) 3(=c) Voltage calculation
* (alfa,beta) -> (a,b,c) axis transformation
* Ua = Valfar
* Ub = (-Valfar + sqrt(3) * Vbetar) / 2
* Uc = (-Valfar - sqrt(3) * Vbetar) / 2
*********************************************
    lt      Vbetar          ;TREG0=Vbetar
    mpy     SQRT32          ;PREG=Vbetar*(SQRT(3)/2)
    pac                     ;ACC=PREG
    sub     Valfar,11       ;ACC-=Valfar*2^11
```

```
    sach    Ub,4            ;
    pac                     ;ACC=PREG
    neg                     ;ACC=-ACC
    sub     Valfar,11       ;ACC-=Valfar*2^11
    sach    Uc,4            ;
    lacl    Valfar          ;ACC=Valfar
    sacl    Ua              ;Ua=ACCL
*** END Phase 1(=a) 2(=b) 3(=c) Voltage calculation

*********************************************
* 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 = Vbetar
* Vb = (-Vbetar + sqrt(3) * Valfar) / 2
* Vc = (-Vbetar - sqrt(3) * Valfar) / 2
*********************************************
    lt      Valfar          ;TREG0=Valfar
    mpy     SQRT32          ;PREG=Valfar*(SQRT(3)/2)
    pac                     ;ACC=PREG
    sub     Vbetar,11       ;ACC-=Vbetar*2^11
    sach    Vb,4
    pac                     ;ACC=PREG
    neg                     ;ACC=-ACC
    sub     Vbetar,11       ;ACC-=Vbetar*2^11
    sach    Vc,4
    lacl    Vbetar          ;ACC=Vbetar
    sacl    Va              ;Va=ACCL
*** END Phase 1(=a) 2(=b) 3(=c) Voltage calculation

************************************
* SPACE VECTOR Pulse Width Modulation
* (see SVPWM references)
************************************
    lt      vDCinvTc
    mpy     SQRT32
    pac
    sach    tmp,4
    lt      tmp
    mpy     Vbetar
    pac
    sach    X,4
    lacc    X               ;ACC = Vbetar*K1
    sach    accb
    sacl    accb+1          ;ACCB = Vbetar*K1
    sacl    X,1             ;X=2*Vbetar*K1
    lt      vDCinvTc
    splk    #1800h,tmp
    mpy     tmp             ;implement mpy #01800h
    pac
    sach    tmp,4
    lt      tmp
    mpy     Valfar
    pac
    sach    tmp,4
    lacc    tmp             ;reload ACC with Valfar*K2
    add     accb+1
    add     accb,16
    sacl    Y               ;Y = K1 * Vbetar + K2 * Valfar
    sub     tmp,1
    sacl    Z               ;Z = K1 * Vbetar - K2 * Valfar

*** 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


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    #i1             ;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
*** END DAC out channel 'da1'

*** DAC out channel 'da2'
    lacc    #i1             ;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
*** END DAC out channel 'da2'

*** DAC out channel 'da3'
    lacc    #i1             ;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
*** END DAC out channel 'da3'

*** DAC out channel 'da4'
    lacc    #i1             ;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 channel 'da4'

    OUT     tmp,DAC_VAL     ;start conversion

    ldp     #IFRA>>7
    splk    #0200h,IFRA     ;Clear all flags, may be change with only T1 underflow int.

    ldp     #DP_PF2
    splk    #0FF88h,PCDATDIR;bit IOPC7 set to 1

*** END: PWM enable

    b       ContextRestoreReturn
*END ControlRoutine


_c_int0:
************************
* Board general settings
************************
    clrc    CNF
    clrc    xf

*****************************************************************
* Function to disable the watchdog timer                       *
```

```
*****************************************************************
    ldp     #DP_PF1
    splk    #006Fh, WD_CNTL
    splk    #05555h, WD_KEY
    splk    #0AAAAh, WD_KEY
    splk    #006Fh, WD_CNTL

*****************************************************************
* Function to initialise the Event Manager                      *
* GPTimer 1 => Full PWM                                         *
* Enable Timer 1==0 interrupt on INT2 and CAP1 on INT4          *
* Capture 1 reads tacho input                                   *
* All other pins are IO                                         *
*****************************************************************
    ; Set up SYSCLK and PLL for C24 EVM with 10MHz External Clk
    ldp     #DP_PF1
    splk    #00000010b,CKCR0;PLL disabled
                            ;LPM0
                            ;ACLK enabled
                            ;SYSCLK 5MHz
;    splk    #10110001b,CKCR1;10MHz clk in for ACLK
    splk    #01100000b,CKCR1;20MHz clk-in for ACLK
                            ;Do not divide PLL
                            ;PLL ratio x1
    splk    #10000011b,CKCR0;PLL enabled
                            ;LPM0
                            ;ACLK enabled
                            ;SYSCLK 10MHz

    ; Set up CLKOUT to be SYSCLK
    splk    #40C0h,SYSCR

    ; Clear all reset variables
    lacc    SYSSR
    and     #69FFh
    sacl    SYSSR

    ; Set up zero wait states for external memory
    lacc    #0004h
    ldp     #tmp
    sacl    tmp
    out     tmp,WSGR

    ; Clear All EV Registers
    zac
    ldp     #DP_EV
    sacl    GPTCON
    sacl    T1CNT
    sacl    T1CMP
    sacl    T1PER
    sacl    T1CON
    sacl    T2CNT
    sacl    T2CMP
    sacl    T2PER
    sacl    T2CON
    sacl    T3CNT
    sacl    T3CMP
    sacl    T3PER
    sacl    T3CON
    sacl    COMCON
    sacl    ACTR
    sacl    SACTR
    sacl    DBTCON
    sacl    CMPR1
    sacl    CMPR2
    sacl    CMPR3
    sacl    SCMPR1
    sacl    SCMPR2
    sacl    SCMPR3
```

```
    sacl    CAPCON
    sacl    CAPFIFO
    sacl    FIFO1
    sacl    FIFO2
    sacl    FIFO3
    sacl    FIFO4

    ; Initialise PWM        ;1us software dead-band
                            ;Nicol activation
    splk    #0999h,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    #100,CMPR1
    splk    #200,CMPR2
    splk    #300,CMPR3
    splk    #0000h,DBTCON   ;no dead band from Nicol
    splk    #0207h,COMCON   ;FIRST enable PWM operation
                            ;Reload Full Compare when T1CNT=0
                            ;Disable Space Vector
                            ;Reload Full Compare Action when T1CNT=0
                            ;Enable Full Compare Outputs
                            ;Disable Simple Compare Outputs
                            ;Full Compare Units in PWM Mode
    splk    #8207h,COMCON   ;THEN enable Compare operation

    splk    #PWMPRD,T1PER   ;Set T1 period
    splk    #0,T1CNT
    splk    #0A800h,T1CON   ;Ignore Emulation suspend
                            ;Cont Up/Down Mode
                            ;x/1 prescalar
                            ;Use own TENABLE
                            ;Disable Timer,enable later
                            ;Internal Clock Source
                            ;Reload Compare Register when T1CNT=0
                            ;Disable Timer Compare operation

    ; Enable Timer 1
    lacc    T1CON
    or      #40h
    sacl    T1CON

*****************************************************************
* MICHEL 23/10/96 add for Nicol Board
* 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    #000Fh,OPCRA    ;ADCIN 0-1-8-9 enabled
                            ;IOPB 1 enabled
    splk    #0079h,OPCRB    ;IOPC 0-3-7 enabled
    splk    #0FF02h,PBDATDIR;IOPB 1 conf. as output, set to logic 1
                            ;used to re-enable protection circuitry
                            ;after a fault on ACPM750E
    splk    #0FF08h,PCDATDIR;IOPC 0-3-7 conf. as output,
                            ;IOPC 3 set to logic 1
*** END: PWM enable

*****************************************************************
* Incremental encoder initialization
* Capture for Incremental encoder correction with Xint2
*****************************************************************
    ldp     #DP_EV
    splk    #0000h,T3CNT    ;configure counter register
    splk    #00FFh,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
```

```
*** END encoder/capture initialization

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

*** END A/D initialization

*****************************************************************
* Variables initialization
*****************************************************************
    ldp     #i1
    lacc    ismax
    sacl    iqrmax
    neg
    sacl    iqrmin
    zac
    sacl    iqr
    sacl    idr
    sacl    n_ref
    sacl    idr
    sacl    indice1
    sacl    xid
    sacl    xiq
    sacl    xispeed
    sacl    upi
    sacl    elpi
    sacl    Va
    sacl    Vb
    sacl    Vc
    sacl    faultreset
    splk    #24,da1
    splk    #25,da2
    splk    #42,da3
    splk    #0,da4
    splk    #VKcurrent,Kcurrent
    splk    #VSQRT3inv,SQRT3inv
    splk    #VSQRT32,  SQRT32
    splk    #VKi,      Ki
    splk    #VKpi,     Kpi
    splk    #VKcor,    Kcor
    splk    #VKispeed, Kispeed
    splk    #VKpispeed,Kpispeed
    splk    #VKcorspeed,Kcorspeed
    splk    #VKiweak,  Kiweak
    splk    #VKpiweak, Kpiweak
    splk    #VKcorweak,Kcorweak
    splk    #Vismax,   ismax
    splk    #VKencoder,Kencoder

*****************************************************************
* Table initialization
*****************************************************************
    mar     *,AR5
    lar     AR5,#sintab
    rpt     #255
    blpd    #sintab_flash,*+

    setc    OVM
    spm     0               ;no shift after multiplication
    setc    sxm             ;no sign extension
*** END initializations
```

```
*****************************************************************
* Code added to make program run without UART interface
*****************************************************************
    splk    #122h,vDCinvTc  ;Tc/vDC/2 or PWMPRD/vDC
                            ;The DC voltage is 310V
                            ;The BEMF at 3000rpm is Vbase
                            ;vDC is VDCpu
                            ;vDC = VDC/Vbase = 2.07 with Vbase=150V

    splk    #000h,initphase ;initialization phase at startup

*****************************************************************
* Initialize ar4 as the stack for context save
* space reserved: DARAM B2 60h-80h (page 0)
*****************************************************************
    lar     ar4,#79h
    lar     ar5,#60h

*****************************************************************
* Enable Interrupts
*****************************************************************
    ; Clear EV IFR and IMR regs
    ldp     #DP_EV
    splk    #07FFh,IFRA
    splk    #00FFh,IFRB
    splk    #000Fh,IFRC

    ; Enable T1 Underflow Int
    splk    #0201h,IMRA     ;PDPINT is enabled
    splk    #0000h,IMRB
    splk    #0000h,IMRC

    ; Enable XINT2 interruption for encoder synchronization
    ldp     #DP_PF1
    splk    #0006 ,XINT2_CNTL   ;set Pin as an input
    splk    #0007 ,XINT2_CNTL   ;set Pin as an input
                                ;clear flag, detect rising edge
                                ;low priority, enable interrupt (p6.41)

    ;Set IMR for INT2 and INT4 and clear any Flags
    ;INT2 (PWM interrupt) is used for motor control synchronization
    ;INT4 () is used for encoder synchronization
    ldp     #0h
    lacc    #0FFh
    sacl    IFR
    lacc    #0000010b
    sacl    IMR

    ldp     #i1             ;set the right control variable page
    clrc    INTM            ;enable all interrupts, now we may serve
                            ;interrupts
*** END Enable Interrupts

*****************************************************************
* Serial communication initialization
*****************************************************************
    ldp     #DP_PF1
    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


***************
* Virtual Menu
***************
```

```
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    vDCinvTc          ;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
    lacc    option
    sub     #037h             ;is it option 7 ?
    bcnd    notseven,neq      ;if not branch to notseven

*****************************
* Option 7): Kpiweak , Kiweak , Kcorweak
*****************************
navail71
    ldp     #DP_PF1
    bit     SCIRXST,BIT6      ;is there any character available (8 LSB)?
    bcnd    navail71,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
navail72
    ldp     #DP_PF1
    bit     SCIRXST,BIT6      ;8 MSB available ?
    bcnd    navail72,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    Kpiweak           ;store it
navail73
    ldp     #DP_PF1
    bit     SCIRXST,BIT6      ;is there any character available (8 LSB)?
```

```
    bcnd    navail73,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
navail74
    ldp     #DP_PF1
    bit     SCIRXST,BIT6      ;8 MSB available ?
    bcnd    navail74,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    Kiweak            ;store it
navail75
    ldp     #DP_PF1
    bit     SCIRXST,BIT6      ;is there any character available (8 LSB)?
    bcnd    navail75,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
navail76
    ldp     #DP_PF1
    bit     SCIRXST,BIT6      ;8 MSB available ?
    bcnd    navail76,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    Kcorweak          ;store it
    b       menu              ;return to the main polling cycle
*** END Option

notseven
    lacc    option
    sub     #038h             ;is it option 8 ?
    bcnd    noteight,neq      ;if not branch to noteight

*****************************
* Option 8): faultreset
*****************************
navail81
    ldp     #DP_PF1
    bit     SCIRXST,BIT6      ;is there any character available (8 LSB)?
    bcnd    navail81,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
navail82
    ldp     #DP_PF1
    bit     SCIRXST,BIT6      ;8 MSB available ?
    bcnd    navail82,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    faultreset        ;store it
    b       menu              ;return to the main polling cycle
*** END Option 8): faultreset

noteight
    b       menu
```

# Appendix B. Linker File

```
/*****************************************************************/
/*                      TEXAS INSTRUMENTS                        */
/*****************************************************************/
/*   File Name:  link.cmd                                        */
/*   Originator: Michel Platnic                                  */
/*                                                               */
/*   Description:Link command file                               */
/*  MEMORY SPECIFICATION FOR THE MCK240  or EVMF240              */
/*                                                               */
/*   Target:     TMS320F240, MCK240 or EVMF240                   */
/*   status:     Working                                         */
/*                                                               */
/*   History:    Completed on 26 April 98                        */
/*****************************************************************/

MEMORY
{
 PAGE 0:
       FLASH_VEC   : origin =    0h, length =   40h
       FLASH       : origin =  040h, length = 0FC0h
       FLASH_TAB   : origin = 1000h, length = 0200h

       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                       */
        table_f     : { } > FLASH_TAB PAGE 0  /* Table in flash program mem */
        blockb2     : { } > BLK_B22   PAGE 1  /* Data storage on DP 0       */
        .data       : { } > BLK_B0    PAGE 1
        .bss        : { } > BLK_B0    PAGE 1  /* GLOBAL VARS, STACK, HEAP   */
        table       : { } > BLK_B1    PAGE 1
}
```

# Appendix C. Sinewave Table

```
.word   0          .word   3166         .word   61833
.word   101        .word   3102         .word   61791
.word   201        .word   3035         .word   61752
.word   301        .word   2967         .word   61714
.word   401        .word   2896         .word   61679
.word   501        .word   2824         .word   61647
.word   601        .word   2751         .word   61616
.word   700        .word   2675         .word   61588
.word   799        .word   2598         .word   61563
.word   897        .word   2520         .word   61540
.word   995        .word   2440         .word   61519
.word   1092       .word   2359         .word   61500
.word   1189       .word   2276         .word   61484
.word   1285       .word   2191         .word   61471
.word   1380       .word   2106         .word   61460
.word   1474       .word   2019         .word   61451
.word   1567       .word   1931         .word   61445
.word   1660       .word   1842         .word   61441
.word   1751       .word   1751         .word   61440
.word   1842       .word   1660         .word   61441
.word   1931       .word   1567         .word   61445
.word   2019       .word   1474         .word   61451
.word   2106       .word   1380         .word   61460
.word   2191       .word   1285         .word   61471
.word   2276       .word   1189         .word   61484
.word   2359       .word   1092         .word   61500
.word   2440       .word   995          .word   61519
.word   2520       .word   897          .word   61540
.word   2598       .word   799          .word   61563
.word   2675       .word   700          .word   61588
.word   2751       .word   601          .word   61616
.word   2824       .word   501          .word   61647
.word   2896       .word   401          .word   61679
.word   2967       .word   301          .word   61714
.word   3035       .word   201          .word   61752
.word   3102       .word   101          .word   61791
.word   3166       .word   0            .word   61833
.word   3229       .word   65435        .word   61877
.word   3290       .word   65335        .word   61924
.word   3349       .word   65235        .word   61972
.word   3406       .word   65135        .word   62023
.word   3461       .word   65035        .word   62075
.word   3513       .word   64935        .word   62130
.word   3564       .word   64836        .word   62187
.word   3612       .word   64737        .word   62246
.word   3659       .word   64639        .word   62307
.word   3703       .word   64541        .word   62370
.word   3745       .word   64444        .word   62434
.word   3784       .word   64347        .word   62501
.word   3822       .word   64251        .word   62569
.word   3857       .word   64156        .word   62640
.word   3889       .word   64062        .word   62712
.word   3920       .word   63969        .word   62785
.word   3948       .word   63876        .word   62861
.word   3973       .word   63785        .word   62938
.word   3996       .word   63694        .word   63016
.word   4017       .word   63605        .word   63096
.word   4036       .word   63517        .word   63177
.word   4052       .word   63430        .word   63260
```

```
.word   4065        .word   63345        .word   63345
.word   4076        .word   63260        .word   63430
.word   4085        .word   63177        .word   63517
.word   4091        .word   63096        .word   63605
.word   4095        .word   63016        .word   63694
.word   4096        .word   62938        .word   63785
.word   4095        .word   62861        .word   63876
.word   4091        .word   62785        .word   63969
.word   4085        .word   62712        .word   64062
.word   4076        .word   62640        .word   64156
.word   4065        .word   62569        .word   64251
.word   4052        .word   62501        .word   64347
.word   4036        .word   62434        .word   64444
.word   4017        .word   62370        .word   64541
.word   3996        .word   62307        .word   64639
.word   3973        .word   62246        .word   64737
.word   3948        .word   62187        .word   64836
.word   3920        .word   62130        .word   64935
.word   3889        .word   62075        .word   65035
.word   3857        .word   62023        .word   65135
.word   3822        .word   61972        .word   65235
.word   3784        .word   61924        .word   65335
.word   3745        .word   61877        .word   65435
.word   3703
.word   3659
.word   3612
.word   3564
.word   3513
.word   3461
.word   3406
.word   3349
.word   3290
.word   3229
```

# Appendix D. Qbasic User Interface

```
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

speedref = 0
init = 0
VDC = 310
da1 = 24: da2 = 25
da3 = 42: da4 = 3
Ki = .03
Kpi = .6
Kcor = .05
Kispeed = .03
Kpispeed = 6.5
Kcorspeed = .0046
Kiweak = .03
Kpiweak = .6
Kcorweak = .05

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

Tc = 600: REM PWM period in us
speedpu = 3000: REM base speed
ibase = 4.1: REM base current
Vbase = 152: REM base voltage

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$(17) = "theta"
daout$(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"
```

```
daout$(41) = "encoder"
daout$(42) = "Vr"
daout$(43) = "iqrmin"
daout$(44) = "iqrmax"


nDA = 11

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: PRINT "<8>"; : COLOR 2: PRINT " Re-enable after fault"

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 9, 50: PRINT "   <7>"; : COLOR 2: PRINT " Kpiweak   ("; Kpiweak; "pu)"
COLOR 10: LOCATE 10, 50: PRINT "      "; : COLOR 2: PRINT " Kiweak    ("; Kiweak; "pu)"
COLOR 10: LOCATE 11, 50: PRINT "      "; : COLOR 2: PRINT " Kcorweak  ("; Kcorweak; "pu)"

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

init88 = CLNG(init * 256)
VDCpu = VDC / Vbase
VDCinvTc = Tc / VDCpu
Kpipu = 4096 * Kpi
Kipu = 4096 * Ki
Kcor = (Ki / Kpi)
Kcorpu = 4096 * Kcor
Kpispeedpu = 4096 * Kpispeed
Kispeedpu = 4096 * Kispeed
Kcorspeed = (Kispeed / Kpispeed)
Kcorspeedpu = 4096 * Kcorspeed
Kpiweakpu = 4096 * Kpiweak
Kiweakpu = 4096 * Kiweak
Kcorweak = (Kiweak / Kpiweak)
Kcorweakpu = 4096 * Kcorweak

DO
a$ = INKEY$
LOOP UNTIL ((a$ <= "8") 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$(da4 AND
255)
    GOTO 1
CASE "3"
    IF init = 1 THEN init = 0 ELSE init = 1
    IF (init >= 255.9960938#) THEN init = 255.9960938#
    IF (init < 0) THEN init = 0
    init88 = CLNG(init * 256)
    PRINT #1, "3"; CHR$(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$)
    IF (Kpi >= 7.9) THEN Kpi = 7.9
    IF (Kpi <= 0) THEN Kpi = 0
51
    PRINT "                      Ki ("; Ki; " ) : ";
    INPUT Ki$
    IF Ki$ = "" THEN 52
```

```
    Ki = VAL(Ki$)
    IF (Ki >= 1) THEN Ki = 1
    IF (Ki <= 0) THEN Ki = 0
52
    Kpipu = 4096 * Kpi
    Kipu = 4096 * Ki
    Kcor = (Ki / Kpi)
    Kcorpu = 4096 * Kcor
    PRINT #1, "5"; CHR$(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 "7"
    REM 4.12 format
    PRINT a$; ") ";
    PRINT "Kpiweak ("; Kpiweak; " ) : ";
    INPUT Kpiweak$
    IF Kpiweak$ = "" THEN 71
    Kpiweak = VAL(Kpiweak$)
    IF (Kpiweak >= 7.9) THEN Kpiweak = 7.9
    IF (Kpiweak <= 0) THEN Kpiweak = 0
71
    PRINT "                     Kiweak ("; Kiweak; " ) : ";
    INPUT Kiweak$
    IF Kiweak$ = "" THEN 72
    Kiweak = VAL(Kiweak$)
    IF (Kiweak >= 1) THEN Kiweak = 1
    IF (Kiweak <= 0) THEN Kiweak = 0
72
    Kpiweakpu = 4096 * Kpiweak
    Kiweakpu = 4096 * Kiweak
    Kcorweak = (Kiweak / Kpiweak)
    Kcorweakpu = 4096 * Kcorweak
    REM Send "Option" - "LSB" - "MSB"
    PRINT #1, "7"; CHR$(Kpiweakpu AND 255); CHR$((Kpiweakpu AND 65280) / 256);
CHR$(Kiweakpu AND 255); CHR$((Kiweakpu AND 65280) / 256); CHR$(Kcorweakpu AND 255);
CHR$((Kcorweakpu AND 65280) / 256)
    GOTO 1

CASE "8"
    faultreset = 1
    init88 = CLNG(faultreset * 256)
    PRINT #1, "8"; CHR$(init88 AND 255); CHR$((init88 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)
    PRINT #1, "3"; CHR$(init88 AND 255); CHR$((init88 AND 65280) / 256)
    PRINT #1, "4"; CHR$(VDCinvTc AND 255); CHR$((VDCinvTc AND 65280) / 256)
    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)
    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)
    PRINT #1, "7"; CHR$(Kpiweakpu AND 255); CHR$((Kpiweakpu AND 65280) / 256);
CHR$(Kiweakpu AND 255); CHR$((Kiweakpu AND 65280) / 256); CHR$(Kcorweakpu AND 255);
CHR$((Kcorweakpu AND 65280) / 256)
    GOTO 1
END SELECT
CLOSE #1
```

**IMPORTANT NOTICE**

Texas Instruments and its subsidiaries (TI) reserve the right to make changes to their products or to discontinue any product or service without notice, and advise customers to obtain the latest version of relevant information to verify, before placing orders, that information being relied on is current and complete. All products are sold subject to the terms and conditions of sale supplied at the time of order acknowledgement, including those pertaining to warranty, patent infringement, and limitation of liability.

TI warrants performance of its semiconductor products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

CERTAIN APPLICATIONS USING SEMICONDUCTOR PRODUCTS MAY INVOLVE POTENTIAL RISKS OF DEATH, PERSONAL INJURY, OR SEVERE PROPERTY OR ENVIRONMENTAL DAMAGE (“CRITICAL APPLICATIONS"). TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS. INCLUSION OF TI PRODUCTS IN SUCH APPLICATIONS IS UNDERSTOOD TO BE FULLY AT THE CUSTOMER'S RISK.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards must be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance or customer product design. TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used. TI's publication of information regarding any third party's products or services does not constitute TI's approval, warranty, or endorsement thereof.