Implementation of a Speed Field Orientated Control of Three Phase AC Induction Motor using TMS320F240

Literature Number: BPRA076
Texas Instruments Europe
March 1998

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.

Contents

List of Figures

Implementation of a Sensorless Speed Controlled Brushless DC Drive using the TMS320F240

ABSTRACT

Since the integration of high computationnal DSP power with all necessary motor control peripherals into a single chip, TMS320F240, it has become possible to design and implement a highly efficient and accurate AC induction drive control. The AC induction drive presented here is based on document [6] and on a dedicated and exhaustive study of this DSP solution. Both the theoretical and practical characteristics of this drive implementation allow the reader to quickly gain an understanding of the Field Orientated Control of an induction motor. As such, the reader might not only gain a **short time to market solution,** but also a speed adjustable, reliable and highly effective induction drive.

1. Introduction

For many years the asynchronous drive has been preferred for a variety of industrial applications because of its robust nature and simplicity of control. Until a few years ago, the asynchronous motor could either be plugged directly into the network or controlled by means of the well-known scalar V/f method. When designing a variable speed drive, both methods present serious drawbacks in terms of the drive efficiency, the drive reliability and EMI troubles. With the first method, even simple speed variation is impossible and its system integration highly dependent on the motor design (starting torque vs maximum torque, torque vs inertia, number of pole pairs). The second solution is able to provide a speed variation but does not handle real time control as the implemented is valid only in steady stage. This leads to over-currents and over-heating, which necessitate a drive which is then oversized and no longer cost effective.

It is the real time-processing properties of silicon, such as the TMS320F240 DSP controller, and the accurate asynchronous motor model that have resulted in the development of a highly reliable drive with highly accurate and variable speed controls. Application of The Field Orientated Control to the AC induction drive results in the instant control of a high performance drive (short response time with neither the motor nor the power component oversized). The ability to achieve such control renders the asynchronous drive a highly advantageous system for both home appliances and for industrial or automotive applications. Key advantages are the robust nature of the drive, its reliability and efficiency, the cost effectiveness of both the motor and the drive, the high torque at zero speed, the speed variation capacity, the extended speed range, the direct torque and flux control and the excellent dynamic behaviour.

In this document we will look not only at the complete integration of the software, but also at the theoretical and practical aspects of the application. By the end of this report the reader will have gained an understanding of each of the developmental steps and will be able to apply this asynchronous drive solution to his own system.

The first section deals with the presentation of the field orientated controlled AC induction drive; it explains the AC induction motor, the control hardware, the power electronics hardware as well as the complete FOC structure. The second section deals with the implementation of TMS320F240 drive speed control. Here, the details of how and why the software is organized, the Per Unit model, the numerical consideration, the current and speed sensing and scaling, the regulators, the system transformations, the current model, the field weakening and the space vector modulation are fully explained step by step. Results of intermediate experiments illustrate the presentations in each block. At the end of this document flowcharts have been incorporated to explain the operating system. The final experiment results demonstrate the dynamic behaviour and effectiveness of the drive.

2. The Field Orientated Controlled AC Induction Drive

This chapter presents each component of the AC induction drive. The different sub-chapters cover the motor parameters and the implemented control structure, including both the control and the power hardware

2.1 The AC induction motor

The AC induction machine used in this explanation is a single cage three phase Y-connected motor. The rated value and the parameters of this motor are as follows:

Rated power	P_n=500W
Rated voltage	V_n=127V rms (phase)
Rated current	I_n=2.9A rms
Rated speed	1500rpm
Pole pairs	2
Slip	0.066
Rated torque	$M_{nom} = \frac{P_{nom}}{\omega_{nom}} = \frac{500}{\left(\frac{2\pi \times 1400}{60}\right)} = 3.41\text{Nm}$
Stator resistance (R_S)	4.495Ω
Magnetizing inductance (L_H)	149mH
Stator leakage inductance ($L_{\sigma S}$)	16mH
Stator inductance (L_S=$L_{\sigma S}$+L_H)	165mH
Rotor leakage inductance ($L_{\sigma R}$)	13mH
Rotor inductance (L_R=$L_{\sigma R}$+L_H)	162mH
Rotor resistance (R_R)	5.365Ω
Rotor inertia	$0.95*10^{-3}$Kgm^2

An embedded incremental encoder is also provided with this motor. This is capable of 1000 pulses per revolution and is used in this application to obtain the rotor mechanical speed feedback.

2.2 The control hardware

The control hardware can be either the TMS320F240 Evaluation Module introduced by Texas Instruments or the MCK240 developed by Portescap/Technosoft. In this application the second board can be plugged directly on to the power electronics board. The two boards contain a DSP controller TMS320F240 and its oscillator, a JTAG, and an RS232 link with the necessary output connectors. The figure below depicts the EVM board.

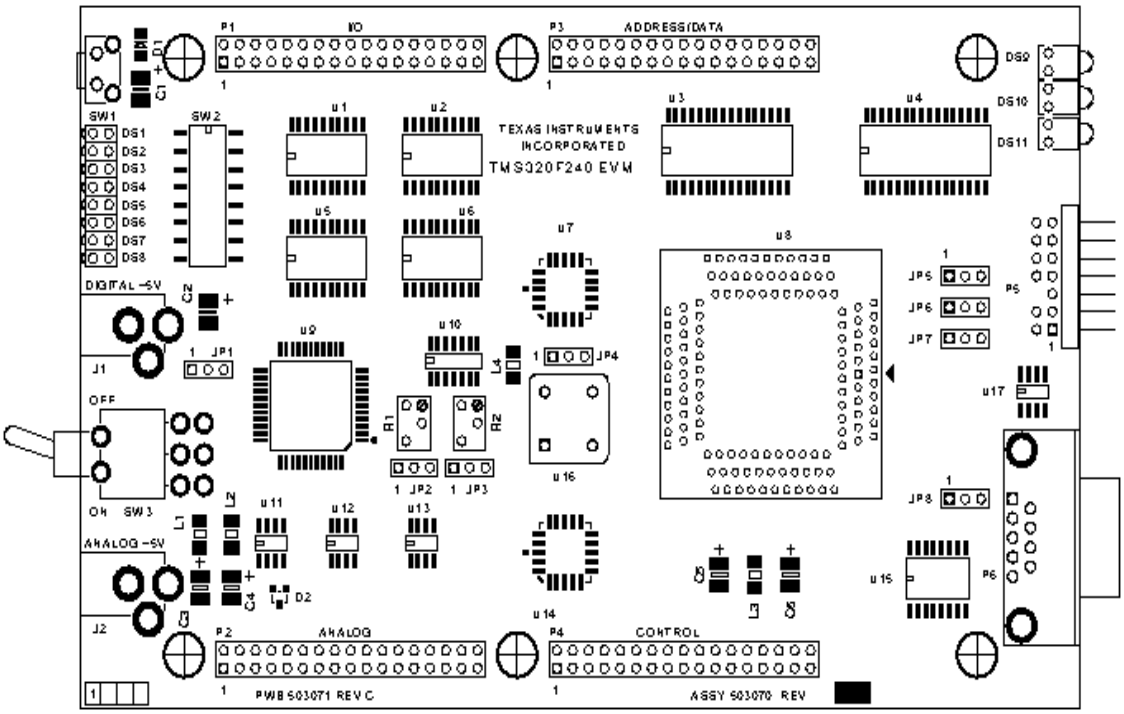

Figure 1: Top View of the TMS320F240 Evaluation Module

The EVM board provides access to any signal from the DSP Controller and contains test LED's and Digital to Analog Converters. These characteristics are particularly interesting during the developmental stage.

2.3 The Power Electronics Hardware

The power hardware used to implement and test this AC induction drive can support an input voltage of 220V and a maximum current of 10A. It is based on six power IGBT (IRGPC40F) driven by the DSP Controller via the integrated driver IR2130. The power and the control parts are insulated by means of opto-couplers. The phase current sensing is performed via two current voltage transducers supplied with +/-15V. Their maximum input current is 10A, which is converted into a 2.5V output voltage. Furthermore, this powered electronics board **supports** bus voltage measurement, control LED's and input current filter. All the power device securities are wired (Shutdown, Fault, Clearfault, Itrip, reverse battery diode, varistor peak current protection).

2.4 Complete Field Orientated Speed Control Structure Presentation

The control algorithm implemented in this application report is a rotor flux orientated control strategy, based on the Field Orientated Control structure presented in [6]. Given the position of the rotor flux and two phase currents, this generic algorithm operates the instantaneous direct torque and flux controls by means of coordinate transformations and PI regulators, thereby achieving a really accurate and efficient motor control. The generic FOC structure needs to be augmented with two modules in order to address the asynchronous drive specificity.

With the asynchronous drive, the mechanical rotor angular speed is not, by definition, equal to the rotor flux angular speed. This implies that the necessary rotor flux position can not be detected directly by the mechanical position sensor provided with the asynchronous motor used in this application. The **Current Model** block must be added to the generic structure in the block diagram. This current model [1][3][5] takes as input both i_{Sq} and i_{Sd} current as well as the rotor mechanical speed and gives the rotor flux position as output. A complete description and the software implementation for the necessary equations are given in a subsequent chapter.

The speed control of the AC induction drive is often split into two ranges: the *low speed* range, where the motor speed is below the nominal speed, and the *high speed* range, where the motor speed is higher than the nominal speed. Above the nominal speed the effective back electromotive force (which depends on both the motor speed and on the rotor flux) is high enough, given the DC bus voltage limitation, to limit the current in the winding. As such, this limits both the torque production and the drive efficiency (due to problems with magnetic saturation and heat dissipation). Where the rotor flux has been maintained at its nominal value during the *low speed* operation so as to achieve the highest mutual torque production, it must be reduced in the *high speed* operation in order to avoid magnetic saturation and the generation of too high back electromotive force. Reducing the rotor flux in this way extends the high efficiency operating range of the drive. This functionality is integrated into the **Field Weakening** module. A complete explanation and outline of the correct software implementation are given in a later chapter.

These two induction motor control dedicated modules are added to the basic FOC structure. This results in the following complete FOC AC induction drive structure:

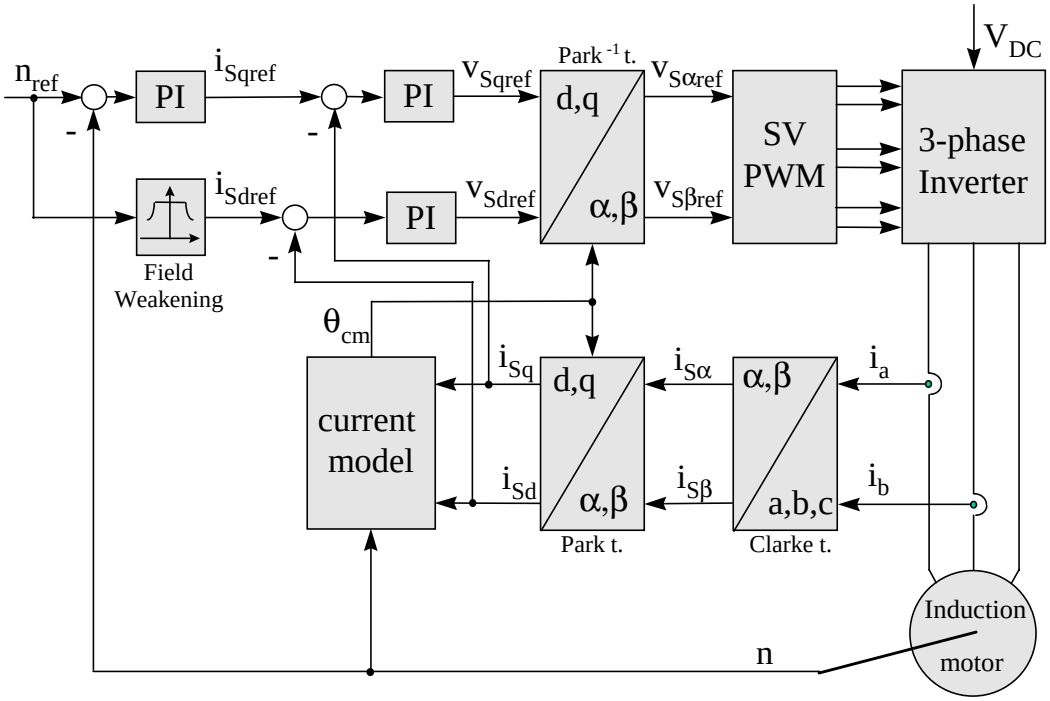

Figure 2: Complete AC Induction Drive Dedicated FOC Structure

Two phase currents feed the Clarke transformation module. These projection outputs are indicated $i_{s\alpha}$ and $i_{s\beta}$. These two components of the current provide the input of the Park transformation that gives the current in the *d,q* rotating reference frame. The i_{Sd} and i_{Sq} components are compared to the references i_{Sdref} (the flux reference) and i_{Sqref} (the torque reference). The torque command i_{Sqref} corresponds to the output of the speed regulator. The flux command i_{Sdref} is the output of the field weakening function that indicates the right rotor flux command for every speed reference. The current regulator outputs are v_{Sdref} and v_{Sqref}; they are applied to the inverse Park transformation. The output of this projection are $v_{S\alpha_{ref}}$ and $v_{S\beta_{ref,}}$ the components of the stator vector voltage in the α,β orthogonal reference frame. These are the input of the Space Vector PWM. The outputs of this block are the signals that drive the inverter. Note that both Park and inverse Park transformations require the rotor to be in flux position which is given by the current model block. This block needs the rotor resistance as a parameter. Accurate knowledge and representation of the rotor resistance is essential to achieve the highest possible efficiency from the control structure.

3. Field Orientated Speed Controlled AC Induction Drive Software Implementation

This chapter deals with the practical aspects of the drive implementation. It describes the software organization, the utilization of different variables and the handling of the DSP Controller resource. In the second part the control structure for the per unit model is presented. This explanation allows the reader to instantly adapt the given software to match the parameters of his drive. As numerical considerations have been made in order to address the problems inherent within fixed-point calculation, this software can be used with a wide range of drive parameters and regulator coefficients.

3.1 Software Organization

This software is based on two modules: the initialization module and the run module. The former is performed only once at the beginning. The second module is based on a waiting loop interrupted by the PWM underflow. When the interrupt flag is set, this is acknowledged and the corresponding Interrupt Service Routine (ISR) is served. The complete FOC algorithm is computed within the PWM ISR and thus runs at the same frequency as the chopping frequency. The waiting loop can be easily replaced by a user interface. Presentation of the interface is beyond the scope of this report, but is useful to fit the control code and to monitor the control variables. An overview of the software is given in the flow chart below:

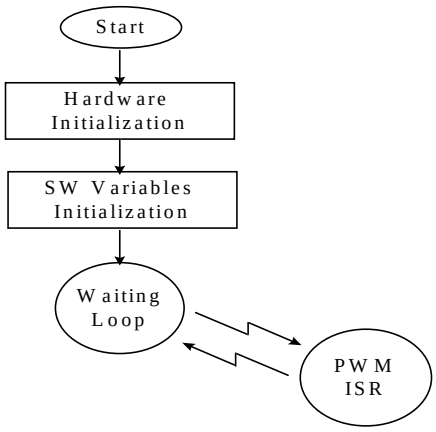

Figure 3: General Software Flowchart

The DSP Controller Full Compare Unit is used to generate the necessary pulsed signals to the power electronics board. It is programmed to generate symmetrical complementary PWM signals at a frequency of 10kHz, with TIMER1 as the time base and with the DEADBAND unit disabled. The sampling period (T) of 100 μs can be established by setting the timer period T1PER to 1000 (PWMPRD=1000).

The following figure illustrates the time diagram for the initialization and the operating system.

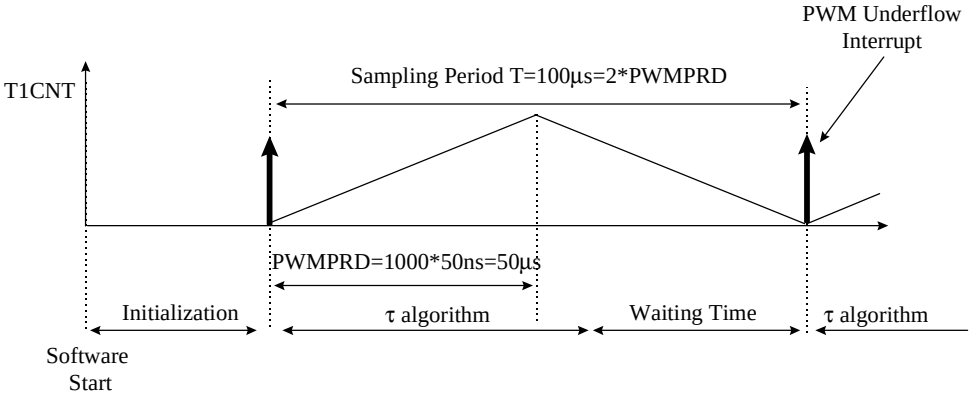

Figure 4: FOC Software Initialization and Operating System

3.1.1 DSP Controller Setup

This section is dedicated to the handling of the different DSP Controller resources (Core and peripheral settings). First of all, the PLL unit is set so that the CPUCLK runs at 20MHz based on the 10Mhz quartz provided on the EVM board. For the developmental stage it is necessary to disable the watchdog unit: first set the Vccp pin voltage to five volts utilising the EVM jumper JP5 and then set the two watchdog dedicated registers to inactive. Finally, correctly set the core and EV mask registers to enable the PWM underflow to interrupt serving.

3.1.2 Software Variables

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

i_a, i_b, i_c	*phase current*
$i_{S\alpha}$, $i_{S\beta}$	*stator current (α,β) components*
i_{Sd}, i_{Sq}	*stator current flux & torque comp.*
i_{Sdref}, i_{Sqref}	*flux and torque command*
Teta_cm or θ_{cm}	*rotor flux position*
f_s	*rotor flux speed*
i_mR, i_{mR}	*magnetizing current*
v_{Sdref}, v_{Sqref}	*(d,q) components of the stator voltage*
$v_{S\alpha ref}$, $v_{S\beta ref}$	*(α,β) components of the stator voltage (input of the SVPWM)*
v_{DC}	*DC bus voltage*
v_{DCinvT}	*constant using in the SVPWM*
v_{ref1}, v_{ref2}, v_{ref3}	*voltage reference used for SV sector determination*
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*
$i_{Sqrefmin}$, $i_{Sqrefmax}$	*speed regulator output limitation*
v_{min},v_{max}	*d,q current regulator output limitation*
K_i, K_{pi}, K_{cor}	*current regulator parameters*
K_{in}, K_{pin}, K_{corn}	*speed regulator parameters*
x_{id}, x_{iq}, x_{in}	*regulator integral components*
e_{pid}, e_{piq}, e_{pin}	*d,q-axis, speed regulator errors*
K_r, K_t, K	*current model parameters*
p_3, p_2, p_1, p_0	*field weakening polynomial coeff*
K_{speed},	*4.12 speed formatting constant*
SPEEDSTEP	*speed loop period*
speedstep	*speed loop counter*
encincr,	*encoder pulses storing variable*
speedtmp	*occurred pulses in SPEEDSTEP*
$K_{current}$	*4.12 current formatting constant*
$\sin\theta_{cm}$, sinTeta_cm,	
$\cos\theta_{cm}$, cosTeta_cm	*sine and cosine of the rotor flux position*

3.2 Base values and PU model

Since the TMS320F240 is a fixed point DSP, a per unit (pu) model of the motor has been used. In this model all quantities refer to base values. The base values are determined from the nominal values by using the following equations, where I_n, V_n , f_n are respectively the phase nominal current, the phase to neutral nominal voltage and the nominal frequency in a star-connected induction motor

$$I_b = \sqrt{2}\, I_n$$
$$V_b = \sqrt{2}\, V_n$$
$$\omega_b = 2\pi f_n$$
$$\Psi_b = \frac{V_b}{\omega_b}$$

and where I_b, V_b are the maximum values of the phase nominal current and voltage; ω_b is the electrical nominal rotor flux speed; Ψ_b is the base flux. The base values of the motor used in this asynchronous drive are stated below.

$$I_b = \sqrt{2} I_n = \sqrt{2} \cdot 2.9 = 4.1A$$
$$V_b = \sqrt{2} V_n = \sqrt{2} \cdot 127 \cong 180\,V$$
$$\omega_b = 2\pi f_n = 2\pi \cdot 50 = 314.15\ \frac{rad}{\sec}$$
$$\Psi_b = \frac{V_b}{\omega_b} = \frac{180}{314.15} = 0.571 Wb$$

The real quantities are implemented in to the control thanks to the pu quantities, which are defined as follows:

$$i = \frac{I}{I_b}$$

$$v = \frac{V}{V_b}$$

$$\psi = \frac{\Psi}{\Psi_b}$$

$$n = \frac{electrical\ rotor\ speed}{\omega_b} = \frac{pole\ pairs * mechanical\ rotor\ speed}{\omega_b}$$

$$f_S = \frac{rotor\ \ flux\ speed}{\omega_b}$$

Where i, v, ψ, n, f_S are respectively pu current, voltage, flux, electrical rotor speed and rotor flux speed. This model can be followed to ensure the easy implementation of the control algorithm into a fixed point DSP.

3.3 Magnetizing current considerations

In the classic speed range (where speed is lower or equal to the nominal speed) the FOC structure requires the magnetizing current as input. Given the following motor equivalent circuit, valid only in stationary steady state, the magnetizing current might be a priori calculated.

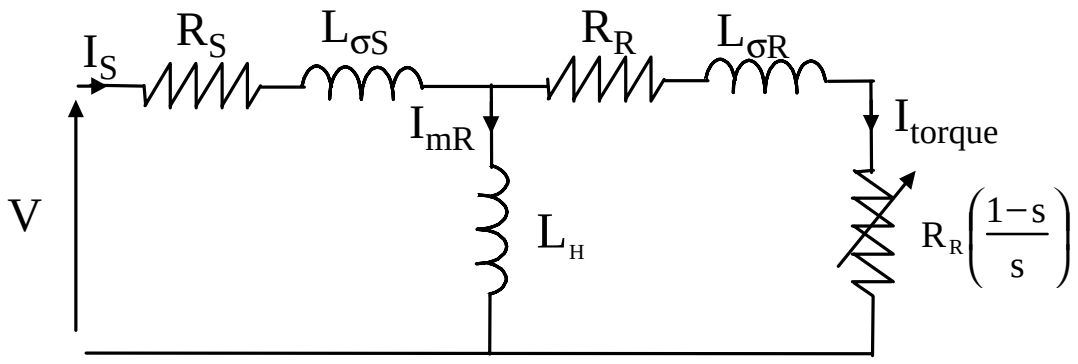

Figure 5: Steady State Phase Electrical Model

Assuming that the motor is running at nominal speed without any load (in other words that slip is equal to zero) and knowing the paramaters of the motor, then the magnetizing current is simply equal to the nominal phase voltage (in this case 127V rms) divided by the equivalent impedance.

A useful tip is to know that the magnetizing current is usually between 40% and 60% of the nominal current.

3.4 Numerical considerations

The PU model has been developed so that the software representation of speed current and flux is equal to one when the drive has reached its nominal speed under nominal load and magnetizing current. Bearing in mind that during the transient the current might reach higher values than the nominal current (I_b) in order to achieve a short response time, and assuming that the motor speed range might be extended above the nominal speed (ω_b), then every per unit value might be greater than one. This fact forces the implementation to foresee these situations and thereby determine the most suitable numerical format.

3.4.1 The numeric format determination

The numeric format used in the major part of this application is such that 4 bits are dedicated to the integer part and 12 bits are dedicated to the fractional part. This numeric format is denoted by 4.12 f. The resolution for this format is:

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

With the sign extension mode set, the link between the real quantity and its 4.12 representation is illustrated by the following chart:

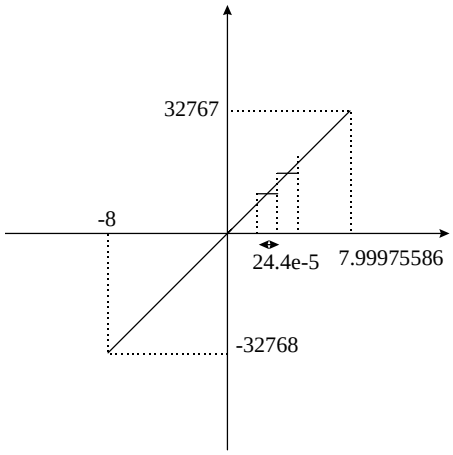

Figure 6: 4.12 Format Correspondence Diagram

The reason for selecting this particular format is that the drive control quantities are (for the most part) not greater than four times their nominal values (in other words, not greater than four when the pu model is considered). Where this is not the case,a different format will be chosen. The selection of a demonstration range of [-8;8] ensures that the software values can handle each drive control quantity, not only during steady state operation but also during transient operation. The next two paragraphs outline some of the numerical considerations and some operations with a generic *x.y* format in order to explain the different formats that can be found in this application report.

The *x.y* numeric format uses *x* bits for the integer part and *y* bits for the fractional part. The resolution is 2^{-y}; if *z* is the pu value to implement, then its software value is $z \cdot 2^{y}$ in *x.y* format. Care must be taken when performing operations with a generic *x.y* format. Adding two *x.y*-formatted numbers may result in numerical representation overflow. To avoid this kind of problem, one possible solution is to perform the addition in the high side of the Accumulator and to set the saturation bit. Another option is to assume that the result will not be out of the maximum range. This second solution can be used in this implementation if we know that the control quantities do not exceed half of the maximum value in the 4.12 format. The result can still be represented in the 4.12 format and directly considered as 4.12 format, thereby allowing for a higher level of precision.

As far as the multiplication is concerned, the result (in the 32-bit Accumulator) must either be shifted *x* position to the left and the most significant word stored, or be shifted *y* position to the right with the last significant word being stored. The stored result is in *x.y* format. The figure below shows two *x.y*-formatted 16-bit variables, that will be multiplied by one another. The result of this multiplication in *x.y* format is represented in gray in the 32-bit Accumulator. Both solutions are depicted below.

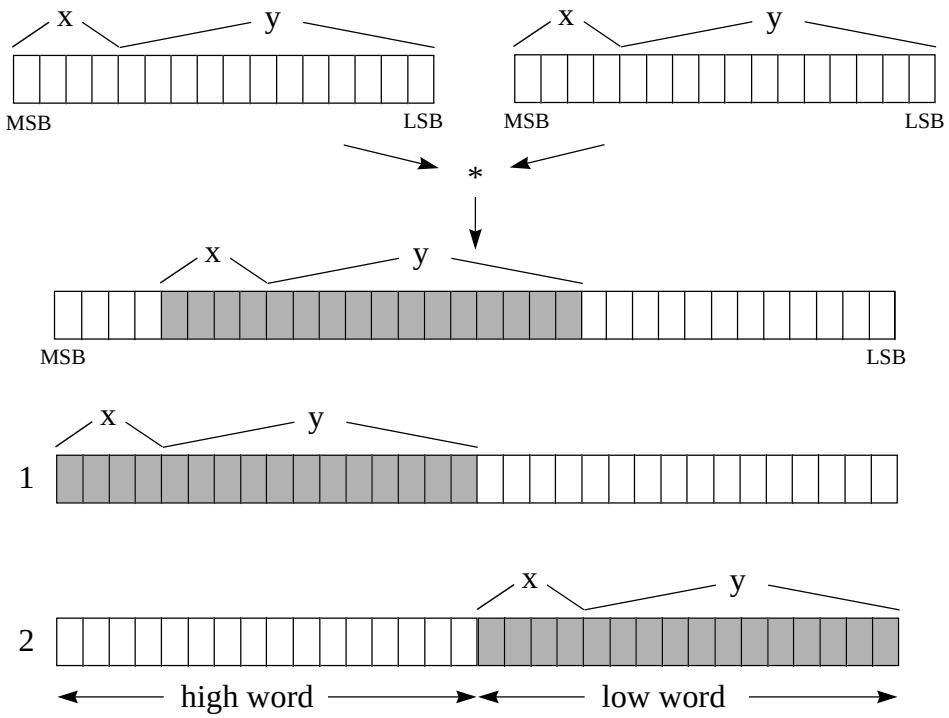

Figure 7: 1) left shift & store high accumulator, 2) right shift & store low accumulator

Note that in this application there are also constants that can not be represented by the 4.12 format. Operations requiring different formats follow exactly the same process as that explained above.

3.5 Current Sensing and Scaling

The FOC structure requires two phase currents as input. In this application current-voltage transducers (LEM type) sense these two currents. The current sensor output therefore needs to be rearranged and scaled so that it can be used by the control software as 4.12 format values. The complete process of acquiring the current is depicted in the figure below:

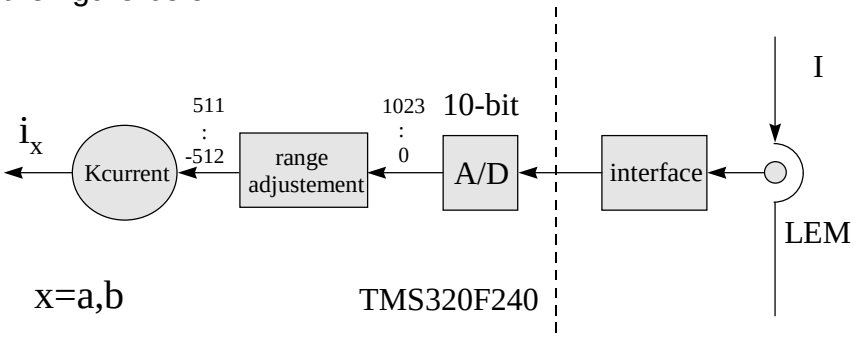

Figure 8: Current Sensing and Scaling Block Diagram

In this application the LEM output signal can be either positive or negative. This signal must therefore be translated by the analogue interface into a range of (0;5V) in order to allow the single voltage ADC module to read both positive and negative values. The block diagram below shows the different steps of the implemented current sensing:

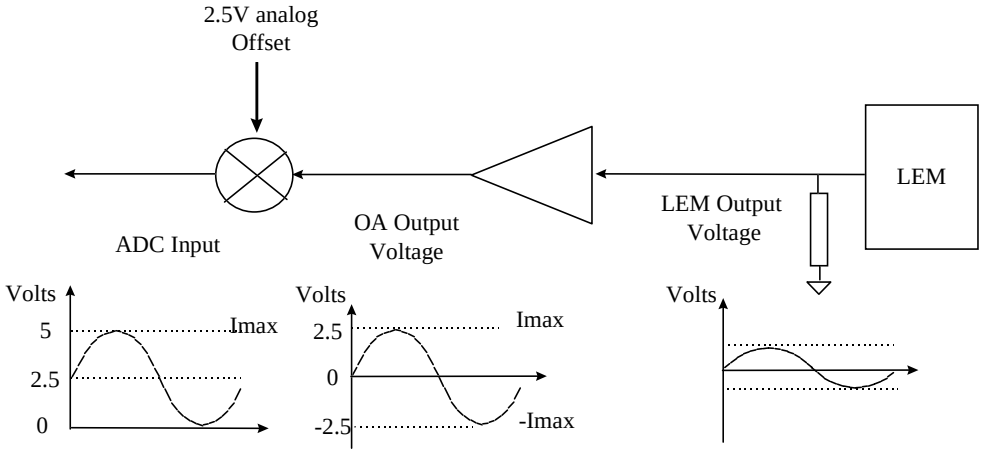

Figure 9: Current Sensing Interface Block Diagram

Note that I_{max} represents the maximum measurable current, which is not necessarily equal to the maximum phase current. This information is useful at the point where current scaling becomes necessary. The ADC input voltage is now converted into a ten bits digital value. The 2.5V analogue offset is digitally subtracted from the conversion result, thereby giving a signed integer value of the sensed current.

The result of this process is represented below:

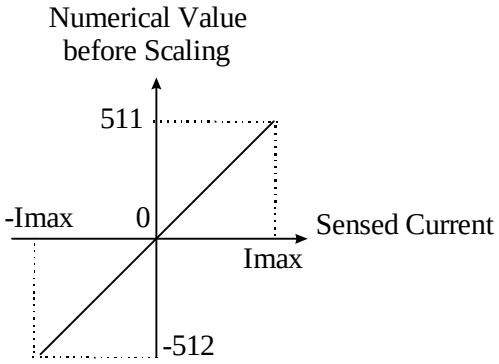

Figure 10: Sensed Current Values before Scaling

Like every other quantity in this application, the sensed phase currents must now be expressed with the pu model and then be converted into the 4.12 format. Notice that the pu 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 pu current conversion into the 4.12 format is achieved by multiplying the sensed current by the following constant:

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

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

In this application the maximum measurable current is I_{max}=10A . The constant value is:

$$K_{current} = \frac{4096}{(\frac{512 \cdot 4.1}{10})} = 19.51 \Leftrightarrow 1383h \quad 8.8f$$

Note that $K_{current}$ is outside the 4.12 format range. The most appropriate format to accommodate this constant is the 8.8 format, which has a resolution of:

$$0.00390625 = \frac{1}{2^8}$$

and the following correspondence (Figure 11):

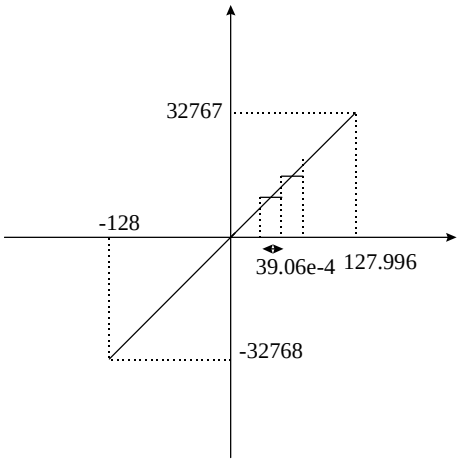

Figure 11: 8.8 Numerical Format Correspondence Diagram

The two phase currents are sampled simultaneously by means of the DSP Controller by using one channel of each ADC module per current. In this application channel 1 (ADCIN0) and channel 9 (ADCIN8) are used to sample the phase currents. Below is the code that waits for the LEM output to be converted and then transforms the conversion result into a 4.12 representation of one phase current.

```
*********************************************************
* Current sampling - AD conversions
* N.B. we will have to take only 10 bit (LSB)
*********************************************************
        ldp   #DP_PF1
        splk  #1801h,ADC_CNTL1  ;ia and ib conversion start
                                ;ADCIN0 selected for ia A/D1
                                ;ADCIN8 selected for ib A/D2
conversion
        bit   ADC_CNTL1,8
        bcnd  conversion,tc     ;wait approximatly 6us
        lacc  ADC_FIFO1,10      ;10.6 format
        ldp   #ctrl_n           ;control variable page
        sach  tmp
        lacl  tmp
        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        ;PM=11, 6 right shift after multiplication
        lt    tmp
        mpy   Kcurrent
        pac            ;
        sfr
        sfr
        sacl  ia       ;PM=11, +2 sfr= 8 right shift
        spm   0
        sub   #112     ;then we subtract a DC offset
                       ;(that should be zero, but it
                       ;isn't)

        sacl  ia       ;sampled current ia, 4.12 format
        spm   0        ;PM=00
*********************************************************
*       END Current sampling - AD conversions
*********************************************************
```

For the minimum and maximum values of the phase current, the following table shows the contents of the ADCFIFO1 register:

ADC module Input Voltage	Related current	ADCFIFO1 hexa. Value	ADCFIFO1 binary value
0 V	Imin	0000h	0000 0000 0000 0000b
5 V	Imax	FFC0h	1111 1111 1100 0000b

This current sensing and scaling module requires 45 words of ROM, 4 words of RAM and 1.98MIPS (this includes the conversion time).

3.6 Speed Sensing and Scaling

In this AC induction drive a 1000 pulse incremental encoder produces the rotor speed. The two sensor output channels (A and B) are wired directly to the QEP unit of the DSP Controller TMS320F240, which counts both edges of the pulses. The software speed resolution is thus based on 4000 increments per revolution. The QEP assigned timer counts the number of pulses, as recorded 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 achieved in this application by means of a software counter. This counter takes as input clock the PWM interrupt. 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 thus the speed can be calculated. The following scheme depicts the structure of the speed feedback generation:

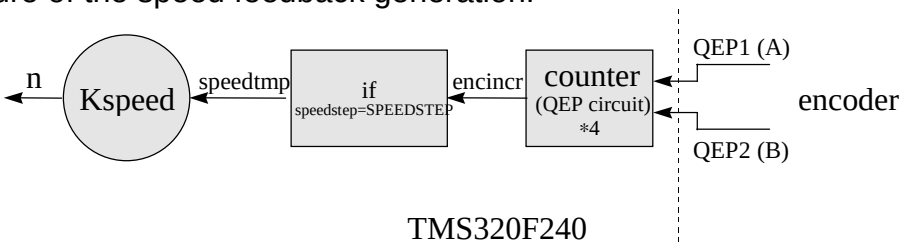

Figure 12: Speed feedback obtaining block scheme

Assuming that n_p is the number of encoder pulses in one SPEEDSTEP period when the motor turns at the nominal speed, a software constant K_{speed} should be chosen as follows:

$$01000h = K_{speed} \cdot n_p$$

to let the speed feedback be transformed into a 4.12 format, that can be used with the control software. In this application the nominal speed is 1500 rpm, SPEEDSTEP is set to 30 and then n_p can be calculated as follows:

$$n_p = \frac{1500 \cdot 4000}{60} \cdot SPEEDSTEP \cdot T = 300$$

and hence K_{speed} is given by:

$$K_{speed} = \frac{4096}{300} = 13.653 \Leftrightarrow 0da7h \qquad 8.8\text{f}$$

Note that K_{speed} is out of the 4.12 format range. The most appropriate format to handle this constant is the 8.8 format. The speed feedback in 4.12 format is then obtained from the encoder by multiplying *speedtemp* by K_{speed} . The flow chart and the code for speed sensing is presented below:

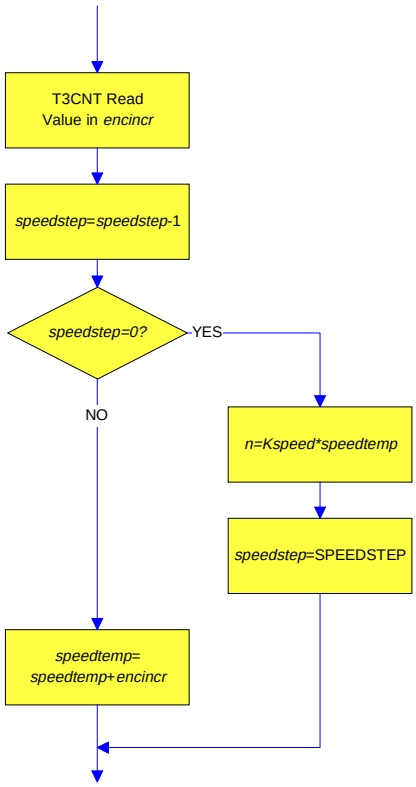

Figure 13: Speed Feedback Computation Flowchart

```
************************************
* Measured speed and control
************************************
*** encoder pulses reading
        ldp   #DP_EV
        lacc  T3CNT             ;we read the encoder pulses
        splk  #0000h,T3CNT
        ldp   #ctrl_n           ;control variable page
        sacl  encincr
*** END Encoder pulses reading

*****************************************************
* 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
*****************************************
        spm   3                 ;PM=11, 6 right shift after multiplication
        lt    speedtmp          ;multiply encoder pulses by Kspeed
                                ;(8.8 format constant)
                                ;to have the value of speed
        mpy   #Kspeed
        pac
        sfr
        sfr                     ;PM=11, +2 sfr= 8 right shift
        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
        spm   0                 ;PM=00, no shift after multiplication

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

This speed sensing and scaling module requires 28 words of ROM, 4 words of RAM and 0.244 MIPS (which includes the speed reference acquisition time).

3.7 The PI regulator

The PI (Proportional-Integral) regulators are implemented with output saturation and with integral component correction. Please refer to report [6] for any further PI structure information. The constants K_{pi}, K_i, K_{cor} (proportional, integral and integral correction components) are selected depending on the sampling period and on the motor parameters. In this application (T=100μs sampling time) the current loop constants are:

$$K_i = 0.0625 \Leftrightarrow 0100h$$

$$K_{pi} = 1 \Leftrightarrow 01000h$$

$$K_{cor} = \frac{K_i}{K_{pi}} = 0.0625 \Leftrightarrow 0100h$$

And the speed loop constants are:

$$K_{in} = 0.0129 \Leftrightarrow 0035h$$

$$K_{pin} = 4.510 \Leftrightarrow 0482Bh$$

$$K_{corn} = \frac{K_{in}}{K_{pin}} = 0.00268 \Leftrightarrow 0Bh$$

Note that all constants are in 4.12 format and the integral correction component is calculated by using the following formula:

$$K_{cor} = \frac{K_i}{K_{pi}}$$

As speed and current regulator have exactly the same software structure, only the speed regulator code is given below.

```
*****************************************************
* Speed regulator with integral component correction
*****************************************************
        lacc  n_ref
        sub   n
        sacl  epin              ;epin=n_ref-n, 4.12 format
        lacc  xin,12
        lt    epin
        mpy   Kpin
        apac
        sach  upi,4             ;upi=xin+epin*Kpin, 4.12 format
                                ;here we start to saturate
        bit   upi,0
        bcnd  upimagzeros,NTC   ;If value >0 we branch
        lacc  #Isqrefmin        ;negative saturation
        sub   upi
        bcnd  neg_sat,GT        ;if upi<ISqrefmin then branch to saturate
        lacc  upi               ;value of upi is valid
        b     limiters
neg_sat
        lacc  #Isqrefmin        ;set acc to -ve saturated value
        b     limiters

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

limiters
        sacl  iSqref            ;Store the acc as reference value
        sub   upi
        sacl  elpi              ;elpi=iSqref-upi, 4.12 format

        lt    elpi              ;if there is no saturation elpi=0
        mpy   Kcorn
        pac
        lt    epin
        mpy   Kin
        apac
        add   xin,12
        sach  xin,4   ;xin=xin+epin*Kin+elpi*Kcorn, 4.12 format

*******************************************************
* END Speed regulator with integral component correction
*******************************************************
```

where $i_{Sqrefmin}$ and $i_{Sqrefmax}$ are the speed regulator limitations. Each PI regulator module requires 44 words of ROM, 10 words of RAM and 0.44 MIPS.

3.8 Clarke and Park transformation

In the next two paragraphs, the TMS320F240 code and experimental results relevant to both the Clarke and Park transformation will be presented. The corresponding theoretical background explanations have already been handled in [6].

3.8.1 The (a,b)->(α,β) projection (Clarke transformation)

In the following code the considered constant and variables are implemented in 4.12 format.

```
*****************************************
*       Clarke transformation
*       (a,b) -> (alfa,beta)
*       iSalfa = ia
*       iSbeta = (2 * ib + ia) / sqrt(3)
*****************************************
        lacc  ia
        sacl  iSalfa            ;iSalfa 4.12 format
        add   ib
        neg
        sacl  ic

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

where SQRT3inv is the following constant:

$$SQRT3inv = \frac{1}{\sqrt{3}} = 0.577 \Leftrightarrow 093dh \qquad 4.12\ \text{f}$$

Scope pictures of the a,b,c currents (input of the Clarke module) and the α,β currents (output of this module) are presented below.

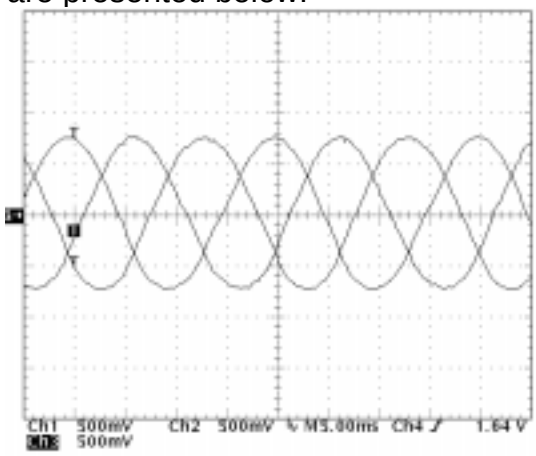

Figure 14: AC Induction Drive Phase Currents

This balanced three-phase system is shown below when transformed into the (α,β) orthogonal frame:

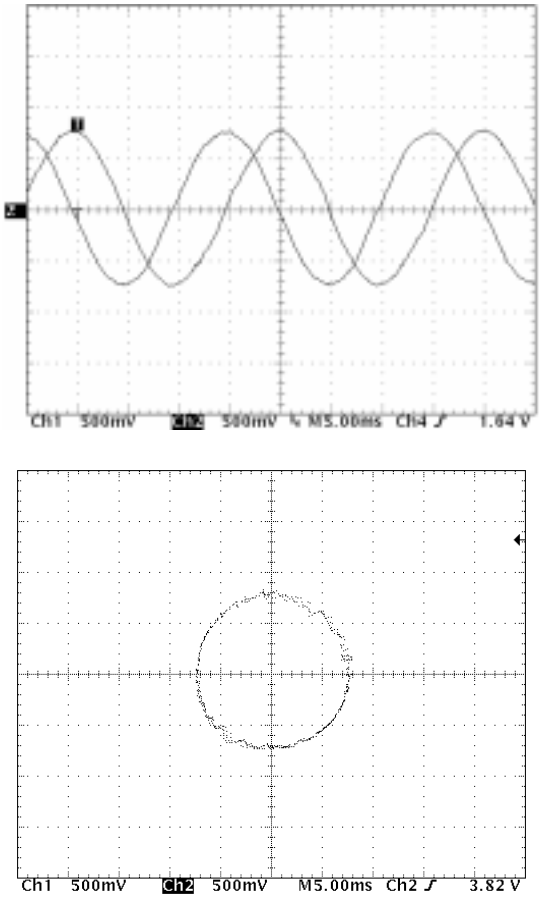

Figure 15: Output of the Clarke Transformation Module

The upper visual depicts the two coordinates Clarke transformation system. The lower visual is an X-Y representation of the transformation output where α is the X input and β is the Y input. This illustrates the resulting orthogonal system.

This Clarke transformation module requires 12 words of ROM, 6 words of RAM and 0.24 MIPS.

3.8.2 The (α,β)->(d,q) projection (Park transformation)

In the following code the constant and variables under consideration are implemented in 4.12 format. The quantity Teta_cm represents the rotor flux position calculated by the current model.

```
*********************************************
*       Park transformation
*       (alfa, beta)->(d,q)
*       iSd=iSalfa*cos(Teta_cm)+iSbeta*sin(Teta_cm)
*       iSq=-iSalfa*sin(Teta_cm)+iSbeta*cos(Teta_cm)
*********************************************
        lt    iSbeta
        mpy   sinTeta_cm
        lta   iSalfa
        mpy   cosTeta_cm
        mpya  sinTeta_cm
        sach  iSd,4             ;iSd 4.12 format
```

```
        lacc  #0
        lt    iSbeta
        mpys  cosTeta_cm
        apac
        sach  iSq,4             ;iSq 4.12 format
*********************************
* END Park transformation
*********************************
```

SinTeta_cm and *cosTeta_cm* indicate respectively the *Teta_cm* sine and cosine values. The modalities required to determine these values are explained later in this document.

This Park transformation module requires 12 words of ROM, 6 words of RAM and 0.24 MIPS.

3.9 The current model

This chapter represents the core module of the Field Orientated Controlled AC induction drive. This module takes as input i_{Sd} , i_{Sq} plus the rotor electrical speed. In addition to the two essential equations, the numerical considerations, code and experimental results will also be discussed.

3.9.1 Theoretical background

The current model [1][2][3][5] consists of implementing the following two equations of the motor in *d,q* reference frame:

$$i_{dS} = T_R \frac{di_{mR}}{dt} + i_{mR}$$

$$f_S = \frac{1}{\omega_b}\frac{d\theta}{dt} = n + \frac{i_{qS}}{T_R i_{mR} \omega_b}$$

where θ is the rotor flux position, i_{mR} the magnetizing current, and where

$$T_R = \frac{L_R}{R_R}$$

is the rotor time constant. Knowledge of this constant is critical to the correct functioning of the current model as it is this system that outputs the rotor flux speed that will be integrated to get the rotor flux position. Assuming that $i_{qS_{k+1}} \approx i_{qS_k}$ the above equations can be **discretized** as follows:

$$i_{mR_{k+1}} = i_{mR_k} + \frac{T}{T_R}(i_{dS_k} - i_{mR_k})$$

$$f_{S_{k+1}} = n_{k+1} + \frac{1}{T_R \omega_b}\frac{i_{qS_k}}{i_{mR_{k+1}}}$$

Implementation of the software for these equations is handled in the following chapters.

3.9.2 Numerical consideration

Let the two above equation constants $\frac{T}{T_R}$ and $\frac{1}{T_R \omega_b}$ be renamed respectively K_t and K_R . In this application their values are:

$$K_R = \frac{T}{T_R} = \frac{100 \cdot 10^{-6}}{30.195 \cdot 10^{-3}} = 3.3117 \cdot 10^{-3} \Leftrightarrow 0eh \quad 4.12\text{ f}$$

$$K_t = \frac{1}{T_R \omega_b} = \frac{1}{30.195 \cdot 10^{-3} \cdot 314.15} = 105.42 \cdot 10^{-3} \Leftrightarrow 01b0h \quad 4.12\text{ f}$$

Once the rotor flux speed (f_S) has been calculated, the necessary rotor flux position (θ_{cm}) is computed by the integration formula:

$$\theta_{cm_{k+1}} = \theta_{cm_k} + \omega_b f_{S_k} T$$

As the rotor flux position range is [0;2π], 16 bit integer values have been used to achieve the maximum resolution. The following chart demonstrates the relationship between the flux position and its numerical representation:

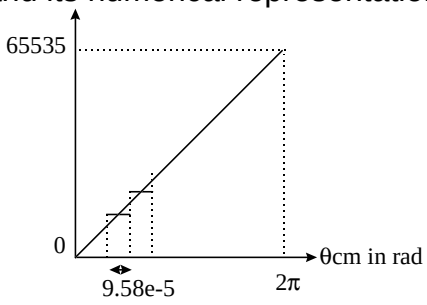

Figure 16: Link between Rotor Flux Position and its Numerical Representation

In the above equation, let $\omega_b f_S T$ be called θincr. This variable is the angle variation within one sample period. At nominal operation (in other words when f_S=1, mechanical speed is 1500rpm) θincr is thus equal to 0.031415rad. In one mechanical revolution performed at nominal speed there are $\frac{2\pi}{0.031415} \approx 200$ increments of the rotor flux position. Let K be defined as the constant, which converts the (0;2π) range into the (0;65535) range. K is calculated as follows:

$$K = \frac{65536}{200} = 327.68 \Leftrightarrow 0148h$$

With the help of this constant, the rotor flux position computation and its formatting becomes

$$\theta_{cm\,k+1} = \theta_{cm_k} + K f_{S_k}$$

The θ_{cm_k} variable is thus represented as a 16-bit integer value. This position is used in the transformation modules as the entry point in the sine look-up table.

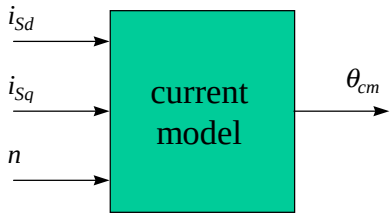

Figure 17: Input and output for the current model block

In conclusion, the current model is a block, as depicted above, with an input variable i_{dS}, i_{qS}, n (represented in 4.12 format) and the rotor flux position θ_{mc} (represented as a 16 bit integer value) as output.

3.9.3 Code and experimental results

The code for the current model is the following:

```
*********************************************
* Current Model
*********************************************
        lacc  iSd
        sub   i_mr
        sacl  tmp
        lt    tmp
        mpy   #Kr
        pac
        sach  tmp,4
        lacc  tmp
        add   i_mr
        sacl  i_mr              ;i_mr=i_mr+Kr*(iSd-i_mr), 4.12 f
        bcnd  i_mrnotzero,NEQ
        lacc  #0
        sacl  tmp               ;if i_mr=0 then tmp=iSq/i_mr=0
        b     i_mrzero
i_mrnotzero
*** division (iSq/i_mr)
        lacc  i_mr
        bcnd  i_mrzero,EQ
        sacl  tmp1
        lacc  iSq
        abs
        sacl  tmp
        lacc  tmp,12
        rpt   #15
        subc  tmp1
        sacl  tmp               ;tmp=iSq/i_mr
        lacc  iSq
        bcnd  iSqpos,GT
        lacc  tmp
        neg
        sacl  tmp               ;tmp=iSq/i_mr, 4.12 format
iSqpos
i_mrzero
*** END division ***
        lt    tmp
        mpy   #Kt
        pac
        sach  tmp,4             ;slip frequency, 4.12 format
        lacc  tmp               ;load tmp in low ACC
        add   n
        sacl  fs                ;rotor flux speed, 4.12 format,
```

```
                                ;fs=n+Kt*(iSq/i_mr)
*** rotor flux position calculation ***
        lacc  fs
        abs
        sacl  tmp
        lt    tmp
        mpy   #K
        pac
        sach  tetaincr,4
        bit   fs,0
        bcnd  fs_neg,TC
        lacl  tetaincr
        adds  Teta_cm
        sacl  Teta_cm
        b     fs_pos
fs_neg
        lacl  Teta_cm
        subs  tetaincr
        sacl  Teta_cm
                ;Teta_cm=Teta_cm+K*fs=Teta_cm+tetaincr
                ;(0;360)<->(0;65535)
fs_pos
        rpt   #3
        sfr
        sacl  Teta_cm1          ;(0;360)<->(0;4096), this variable
                                ;is used only for the visualization
*********************************************
*       END Current Model
*********************************************
```

This current model module requires 62 words of ROM, 10 words of RAM and 0.88 MIPS.

The scope picture below depicts, from top to bottom, the computed rotor flux position, the flux component and the torque component in steady state operation.

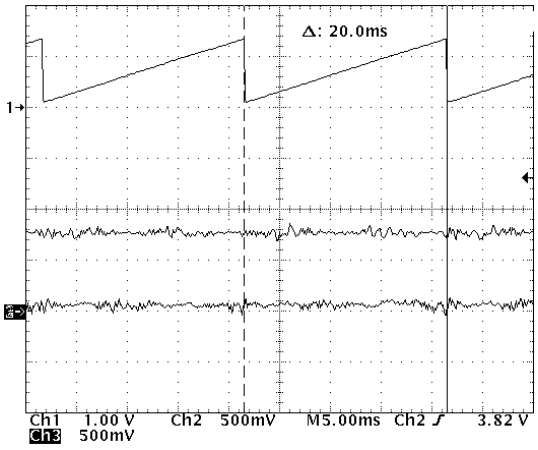

Figure 18: Rotor Flux Position, Flux and Torque Components

Note that this scope picture has been stored when the motor is running at nominal speed without any load. This makes the slip equal to zero, thus leading to the 20ms period of the rotor flux position. This also makes the torque component roughly equal to zero.

3.10 Generation of sine and cosine values

In order to generate sine and cosine values, a sine 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) therefore 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 $\sin\theta_{cm}$ value represented in 4.12 format. The following figure shows the *Teta_cm*, the *Index* and the sine look-up table.

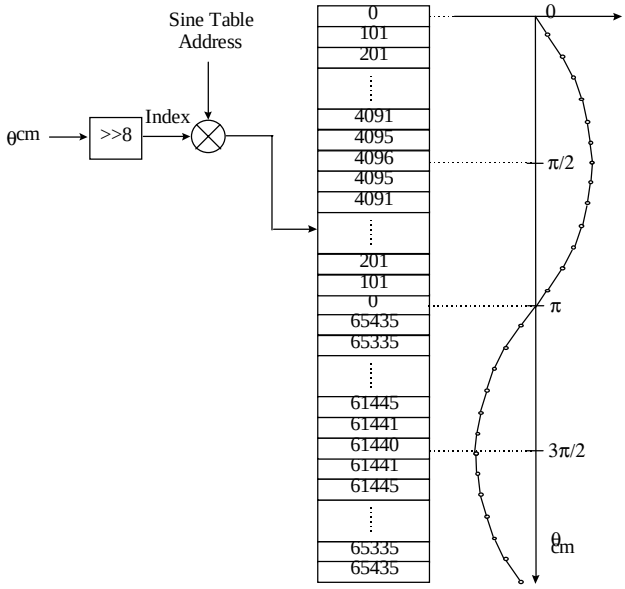

Figure 19: Sinθ$_{cm}$ Calculation using the Sine Look-up Table

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:

```
*****************************************
* sinTeta_cm, cosTeta_cm calculation
*****************************************
        mar   *,ar5
        lt    Teta_cm           ;current model rotor flux position
        mpyu  SR8BIT
        pac
        sach  Index
        lacl  Index
        and   #0ffh
        add   #sintab
        sacl  tmp
        lar   ar5,tmp
        lacl  *
        sacl  sinTeta_cm        ;sine Teta_cm value, 4.12 format
        lacl  Index             ;The same for Cos ...
                                ;cos(teta)=sin(teta+90ø)
        add   #40h              ;90ø = 40h elements of the table
        and   #0ffh
        add   #sintab
        sacl  tmp
        lar   ar5,tmp
```

```
        lacc  *
        sacl  cosTeta_cm        ;cosine Teta_cm value, 4.12 format
*****************************************
* END sinTeta_cm, cosTeta_cm calculation
*****************************************
```

This Sine and Cosine module requires 24 words of ROM, 6 words of RAM and 0.33 MIPS.

3.11 The Field Weakening

In certain circumstances, it is possible to extend the control speed range beyond the nominal speed. This chapter explains one possible process to follow in order to achieve such a speed range extension.

3.11.1 Field Weakening Principles

The aim of this application was to reach several times the nominal speed. The following theoretical background will show that it is possible to reach up to four times the nominal speed under certain conditions. 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). In this operating range the flux is maintained constant and equal to the nominal flux. Given that mechanical power is proportional to torque time speed and that its nominal value has been reached when speed is equal to 1500rpm (nominal value), the torque production must be reduced if a speed greater than 1500rpm is desired. This is shown in the following chart:

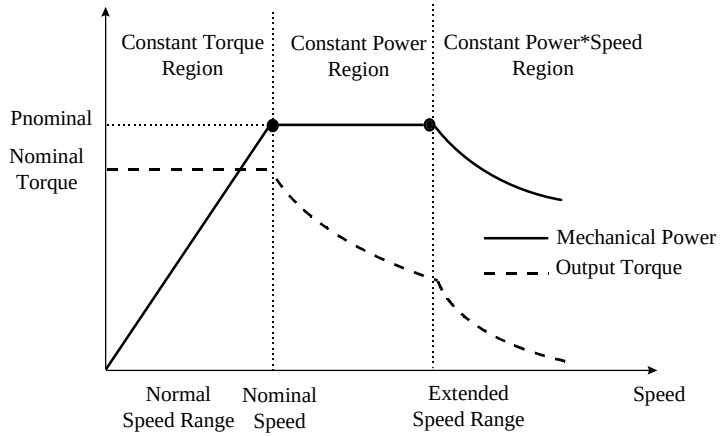

Figure 20: Field weakening Real Operation

Note the three different zones. In the constant power region the **nominal** torque production behaves like the inverse function of the speed, thereby enabling constant power production ($P=M\omega$). In the constant Power*Speed region the **nominal** torque production behaves like the inverse function of the squared speed. To explain this brake between the last two zones, the **maximal** torque function in the steady state operation must be studied here.

According to [1][11] in the steady state operation, the **maximum** torque can be calculated approximately by using the following formula:

$$M_{\max} \approx \frac{3z_p}{2\omega^2} * \frac{V^2}{(L_{\sigma S} + L_{\sigma R})} = \frac{3z_p}{2(2\pi f_S)^2} * \frac{V^2}{(L_{\sigma S} + L_{\sigma R})}$$

where $L_{\sigma S}$ and $L_{\sigma R}$ are respectively the stator and rotor leakage inductance and z_p is the number of pole pairs. In the first zone, the **maximum** torque function is equal to a constant 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, thereby causing the **maximum** torque function to behave as the inverse function of squared speed. This results in the following picture

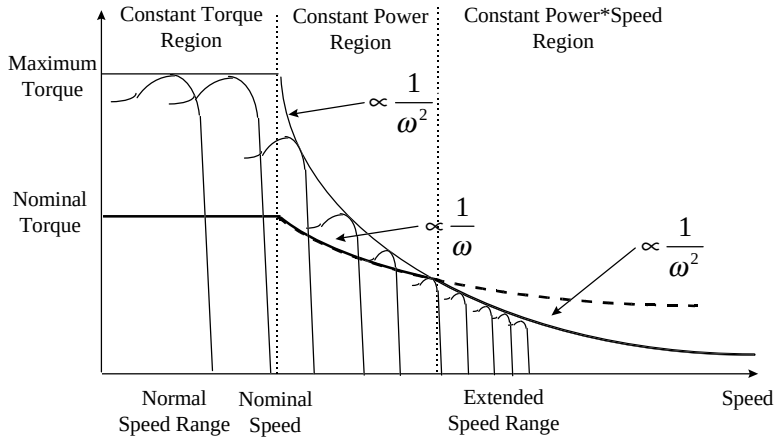

Figure 21: Maximum and Nominal Torque vs Speed

Note that the **nominal** torque curve crosses the **maximum** torque curve. This crossover point is the brake point delineating 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 (thereby making **nominal** torque smaller than the **maximum** torque) until the brake point. Once this point has been crossed, the **nominal** torque is rendered equal to the **maximum** torque, forcing the power function to behave like the inverse function of speed. With help of the formula given above and the motor parameters, it is possible to predict the crossing point.

In this application the crossing point occurs when speed is equal to 1.7 times the nominal speed (1.8*1500=2700rpm). A much bigger ratio can be achieved by simply controlling a motor with a higher maximum/nominal torque ratio, thus shifting the crossing point. The experimental measurements presented below confirm the computed crossing point.

3.11.2 Field Weakening Constraints

The drive constraints for the extended speed range are, firstly, the phase voltages and, secondly, the phase currents. Given that the phase voltage references increase with speed and that their value can not exceed the nominal value, the flux component must then be reduced to a value which 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 maintains the phase currents at a level no greater than their nominal value. The maximum resistive torque decreases then as a function of speed.

In the following scheme, both the maximum phase voltage and the flux references are shown for normal and extended speed range.

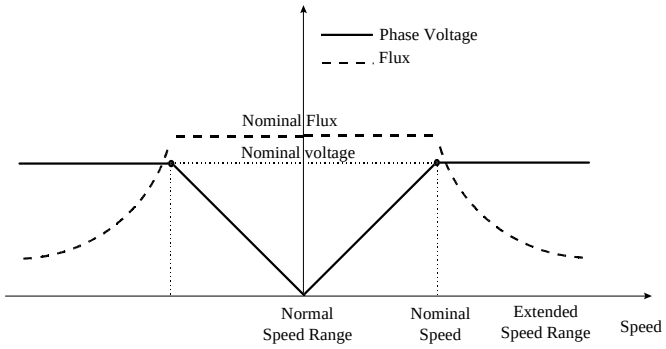

Figure 22: Field Weakening Voltage Constraints

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 to the drive. This assumes that the resulting overheating of the drive can be dissipated before performing another transient operation.

3.11.3 TMS320F240 Field Weakening Implementation

As far as the software implementation is concerned, the field-weakening module takes as input the 4.12 format speed reference and gives as output the flux reference (proportional to i_{Sdref}).

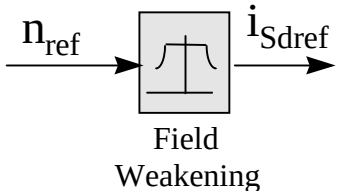

Figure 23: Field Weakening Block Diagram

The field weakening implementation requires the following steps to be performed: some motor operating points measurements, one off-line polynomial interpolation and one polynomial implementation.

The normal speed range flux reference has been set so that the phase voltage achieved is equal to the nominal value when the motor is running at nominal speed without load (slip is thus almost equal to zero and phase current is only magnetizing

current). In order to protect the drive, this drive has been developed using only 90% of the nominal voltage. There would be no problem in performing the same process with 100% of the nominal phase voltage. This technique leads to a flux reference equal to 0.6pu at nominal speed. Above the nominal speed, the following table gives the measured flux references at different speeds keeping the phase voltage at 0.9pu, up to four times the nominal speed.

nref (pu)	*Idsref (pu)*	*nref (pu)*	*Idsref (pu)*
1.1	0.52	2.6	0.200
1.2	0.47	2.7	0.195
1.3	0.42	2.8	0.190
1.4	0.39	2.9	0.188
1.5	0.36	3.0	0.185
1.6	0.33	3.1	0.182
1.7	0.31	3.2	0.179
1.8	0.29	3.3	0.175
1.9	0.27	3.4	0.172
2.0	0.26	3.5	0.170
2.1	0.25	3.6	0.168
2.2	0.23	3.7	0.166
2.3	0.22	3.8	0.165
2.4	0.21	3.9	0.164
2.5	0.21	4.0	0.163

In order to get a continuous field weakening, all along the extended speed, an off-line interpolation of these measured points has been achieved by means of the MATLAB *polyfit* and *polyval* functions. As a compromise between interpolation correctness and software optimization, the third order polynomial interpolation appeared to be the most appropriate solution. The MATLAB output polynomial is

$$i_{sdref} = -0.0195 * n_{ref}{}^{3} + 0.2196 * n_{ref}{}^{2} - 0.8158 * n_{ref} + 1.17$$

As the speed reference pu value reaches four and since this value needs to be raised to the third power, the 8.8 format has been selected to implement this field weakening function. The above polynomial coefficients become, in 8.8 format:

$$i_{sdref} = -5 * n_{ref}{}^{3} + 56 * n_{ref}{}^{2} - 209 * n_{ref} + 300$$

Note that the output flux reference (i_{Sdref}) needs to be in 4.12 format. Below the reader can find the MATLAB figure, representing the measured points, the MATLAB interpolated points and the resulting 8.8 implementation points.

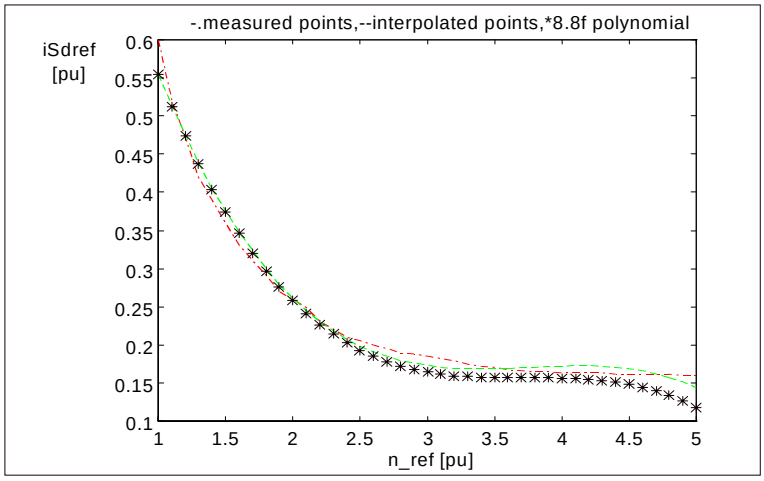

Figure 24: Matlab Interpolation Results and Numerical Implementation Result

Once this polynomial has been implemented it is possible to elicit the torque and power characteristics of this drive. These points are measured by first running the unloaded motor at the desired speed (in the extended speed range). The resistive torque is then increased until the phase currents reach their nominal value or the system becomes unstable. At this point (motor runs at desired speed under nominal voltage) mechanical power, produced torque and running speed are stored. By using the MATLAB *plot* function it is possible to produce the two weakening characteristics shown in the experimental field below.

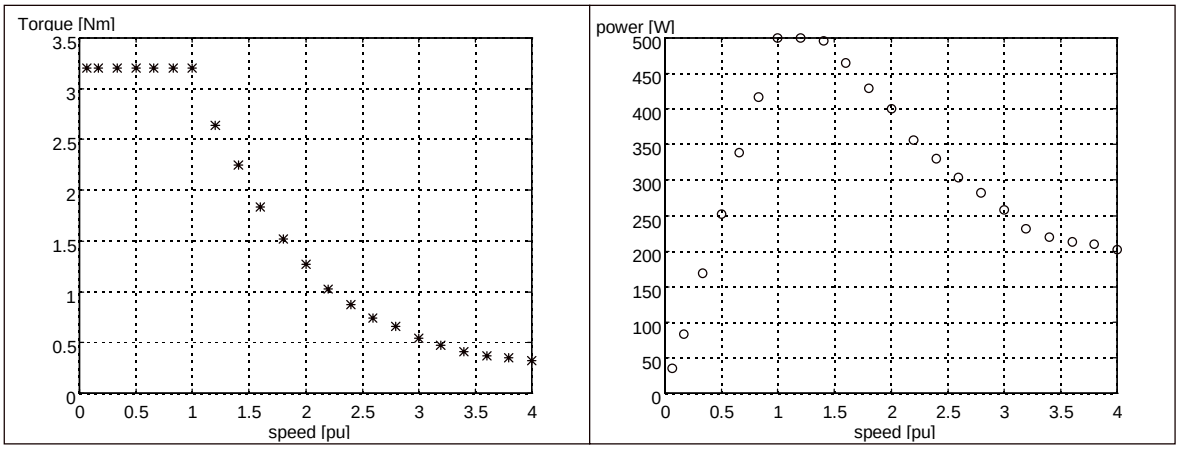

Figure 25: Experimental Torque & Power Charact. in the Extended Speed Range

The first picture shows that nominal torque can be achieved all along the normal speed range. This result is one of the most interesting advantages of the Field Orientated Control.

Looking at the power versus speed chart, note the three different regions. The experimental brake point has been measured at 1.5 times the nominal speed; given the uncertainty on the motor leakage inductance, this result confirms the theoretical model. Of course it is possible to extend the constant power region up to three times

the nominal value using exactly the same control software, simply by choosing an induction motor with a higher $\frac{M_{\text{maximum}}}{M_{\text{nominal}}}$ ratio.

```
*************************************************
*       Field-weakening function
*       Input:n_ref, output iSdref  4.12 format
*************************************************
        spm   2       ;PM=10, four left shift after multiplication
        lacc  n_ref
        abs           ;we consider absolute value of speed reference
        rpt   #3
        sfr
        sacl  n_ref8_8;speed reference 8.8f
        sub   #100h
        bcnd  noFieldWeakening,LEQ
        lacc  p0,12
        lt    n_ref8_8
        mpy   p1
        apac
        sach  tmp,4   ;tmp=p0+p1*n_ref
        sqra  n_ref8_8
        pac
        sach  tmp1,4
        lacc  tmp,12
        lt    tmp1    ;tmp1=n_ref^2
        mpy   p2
        apac
        sach  tmp,4   ;tmp=p0+p1*n_ref+p2*(n_ref^2)
        lt    tmp1
        mpy   n_ref8_8
        pac
        sach  tmp1,4  ;tmp1=n_ref^3
        lacc  tmp,12
        lt    tmp1
        mpy   p3
        apac
        sach  tmp,4   ;tmp=p0+p1*n_ref+p2*(n_ref^2)+p3*(n_ref^3)
        lacc  tmp,4   ;iSdref 8.8 f
        sacl  iSdref  ;iSdref 4.12 f with Field Weakening
        b     endFW
noFieldWeakening
        lacc  #2458   ;iSdref=0.6 pu
        sacl  iSdref  ;iSdref 4.12 f without Field Weakening
endFW
        spm   0       ;PM=0
*************************************************
*       END Field Weakening function
*************************************************
```

This Field Weakening module requires 40 words of ROM, 10 words of RAM and 0.33 MIPS.

3.12 The Space Vector Modulation

The Space Vector Modulation is a highly efficient way to generate the six pulsed signals [6] necessary at the power stage. The SVM needs the reference voltages $v_{S\alpha ref}$, $v_{S\beta ref}$ as input, the DC bus voltage as parameter and gives the three PWM patterns as output. These values are once again expressed in pu quantities, so that

they may be implemented in 4.12 format. The conversions of these inputs into the required numerical format are given below.

$$v_{DC} = \frac{V_{DC}}{V_b} = \frac{310}{180} = 1.722 \Leftrightarrow 1b8dh \quad 4.12\,\mathrm{f}$$

where V_{DC} is the DC bus voltage in the used inverter and v_{DC} the correspondent pu value. The software [3][4] presented below also requires the following constant definition:

$$v_{DCinvT} = \frac{T}{2v_{DC}} \Leftrightarrow \frac{PWMPRD}{v_{DC}} = \frac{1000}{1.722} = 581 \quad \Leftrightarrow 245h$$

and the following variable definition:

$$v_{ref1} = v_{S\beta ref}$$

$$v_{ref2} = \frac{1}{2}(\sqrt{3}v_{S\alpha ref} - v_{S\beta ref})$$

$$v_{ref3} = \frac{1}{2}(-\sqrt{3}v_{S\alpha ref} - v_{S\beta ref})$$

$$X = \sqrt{3}v_{DCinvT}v_{S\beta ref}$$

$$Y = \frac{\sqrt{3}}{2}v_{DCinvT}v_{S\beta ref} + \frac{3}{2}v_{DCinvT}v_{S\alpha ref}$$

$$Z = \frac{\sqrt{3}}{2}v_{DCinvT}v_{S\beta ref} - \frac{3}{2}v_{DCinvT}v_{S\alpha ref}$$

According to [6], the first step to undertake is to determine in which sector the voltage vector defined by $v_{S\alpha ref}$, $v_{S\beta ref}$ is found. The following few code lines give the sector as output:

sector determination

$IF\ v_{ref1} > 0 \quad THEN\ A{:=}1, \quad ELSE\ A{:=}0$

$IF\ v_{ref2} > 0 \quad THEN\ B{:=}1, \quad ELSE\ B{:=}0$

$IF\ v_{ref3} > 0 \quad THEN\ C{:=}1, \quad ELSE\ C{:=}0$

$\mathrm{sec\,tor}{:=}\ A{+}2B{+}4C$

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

CASE sector *OF*

1 $t_1 = Z$ $t_2 = Y$

2 $t_1 = Y$ $t_2 = -X$

3 $t_1 = -Z$ $t_2 = X$

4 $t_1 = -X$ $t_2 = Z$

5 $t_1 = X$ $t_2 = -Y$

6 $t_1 = -Y$ $t_2 = -Z$

end times calculation

Saturations

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

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

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

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

Figure 26: Table Assigning the Right Duty Cycle to the Right Motor Phase

The following picture shows an example of one vector which would be in sector 3 according to [6] notations.

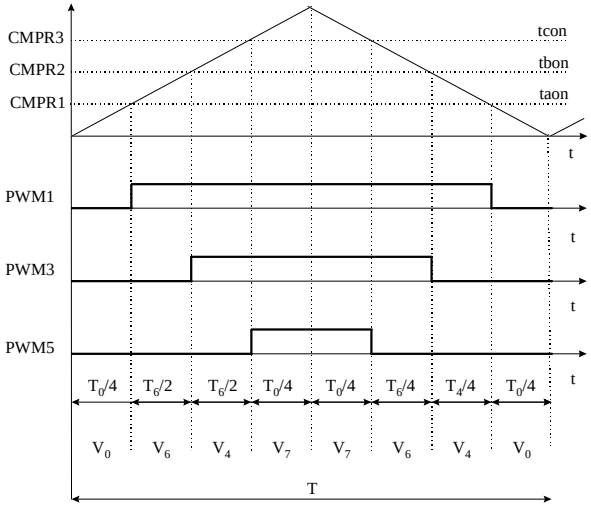

Figure 27: Sector 3 PWM Patterns and Duty Cycles

According to [6] the maximum phase voltage that can be used out of this inverter is:

$$\frac{V_{DC}}{\sqrt{3}} = \frac{310}{\sqrt{3}} \cong 179V$$

Given that the base voltage of the motor used in this application is equal to 180V, the above information shows the very high efficiency of the power conversion when the maximum available voltage is used.

This Space Vector Modulation module requires 215 words of ROM, 17 words of RAM and 1.69 MIPS.

3.13 Experimental Results

This chapter handles the results of the different drive operations. The motor has been mounted on to a test bench with adjustable resistive torque. The test results are split into two categories: operations where speed is smaller or equal to the nominal value and operations where speed is higher than the nominal value.

As explained in a previous chapter, the flux reference (i_{Sdref}) in the normal speed range has been set to 0.6 pu. Knowing that the pu phase current magnitude (i) must be smaller than or equal to one and that $i = \sqrt{{i_{Sdref}}^2 + {i_{Sqref}}^2}$, then i_{Sqref} may not be higher than 0.8 pu. This torque reference limitation is integrated into the control software using the $i_{Sqrefmax}$ constant, that is set to 0ccdh (4.12 format). The following scope pictures show, on one hand, the steady state operation at 1500rpm under nominal load and, on the other hand, the transient operation from 100rpm to 1500rpm under nominal load.

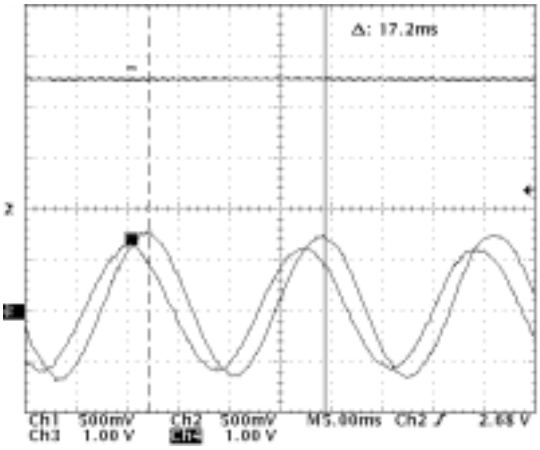

Figure 28: Steady State Operation under Nominal Conditions

In order to produce this steady state picture the motor has first been accelerated up to the nominal speed without any resistive torque and, in a second step, it has been loaded with the braking torque nominal value. Knowing that 1.25V represents a one in pu model, then this picture shows that the motor runs at nominal speed (achieved speed superimposed with speed reference) under nominal phase voltage with nominal phase current.

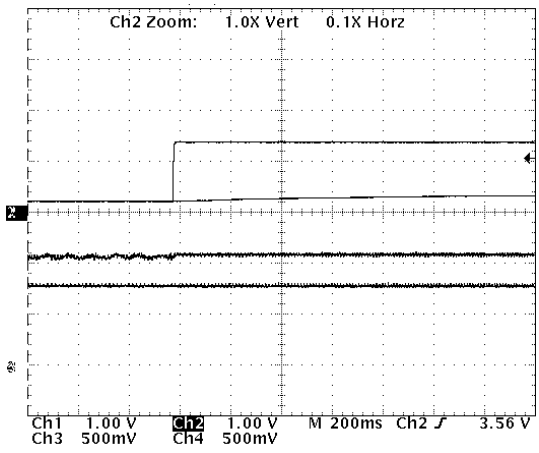

Figure 29: Transient Operation under Nominal Torque / Torque Limitation set to 0.8

This transient picture shows that the nominal operating point can not be achieved if the braking torque is maintained constant and equal to its nominal value. This is due to the limitations of the torque component In fact it has first been set so that the maximum phase current is equal to the nominal value, but to achieve the desired operating point with a quick mechanical time constant, the motor needs to get transient currents higher than nominal current. The following scope picture shows that simply increasing the torque component limitation can solve this transient trouble. In this case the $i_{Sqrefmax}$ has been set to 1.2 instead of 0.8.

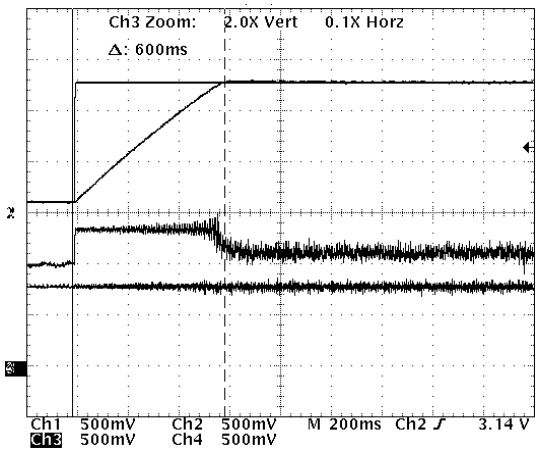

Figure 30: Transient Operation under Nominal Torque / Torque Limit. set to 1.2

This shows that nominal operating point can be reached by this field orientated control structure with a maximum transient phase current of 1.2 times the nominal current, thereby minimizing the problems associated with drive overheating. Furthermore, the transient duration under nominal load is very short as it is equal to 0.6 sec**,** which confirms the predicted excellent dynamic behaviour of the field-orientated control.

The next scope picture shows the transient behaviour in the field weakening area. The torque component limitation is equal to one. The speed reference changes from 2000 to 3000 rpm. The load torque is set to the maximum achievable value, 3000rpm. Note that steady state behaviour has already been discussed in the chapter dedicated to field weakening.

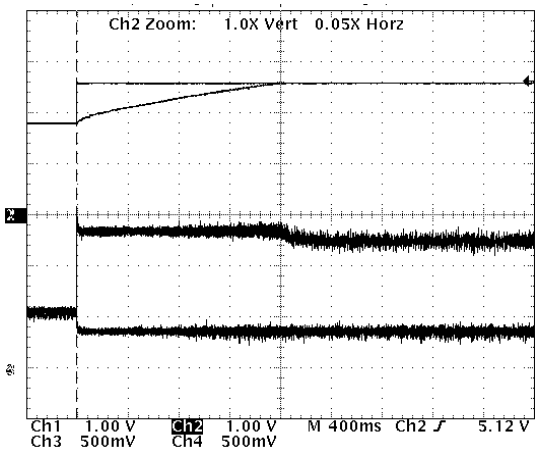

Figure 31: Transient Operation in the Extend. Speed Range / Torque Limit. set to 1

Note that during the field-weakening transient, the flux reference (i_{Sdref}) decreases and hence the torque component may be increased under the constraint $\sqrt{{i_{Sdref}}^2 + {i_{Sqref}}^2} \leq 1$. The software function that would allow new torque component limitations relative to a decrease in the flux reference has not been implemented.

The following scope picture shows a speed reversion test from 1000rpm to -1000rpm under nominal load.

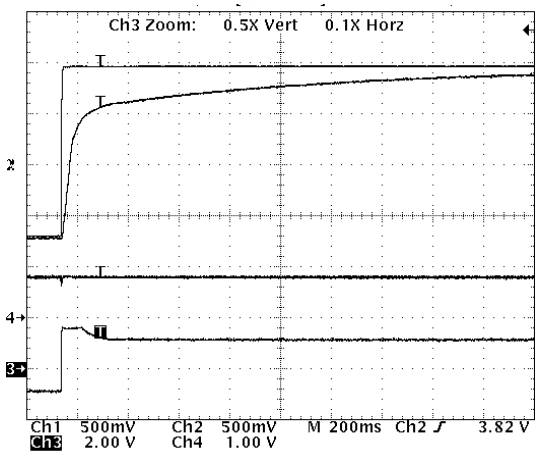

Figure 32: Speed Reversion from -1000rpm to 1000rpm under nominal load

Notice that the torque component reached its limitation at 100ms and that flux and torque components are decoupled.

3.14 The control algorithm flow chart

The flow chart of the TIMER1 underflow Interrupt Service Routine (ISR containing the complete FOC structure computation) is given below:

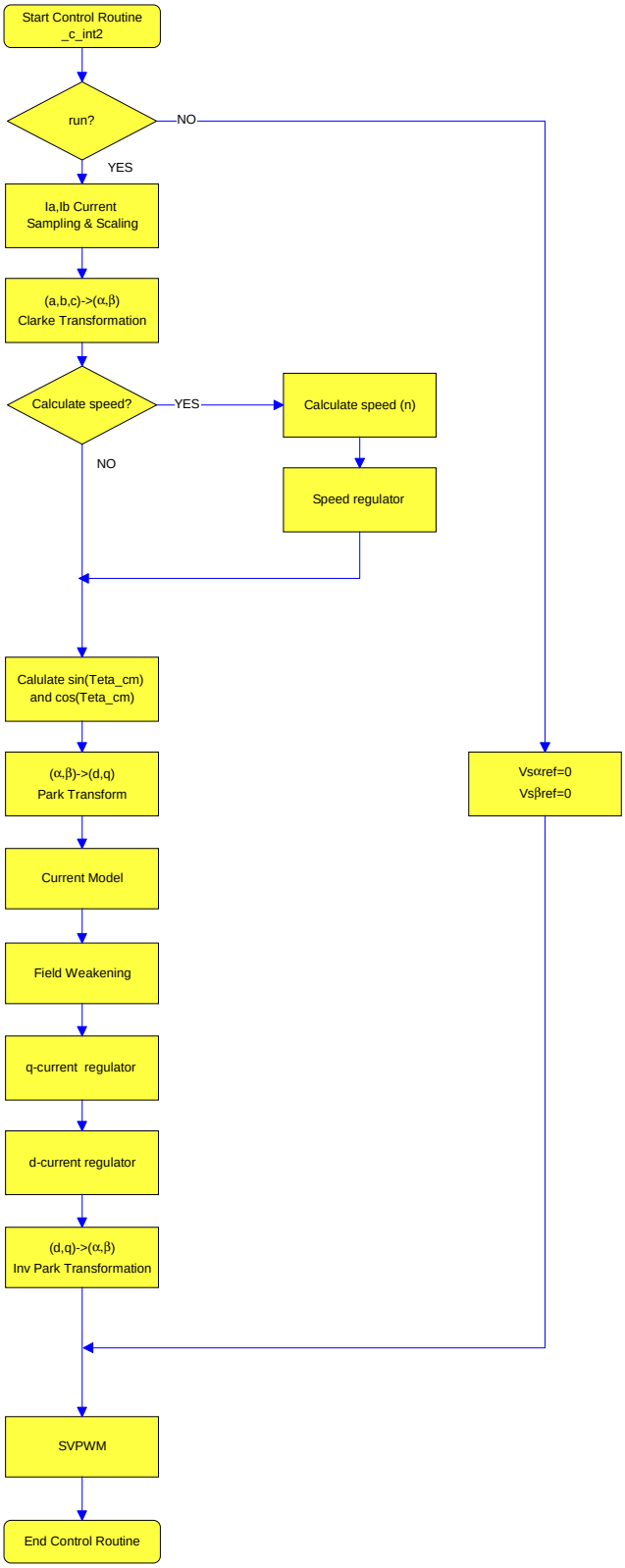

Figure 33: FOC Implementation Flowchart

4. User Interface

An exhaustive explanation of the user interface implementation is beyond the scope of this document. This chapter simply presents the screen picture that has been used as user interface to develop and improve this software. The corresponding Quick Basic program and the assembly communication software are both included for information purposes. One can be found in the appendix file, while the other is directly included in the FOC algorithm. Below is a copy of the screen picture:

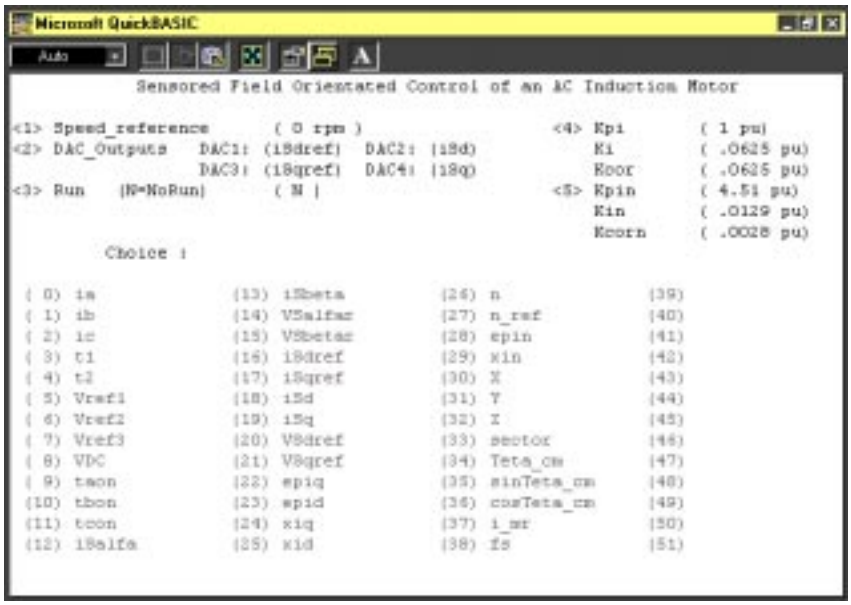

Figure 34: Communication Program. Screen picture

5. Conclusion

This document has introduced the Field Orientated Controlled AC induction drive based on the DSP Controller TMS320F240. We have demonstrated that the real time processing capability of this motor control dedicated device allows for a highly reliable and effective drive. In fact, this document has explained not only the drive reliability and efficiency, but the cost efficiency of the motor and drive, the high torque at zero speed, the speed variation capability, the extended speed range, the direct torque and flux control and the excellent dynamic behaviour. These results have been achieved using 7.7 MIPS from the 20 MIPS available and with a code size no greater than 1K word. 320 words (out of a possible 544) of data memory are enough to implement this control.

The major objective of this report was to provide the reader with the tools to enable him to develop his own AC induction drive in a very short time. To achieve this, detailed explanations, the results of various experiments, implementation tips and the background theory to these processes have all been fully explained. In fact, the modular structure of the presentation and the guidelines for correctly adapting software allow the reader to quickly grasp the different aspects of this FOC structure and be able to adapt this software to the required specification.

References

[1] F. Parasiliti, “Appunti delle lezioni di Azionamenti Elettrici: Controllo Vettoriale ad Orientamento di Campo”, Università degli Studi di L’Aquila

[2] R. Di Gabriele, F. Parasiliti, M. Tursini, “Digital Field Oriented Control for induction motors: implementation and experimental results”, Universities Power Engineering Conference (UPEC’97)

[3] Riccardo Di Gabriele, “Controllo vettoriale di velocità di un motore asincrono mediante il Filtro di Kalman Esteso”, Tesi di Laurea, Università degli Studi di L’Aquila, Anno Accademico 1996-97

[4] Roberto Petrella, “Progettazione e sviluppo di un sistema digitale basato su DSP e PLD per applicazione negli azionamenti elettrici”, Tesi di Laurea, Università degli Studi di L’Aquila, Anno Accademico 1995-96

[5] Angela Del Gobbo, “Controllo Vettoriale digitale di un motore asincrono: strategie di stima e dispositivi di calcolo a confronto”, Tesi di Laurea, Università degli Studi di L’Aquila, Anno Accademico 1994-95

[6] Texas Instruments, “Field Orientated Control of Three Phase AC-motors”, Literature number: BPRA073, December 1997

[7] A. Ometto, “Modulation Techniques”, Università degli Studi di L’Aquila

[8] J.-P. Favre, “Correction de la compsante intégrale de régulateurs digitaux en cas de limitation”, EPF- Lausanne

[9] A. Ometto, “Modulation Techniques”, Università degli Studi di L’Aquila

[10] Werner Leonard, “Control of Electrical Drives”, 2nd Completely Revised and Enlarged Edition, Springer, ISBN 3-540-59380-2

[11] D. W. Novotny and T. A. Lipo, “Vector Control and Dynamics of AC Drives”, Oxford Science Publications, ISBN 0-19-856439-2

[12] Texas Instruments, “DSP Solution for AC Induction Motor”, Literature number: BPRA043, November 1996

Appendix A - TMS320F240 FOC Software

```
*************************************************************
*       TEXAS INSTRUMENTS                                   *
*       Sensored Speed Field Orientated Control of an AC    *
*       induction motor                                     *
*************************************************************
*       ASYNCHRONOUS Motor (LAFERT)                         *
*       File Name     :         foc.ASM                     *
*       USER INTERFACE program  :     foc.BAS               *
*       ia, ib sampled currents, on line current model, FOC *
*       Author        :         Riccardo Di Gabriele        *
*************************************************************

        .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 ;
        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 ;
        .space 16*6   ;reserve 6 words in interrupt table

*********************************************************
*       Auxiliary Register used
*       ar7   pointer for context save stack
*       ar5   used in the interruption c_int2
*********************************************************

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 LAFERT, ST 61 L4 ***
***     Numeric formats: all 4.12 fixed point format twos
***     complement for negative values (4 integer & sign + 12
***     fractional) except otherwise specified
*       Currents: 1000h (4.12)= 4.1 A = Ibase=1.41*In0=1.41*2.9
*       Voltages: 1000h (4.12)= 179.6 V = Vbase=1.41*Vn0=1.41*127
*       Angles  : [0;ffffh]  = [0;360] degrees
*       Speed   : [0;1000h] (4.12) = [0;1500] rpm
***     END Numeric formats

*********************************************************
* Look-up table .includes
* N.B. it includes 256 elements
*********************************************************
        .sect "table"

sintab  .include      sine.tab  ;sine wave look-up
                                ;table for sine and
                                ;cosinewaves generation
                                ;generated by the BASIC
                                ;program "SINTAB.BAS"
                                ;4.12 format
```

```
*** END look-up table .includes

*********************************************************
* Variables and constants initializations
*********************************************************
        .data

*** current sampling constants
;ADCIN0 (ia current sampling)
;ADCIN8 (ib current sampling)
Kcurrent.word 1383h    ;8.8 format (19.51) sampled
                       ;currents normalization constant
Kr      .set  0eh      ;Kr=T/Tr=3.3117*10-3 (4.12 f)
Kt      .set  1b0h     ;Kt=1/(Tr*wBase)=105.42*10-3 (f 4.12)
K       .set  148h     ;K=65536/200, the K constant
                       ;must take the rotor flux
                       ;position from 0 to 65535 in
                       ;200 sample times

*** axis transformation constants
SQRT3inv.set  093dh    ;1/SQRT(3) 4.12 format
SQRT32  .set  0ddbh    ;SQRT(3)/2 4.12 format
SR8BIT  .word 100h     ;used to shift bits 8 right

*** PWM modulation constant
PWMPRD  .set  1000     ;PWM Period=2*1000 ->
                       ;Tc=2*1000*50ns=100us (50ns resolution)

*** PI current regulators parameters
Ki      .word 0100h    ;4.12 format=0.0625
Kpi     .word 01000h   ;4.12 format=1
Kcor    .word 0100h    ;4.12 format=0.0625

*** PI speed regulators parameters
Kin     .word 35h      ;4.12 format=0.012939453
Kpin    .word 482bh    ;4.12 format=4.510498047
Kcorn   .word 0bh      ;4.12 format=0.002685546

*** Field Weakening polynomial coefficients
p3      .word -5       ;8.8 format
p2      .word 56
p1      .word -209
p0      .word 300

*** vSqref and VdSr limitations
Vmin    .set  0ec00h   ;4.12 format=-1.25 pu
Vmax    .set  1400h    ;4.12 format=1.25 pu

*** iSqref limitations
Isqrefmin     .set    -3277     ;4.12 format=-0.8 pu
Isqrefmax     .set    3277      ;4.12 format=0.8 pu

*** Speed calculation constants
Kspeed  .set  0da7h    ;this constant is needed only
                       ;with encoder it is used to
                       ;convert encoder pulses to a speed value.
                       ;8.8 format = 13.65
SPEEDSTEP .set 30      ;speed samplig period =
                       ;current sampling period *
                       ;SPEEDSTEP

ctrl_n                 ;label control variable page
        .bss  tmp,1    ;temporary variable (to use in ISR only !!!)
        .bss  tmp1,1   ;temporary variable
        .bss  n_ref8_8,1        ;8.8 format reference speed
                                 ;for Field Weakening behavior
```

```
        .bss  option,1;virtual menu option number
        .bss  daout,1  ;address of the variable to
                       ;send to the DACs
        .bss  daouttmp,1        ;value to send to the DACs

*** DAC displaying table starts here
        .bss  ia,1     ;phase current ia
        .bss  ib,1     ;phase current ib
        .bss  ic,1     ;phase current ic
        .bss  t1,1     ;SVPWM T1 (see SV PWM references for details)
        .bss  t2,1     ;SVPWM T2 (see SV PWM references for details)
        .bss  Vref1,1  ;variable for sector calculation
        .bss  Vref2,1  ;variable for sector calculation
        .bss  Vref3,1  ;variable 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  iSalfa,1;alfa-axis current
        .bss  iSbeta,1;beta-axis current
        .bss  vSal_ref,1        ;alfa-axis reference voltage
        .bss  vSbe_ref,1        ;beta-axis reference voltage
        .bss  iSdref,1;d-axis reference current
        .bss  iSqref,1;q-axis reference current
        .bss  iSd,1    ;d-axis current
        .bss  iSq,1    ;q-axis current
        .bss  vSdref,1;d-axis reference voltage
        .bss  vSqref,1;q-axis reference voltage
        .bss  epiq,1   ;q-axis current regulator error
        .bss  epid,1   ;d-axis current regulator error
        .bss  xiq,1    ;q-axis current regulator integral component
        .bss  xid,1    ;d-axis current regulator integral component
        .bss  n,1      ;speed
        .bss  n_ref,1  ;speed reference
        .bss  epin,1   ;speed error (used in speed regulator)
        .bss  xin,1    ;speed regulator integral component
        .bss  X,1      ;SVPWM variable
        .bss  Y,1      ;SVPWM variable
        .bss  Z,1      ;SVPWM variable
        .bss  sector,1;SVPWM sector
        .bss  Teta_cm1,1        ;rotor flux position with current
                                ;model used only in the
                                ;communication program
        .bss  sinTeta_cm,1      ;sine rotor flux position with
                                ;current model, 4.12 f
        .bss  cosTeta_cm,1      ;cosine rotor flux position with
              ;current model, 4.12 f
        .bss  i_mr,1   ;magnetizing current (used only in
                       ;the current model), 4.12 f
        .bss  fs,1     ;rotor flux speed 4.12 f
*** END DAC displaying table
        .bss  run,1    ;initialization flag
        .bss  Teta_cm,1         ;real rotor flux position, output
              ;of the current model
        .bss  serialtmp,1       ;serial communication temporary
              ;variable
        .bss  da1,1    ;DAC displaying table offset for DAC1
        .bss  da2,1    ;DAC displaying table offset for DAC2
        .bss  da3,1    ;DAC displaying table offset for DAC3
        .bss  da4,1    ;DAC displaying table offset for DAC4
        .bss  VDCinvT,1         ;VDCinv*(T/2) (used in SVPWM)
        .bss  tetaincr,1        ;variable used in current model
        .bss  Index,1  ;pointer used to access sine look-up table
* PI regulators variable
        .bss  upi,1    ;PI regulators (current and speed) output
        .bss  elpi,1   ;PI regulators (current and speed)
                       ;limitation error
```

```
        .bss  encincr,1         ;encoder pulses increment between
                                ;two consecutive Sampling periods
        .bss  speedtmp,1        ;used to accumulate encoder pulses
                                ;increments (to calculate the
                                ;speed each speed sampling period)
        .bss  speedstep,1       ;sampling periods down counter
                                ;used to define speed sampling
                                ;period

*** END Variables and constants initializations

        .text         ;link in "text section"


*********************************************************** *
* _c_int2 ISR                                              *
* synchronization of the control algorithm with the PWM    *
* underflow interrupt                                      *
*********************************************************** *
_c_int2:
************************
* Context Saving
************************
        larp  ar7     ;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
**********************************
* Control ISR
* Description: Control Algorithm ISR
* Last Update:17 november 1997
**********************************
**********************************
* initialization phase
**********************************
        ldp   #ctrl_n ;control variable page
        lacc  run
        bcnd  noinit,NEQ
        lacc  #0
        sacl  vSal_ref
        sacl  vSbe_ref
        b     init
**********************************
* END initialization phase
**********************************

noinit
************************************************************
* Current sampling - AD conversions
* N.B. we will have to take only 10 bit (LSB)
************************************************************
        ldp   #DP_PF1
        splk  #1801h,ADC_CNTL1  ;ia and ib conversion start
                                ;ADCIN0 selected for ia A/D1
                                ;ADCIN8 selected for ib A/D2
conversion
        bit   ADC_CNTL1,8
        bcnd  conversion,tc     ;wait approximatly 6us
        lacc  ADC_FIFO1,10      ;10.6 format
        ldp   #ctrl_n           ;control variable page
        sach  tmp
        lacl  tmp
        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       ;PM=11, 6 right shift after multiplication
        lt    tmp
        mpy   Kcurrent
        pac           ;
        sfr
        sfr
        sacl  ia      ;PM=11, +2 sfr= 8 right shift
        spm   0
        sub   #112    ;then we subtract a DC offset
                      ;(that should be zero, but it isn't)

        sacl  ia      ;sampled current ia, 4.12 format
        ldp   #DP_PF1
        lacc  ADC_FIFO2,10
        ldp   #ctrl_n ;control variable page
        sach  tmp
        lacl  tmp
        and   #3ffh
        sub   #512
        sacl  tmp
        spm   3
        lt    tmp
        mpy   Kcurrent
        pac
        sfr
        sfr           ;PM=11, +2 sfr= 8 right shift
        add   #-80    ;then we subtract a DC offset
                      ;(that should be zero, but it isn't)
        sacl  ib
        spm   0       ;PM=00
*********************************************************
*       END Current sampling - AD conversions
*********************************************************
*****************************************
*       Clarke transformation
*       (a,b) -> (alfa,beta)
*       iSalfa = ia
*       iSbeta = (2 * ib + ia) / sqrt(3)
*****************************************
        lacc  ia
        sacl  iSalfa  ;iSalfa 4.12 format
        add   ib
        neg
        sacl  ic

        lacc  ib,1    ;iSbeta = (2 * ib + ia) / sqrt(3)
        add   ia
        sacl  tmp
        lt    tmp
        mpy   #SQRT3inv         ;SQRT3inv = (1 / sqrt(3)) = 093dh
                                ;4.12 format = 0.577350269
        pac
        sach  iSbeta,4;iSbeta 4.12 format
*********************************
* END Clarke transformation
*********************************
*************************************
* Measured speed and control
*************************************
*** encoder pulses reading
        ldp   #DP_EV
        lacc  T3CNT   ;we read the encoder pulses
        splk  #0000h,T3CNT
        ldp   #ctrl_n ;control variable page
```

```
        sacl  encincr
*** END Encoder pulses reading

*****************************************************
* 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
*******************************************
        spm   3         ;PM=11, 6 right shift after multiplication
        lt    speedtmp  ;multiply encoder pulses by Kspeed
                        ;(8.8 format constant)
                        ;to have the value of speed
        mpy   #Kspeed
        pac
        sfr
        sfr             ;PM=11, +2 sfr= 8 right shift
        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
        spm   0                 ;PM=00, no shift after multiplication
*******************************************
* END Speed calculation from encoder pulses
*******************************************
*****************************************************
* Speed regulator with integral component correction
*****************************************************
        lacc  n_ref
        sub   n
        sacl  epin      ;epin=n_ref-n, 4.12 format
        lacc  xin,12
        lt    epin
        mpy   Kpin
        apac
        sach  upi,4     ;upi=xin+epin*Kpin, 4.12 format
                        ;here we start to saturate
        bit   upi,0
        bcnd  upimagzeros,NTC   ;If value >0 we branch
        lacc  #Isqrefmin        ;negative saturation
        sub   upi
        bcnd  neg_sat,GT        ;if upi<ISqrefmin then branch to saturate
        lacc  upi       ;value of upi is valid
        b     limiters
neg_sat
        lacc  #Isqrefmin        ;set acc to -ve saturated value
        b     limiters

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

limiters
        sacl  iSqref            ;Store the acc as reference value
```

```
        sub   upi
        sacl  elpi    ;elpi=iSqref-upi, 4.12 format

        lt    elpi    ;if there is no saturation elpi=0
        mpy   Kcorn
        pac
        lt    epin
        mpy   Kin
        apac
        add   xin,12
        sach  xin,4   ;xin=xin+epin*Kin+elpi*Kcorn, 4.12 format
*********************************************************
* END Speed regulator with integral component correction
*********************************************************

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 control
************************************
*****************************************
* sinTeta_cm, cosTeta_cm calculation
*****************************************
        mar   *,ar5
        lt    Teta_cm ;current model rotor flux position
        mpyu  SR8BIT
        pac
        sach  Index
        lacl  Index
        and   #0ffh
        add   #sintab
        sacl  tmp
        lar   ar5,tmp
        lacl  *
        sacl  sinTeta_cm       ;sine Teta_cm value, 4.12 format

        lacl  Index            ;The same for Cos ...
                               ;cos(teta)=sin(teta+90ø)
        add   #40h             ;90ø = 40h elements of the table
        and   #0ffh
        add   #sintab
        sacl  tmp
        lar   ar5,tmp
        lacc  *
        sacl  cosTeta_cm       ;cosine Teta_cm value, 4.12 format
*****************************************
* END sinTeta_cm, cosTeta_cm calculation
*****************************************
*********************************************
*       Park transformation
*       (alfa, beta)->(d,q)
*       iSd=iSalfa*cos(Teta_cm)+iSbeta*sin(Teta_cm)
*       iSq=-iSalfa*sin(Teta_cm)+iSbeta*cos(Teta_cm)
*********************************************
        lt    iSbeta
        mpy   sinTeta_cm
        lta   iSalfa
        mpy   cosTeta_cm
        mpya  sinTeta_cm
        sach  iSd,4            ;iSd 4.12 format
        lacc  #0
        lt    iSbeta
        mpys  cosTeta_cm
        apac
```

```
        sach  iSq,4             ;iSq 4.12 format
*********************************
* END Park transformation
*********************************
*********************************************
* Current Model
*********************************************
        lacc  iSd
        sub   i_mr
        sacl  tmp
        lt    tmp
        mpy   #Kr
        pac
        sach  tmp,4
        lacc  tmp
        add   i_mr
        sacl  i_mr              ;i_mr=i_mr+Kr*(iSd-i_mr), 4.12 f
        bcnd  i_mrnotzero,NEQ
        lacc  #0
        sacl  tmp               ;if i_mr=0 then tmp=iSq/i_mr=0
        b     i_mrzero
i_mrnotzero
*** division (iSq/i_mr)
        lacc  i_mr
        bcnd  i_mrzero,EQ
        sacl  tmp1
        lacc  iSq
        abs
        sacl  tmp
        lacc  tmp,12
        rpt   #15
        subc  tmp1
        sacl  tmp               ;tmp=iSq/i_mr
        lacc  iSq
        bcnd  iSqpos,GT
        lacc  tmp
        neg
        sacl  tmp               ;tmp=iSq/i_mr, 4.12 format
iSqpos
i_mrzero
*** END division ***
        lt    tmp
        mpy   #Kt
        pac
        sach  tmp,4             ;slip frequency, 4.12 format
        lacc  tmp               ;load tmp in low ACC
        add   n
        sacl  fs                ;rotor flux speed, 4.12 format,
                                ;fs=n+Kt*(iSq/i_mr)
*** rotor flux position calculation ***
        lacc  fs
        abs
        sacl  tmp
        lt    tmp
        mpy   #K
        pac
        sach  tetaincr,4
        bit   fs,0
        bcnd  fs_neg,TC
        lacl  tetaincr
        adds  Teta_cm
        sacl  Teta_cm
        b     fs_pos
fs_neg
        lacl  Teta_cm
        subs  tetaincr
        sacl  Teta_cm
```

```
                ;Teta_cm=Teta_cm+K*fs=Teta_cm+tetaincr
                ;(0;360)<->(0;65535)
fs_pos
        rpt   #3
        sfr
        sacl  Teta_cm1;(0;360)<->(0;4096), this variable
                      ;is used only for the visualization
*************************************************
*       END Current Model
*************************************************
*****************************************************
*       Field Weakening function
*       input:n_ref, output iSdref  4.12 format
*****************************************************
        spm   2       ;PM=10, four left shift after multiplication
        lacc  n_ref
        abs           ;we consider absolute value of speed reference
        rpt   #3
        sfr
        sacl  n_ref8_8;speed reference 8.8f
        sub   #100h
        bcnd  noFieldWeakening,LEQ
        lacc  p0,12
        lt    n_ref8_8
        mpy   p1
        apac
        sach  tmp,4   ;tmp=p0+p1*n_ref
        sqra  n_ref8_8
        pac
        sach  tmp1,4
        lacc  tmp,12
        lt    tmp1    ;tmp1=n_ref^2
        mpy   p2
        apac
        sach  tmp,4   ;tmp=p0+p1*n_ref+p2*(n_ref^2)
        lt    tmp1
        mpy   n_ref8_8
        pac
        sach  tmp1,4  ;tmp1=n_ref^3
        lacc  tmp,12
        lt    tmp1
        mpy   p3
        apac
        sach  tmp,4   ;tmp=p0+p1*n_ref+p2*(n_ref^2)+p3*(n_ref^3)
        lacc  tmp,4   ;iSdref 8.8 f
        sacl  iSdref  ;iSdref 4.12 f with Field Weakening
        b     endFW
noFieldWeakening
        lacc  #2458   ;iSdref=0.6 pu
        sacl  iSdref  ;iSdref 4.12 f without Field Weakening
endFW
        spm   0       ;PM=0
*************************************************
*       END Field Weakening function
*************************************************

*********************************************************
*       q-axis current regulator with integral component *     correction
*       (iSq,iSqref)->(vSqref)
*********************************************************
        lacc  iSqref
        sub   iSq
        sacl  epiq    ;epiq=iSqref-iSq, 4.12 format
        lacc  xiq,12
        lt    epiq
        mpy   Kpi
        apac
```

```
        sach  upi,4   ;upi=xiq+epiq*Kpi, 4.12 format

        bit   upi,0
        bcnd  upimagzeroq,NTC
        lacc  #Vmin
        sub   upi
        bcnd  neg_satq,GT       ;if upi<Vmin branch to saturate
        lacc  upi      ;value of upi is valid
        b     limiterq
neg_satq
        lacc  #Vmin   ;set ACC to neg saturation
        b     limiterq
upimagzeroq           ;Value was positive
        lacc  #Vmax
        sub   upi
        bcnd  pos_satq,LT       ;if upi>Vmax branch to saturate
        lacc  upi               ;value of upi is valid
        b     limiterq
pos_satq
        lacc  #Vmax   ;set ACC to pos saturation
limiterq
        sacl  vSqref  ;Save ACC as reference value
        sub   upi
        sacl  elpi    ;elpi=vSqref-upi, 4.12 format
        lt    elpi
        mpy   Kcor    ;change to dma
        pac
        lt    epiq
        mpy   Ki      ;change to dma
        apac
        add   xiq,12
        sach  xiq,4   ;xiq=xiq+epiq*Ki+elpi*Kcor, 4.12 f
*************************************************************
*       END q-axis regulator with integral component correction
*************************************************************

*******************************************************
*       d-axis current regulator with integral component
*       correction
*       (iSd,iSdref)->(vSdref)
*******************************************************
        lacc  iSdref
        sub   iSd
        sacl  epid    ;epid=iSdref-iSd, 4.12 format
        lacc  xid,12
        lt    epid
        mpy   Kpi
        apac
        sach  upi,4   ;upi=xid+epid*Kpi, 4.12 format

        bit   upi,0
        bcnd  upimagzerod,NTC
        lacc  #Vmin
        sub   upi
        bcnd  neg_satd,GT       ;if upi<Vmin branch to saturate
        lacc  upi               ;upi value 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               ;upi value valid
        b     limiterd
```

```
pos_satd
        lacc  #Vmax   ;set acc to pos saturation
limiterd
        sacl  vSdref  ;store ACC as reference value
        sub   upi
        sacl  elpi    ;elpi=vSdref-upi, 4.12 format
        lt    elpi
        mpy   Kcor
        pac
        lt    epid
        mpy   Ki
        apac
        add   xid,12
        sach  xid,4   ;xid=xid+epid*Ki+elpi*Kcor, 4.12 f
*************************************************************
*       END d-axis regulator with integral component correction
*************************************************************

*************************************************************
*       Inverse Park transformation
*       (d,q) -> (alfa,beta)
*       vSbe_ref = vSqref * cos(Teta_cm)+ vSdref * sin(Teta_cm)
*       vSal_ref =-vSqref * sin(Teta_cm) + vSdref * cos(Teta_cm)
*************************************************************
        lacc  #0
        lt    vSdref
        mpy   sinTeta_cm
        lta   vSqref
        mpy   cosTeta_cm
        mpya  sinTeta_cm
        sach  vSbe_ref,4
              ;vSbe_ref=vSqref*cosTeta_cm+vSdref*sinTeta_cm
        lacc  #0
        lt    vSdref
        mpys  cosTeta_cm
        apac
        sach  vSal_ref,4
              ;vSal_ref=vSdref*cosTeta_cm-vSqref*sinTeta_cm
************************************
*       END Inverse Park transformation
************************************

init

*****************************************
*       SPACE VECTOR Pulse Width Modulation
*****************************************
*** sector calculation***

************************************************
*       Vref1 = vSbe_ref
*       Vref2 = (-vSbe_ref + sqrt(3) * vSal_ref) / 2
*       Vref3 = (-vSbe_ref - sqrt(3) * vSal_ref) / 2
************************************************
        lt    vSal_ref
        mpy   #SQRT32
        pac
        sub   vSbe_ref,11
        sach  Vref2,4 ;4.12 format
        pac
        neg
        sub   vSbe_ref,11
        sach  Vref3,4 ;4.12 format
        lacl  vSbe_ref
        sacl  Vref1   ;4.12 format
************************************************
*       END reference voltage for sector calculation
```

```
*************************************************
        lt    VDCinvT
        mpy   #SQRT32
        pac
        sach  tmp,4   ;tmp=VDCinvT*SQRT32, 4.12 format
        lt    tmp
        mpy   vSbe_ref
        pac
        sach  X,4     ;tmp*vSbe_ref, 4.12 format
        lacc  X       ;ACC = vSbe_ref*VDCinvT*SQRT32
        sacl  tmp1    ;tmp1=vSbe_ref*VDCinvT*SQRT32, 4.12 format
        sacl  X,1     ;X=(2*SQRT32*vSbe_ref*VDCinvT), 4.12 format
        lt    VDCinvT
        splk  #1800h,tmp        ;3/2, 4.12 format
        mpy   tmp     ;implement mpy #01800h
        pac
        sach  tmp,4   ;tmp=(3/2)*VDCinvT, 4.12 format
        lt    tmp
        mpy   vSal_ref
        pac
        sach  tmp,4   ;tmp=(3/2)*VDCinvT*vSal_ref, 4.12 format
        lacc  tmp     ;reload ACC with
                      ;(3/2)*VDCinvT*vSal_ref
        add   tmp1    ;tmp1=vSbe_ref*VDCinvT*SQRT32,
                      ;4.12 format
        sacl  Y       ;Y=SQRT32*VDCinvT*vSbe_ref+(3/2)*VDCinvT*vSal_ref,
                      ;4.12 format
        sub   tmp,1
        sacl  Z       ;Z=SQRT32*VDCinvT*vSbe_ref-(3/2)*VDCinvT*vSal_ref,
                      ;4.12 format
*** 60 degrees sector determination
        lacl  #0
        sacl  sector
        lacc  Vref1
        bcnd  Vref1_neg,LEQ     ;If Vref1<0 do not set bit 1 of sector
        lacc  sector
        or    #1
        sacl  sector
Vref1_neg
        lacc  Vref2
        bcnd  Vref2_neg,LEQ     ;If Vref2<0 do not set bit 2 of sector
        lacc  sector
        or    #2
        sacl  sector
Vref2_neg
        lacc  Vref3
        bcnd  Vref3_neg,LEQ     ;If Vref3<0 do not set bit 3 of sector
        lacc  sector
        or    #4
        sacl  sector
Vref3_neg
***     END 60 degrees sector determination

***     T1 and T2 (= t1 and t2) calculation depending on the
***     sector number
        lacl  sector
        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
***      END t1 and t2 calculation

         lacc  t1      ;if t1+t2>PWMPRD we have to saturate t1 and t2
         add   t2
         sacl  tmp
         sub   #PWMPRD
         bcnd  nosaturation,LT,EQ
***      t1 and t2 saturation
         lacc  #PWMPRD,15       ;divide PWMPRD by (t1+t2)
         rpt   #15
         subc  tmp
         sacl  tmp
         lt    tmp     ;calculate saturate values of t1 and t2
         mpy   t1      ;t1 (saturated)=t1*(PWMPRD/(t1+t2))
         pac
         sach  t1,1
         mpy   t2      ;t2 (saturated)=t2*(PWMPRD/(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
        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','da4'
*       Output on 12 bit Digital analog Converter
*       5V equivalent to FFFh
*****************************************************
        lacc  sector,7;scale sector by 2^7 to have good displaying
        sacl  sector

*** DAC out channel 'da1'
        lacc  #ia     ;get the address of the first elements
        add   da1     ;add the selected output variable
```

```
                    ;offset 'da1' sent by the terminal
        sacl  daout   ;now daout contains the address of
                    ;the variable to send to DAC1
        lar   ar5,daout         ;store it in AR5

        lacc  *       ;indirect addressing, load the value to send out
                    ;the following 3 instructions are
                    ;required to adapt the numeric
                    ;format to the DAC resolution
        sfr           ;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,DAC0_VAL
*** END DAC out channel 'da1'

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

        lacc  *       ;indirect addressing, load the
                    ;value to send out
                    ;the following 3 instructions are
                    ;required to adapt the numeric
                    ;format to the DAC resolution
        sfr           ;we have 10 bit DAC, we want to
                    ;have the number 2000h = 5 Volt
        sfr
        add   #800h
        sacl  daouttmp;to prepare the triggering of DAC1 buffer
        out   daouttmp,DAC1_VAL
*** END DAC out channel 'da2'

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

        lacc  *       ;indirect addressing, load the value to send out
                    ;the following 3 instructions are
                    ;required to adapt the numeric
                    ;format to the DAC resolution
        sfr           ;we have 10 bit DAC, we want to have
                    ;the number 2000h = 5 Volt
        sfr
        add   #800h
        sacl  daouttmp;to prepare the triggering of DAC1 buffer
        out   daouttmp,DAC2_VAL
*** END DAC out channel 'da3'

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

        lacc  *       ;indirect addressing, load the value to send out
```

```
                        ;the following 3 instructions are
                        ;required to adapt the numeric
                        ;format to the DAC resolution
        sfr             ;we have 10 bit DAC, we want to have
                        ;the number 2000h = 5 Volt
        sfr
        add   #800h
        sacl  daouttmp;to prepare the triggering of DAC1 buffer
        out   daouttmp,DAC3_VAL
*** END DAC out channel 'da4'

        OUT   tmp,DAC_VAL       ;start convertion

        ldp   #IFRA>>7
        splk  #200h,IFRA        ;Clear all flags, may be
                                ;change with only T1 underflow int.
************************************
*       Context restore and Return
************************************
        larp  ar7
        mar   *+
        lacl  *+        ;Accu. restored for context restore
        add   *+,16
        lst   #0,*+
        lst   #1,*+

        clrc  INTM
        ret
************************************
*       END Context Restore and Return
************************************
*************************************************************
*       END _c_int2 ISR
*       synchronization of the control algorithm with the PWM
*       underflow interrupt
*************************************************************

_c_int0:
*****************************
*       Board general settings
*****************************
        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
*       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
                                ;Do not divide PLL
                                ;PLL ratio x2
```

```
splk  #10000011b,CKCR0  ;PLL enabled
                        ;LPM0
                        ;ACLK enabled
                        ;SYSCLK 10MHz PLL x2
                        ;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
sacl  *
out   *,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         ; No software dead-band
splk  #666h,ACTR        ;Bits 15-12 not used, no space vector
              ;PWM compare actions
              ;PWM6/PWM5 -Active Low/Active High
              ;PWM4/PWM3 -Active Low/Active High
              ;PWM2/PWM1 -Active Low/Active High
splk  #100,CMPR1
splk  #200,CMPR2
splk  #300,CMPR3
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
*********************************************************
*       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      ;Ports A/B all IO except ADCs
        splk  #00F9h,OPCRB      ;Port C as non IO function except
        ;IOPC2&3
        splk  #0FF08h,PCDATDIR  ;bit IOPC3
*** END: PWM enable

*********************************************************
*       Initialize ar7 as the stack for context save
*       space reserved: DARAM B2 60h-80h (page 0)
*********************************************************
        lar   ar7,#79h

*********************************************************
*       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 oscillator
*** END A/D initialization

***************************
*       Variables initialization
***************************
        ldp   #ctrl_n ;control variable page
        zac
        sacl  run
        sacl  Index
        sacl  xid
        sacl  xiq
        sacl  xin
        sacl  upi
        sacl  elpi
        sacl  Vref1
        sacl  Vref2
        sacl  Vref3
        sacl  da1
        splk  #1b9dh,VDC        ;The DC voltage is 310V
                                ;Vdc=1.726 in 4.12 with a Vbase=179.6V
```

```
        splk  #243h,VDCinvT     ;T/(Vdc*2) or PWMPRD/VDC=579
                                ;rescaled by 4.12
        lacc  #1
        sacl  da2
        lacc  #2
        sacl  da3
        lacc  #3
        sacl  da4

        setc  OVM
        spm   0       ;no shift after multiplication
        setc  sxm     ;sign extension mode
*******************************
*       END Variables initialization
*******************************
***************************
*       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  #0200h,IMRA       ;PDPINT is disabled, with 0201h is enabled
        splk  #0000h,IMRB
        splk  #0000h,IMRC
              ;Set IMR for INT2 and clear any Flags
              ;INT2 (PWM interrupt) is used for motor control
              ;synchronization
        ldp   #0h
        lacc  #0FFh
        sacl  IFR
        lacc  #0000010b
        sacl  IMR
        ldp   #ctrl_n ;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
        clrc  xf                ;default mode (will be saved as context)
        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   #ctrl_n           ;control variable page
        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   #ctrl_n           ;control variable page
        add   serialtmp         ;add ACC with lower byte
        sacl  n_ref             ;store it n_ref
        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): flag run
****************************
navail31
        ldp   #DP_PF1
        bit   SCIRXST,BIT6      ;is there any characteravailable (8 LSB)?
        bcnd  navail31,ntc      ;if not repeat the cycle (polling)
        lacc  SCIRXBUF
        and   #0FFh             ;take the 8 LSB
        ldp   #ctrl_n           ;control variable page
        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   #ctrl_n           ;control variable page
        add   serialtmp         ;add ACC with lower byte
        sacl  run               ;store it in run
        b     menu              ;return to the main polling cycle
*** END Option 3): iSdref
notthree
        lacc  option
        sub   #034h             ;is it option 4 ?
        bcnd  notfour,neq       ;if not branch to notthree

*************************************************
*       Option 4): current regulator parameters setting
*************************************************
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   #ctrl_n           ;control variable page
        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   #ctrl_n           ;control variable page
        add   serialtmp         ;add ACC with lower byte
        sacl  Kpi               ;store it in Kpi
navail43
        ldp   #DP_PF1
        bit   SCIRXST,BIT6      ;is there any character available (8 LSB)?
        bcnd  navail43,ntc      ;if not repeat the cycle (polling)
        lacc  SCIRXBUF
        and   #0FFh             ;take the 8 LSB
        ldp   #ctrl_n           ;control variable page
        sacl  serialtmp         ;if yes, get it and store it in serialtmp
navail44
        ldp   #DP_PF1
        bit   SCIRXST,BIT6      ;8 MSB available ?
        bcnd  navail44,ntc      ;if not repeat the cycle (polling)
        lacc  SCIRXBUF,8        ;load ACC the upper byte
        ldp   #ctrl_n           ;control variable page
        add   serialtmp         ;add ACC with lower byte
        sacl  Ki                ;store it in Ki
navail45
        ldp   #DP_PF1
```

```
        bit   SCIRXST,BIT6      ;is there any character available (8 LSB)?
        bcnd  navail45,ntc      ;if not repeat the cycle (polling)
        lacc  SCIRXBUF
        and   #0FFh             ;take the 8 LSB
        ldp   #ctrl_n           ;control variable page
        sacl  serialtmp         ;if yes, get it and store it in serialtmp
navail46
        ldp   #DP_PF1
        bit   SCIRXST,BIT6      ;8 MSB available ?
        bcnd  navail46,ntc      ;if not repeat the cycle (polling)
        lacc  SCIRXBUF,8        ;load ACC the upper byte
        ldp   #ctrl_n           ;control variable page
        add   serialtmp         ;add ACC with lower byte
        sacl  Kcor              ;store it in Kcor
        b     menu              ;return to the main polling cycle
*** END Option 4):  current regulator parameters setting
notfour
        lacc  option
        sub   #035h             ;is it option 5 ?
        bcnd  notfive,neq       ;if not branch to notfive
*************************************************
*       Option 5): speed regulator parameters setting
*************************************************
navail51
        ldp   #DP_PF1
        bit   SCIRXST,BIT6      ;is there any available (8 LSB)?
        bcnd  navail51,ntc      ;if not repeat the cycle (polling)
        lacc  SCIRXBUF
        and   #0FFh             ;take the 8 LSB
        ldp   #ctrl_n           ;control variable page
        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   #ctrl_n           ;control variable page
        add   serialtmp         ;add ACC with lower byte
        sacl  Kpin              ;store it in Kpin
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   #ctrl_n           ;control variable page
        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   #ctrl_n           ;control variable page
        add   serialtmp         ;add ACC with lower byte
        sacl  Kin               ;store it in Kin
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   #ctrl_n           ;control variable page
        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   #ctrl_n           ;control variable page
        add   serialtmp         ;add ACC with lower byte
        sacl  Kcorn             ;store it in Kcorn

        b     menu              ;return to the main polling cycle
*** END Option 5):  speed regulator parameters setting

notfive
        B     menu
*****************************************************************
*       END Sensored Speed FOC for AC three phase induction motor
*****************************************************************
```

Appendix B - Linker File

```
-c
-stack 100h
/*--------------------------------------------------------  */
/*MEMORY SPECIFICATION                                      */
/*Block B0 is configured as data memory (CNFD)              */
/*and MP/MC=1                                               */
/*(microprocessor mode). Note that data memory              */
/*locations 6h--5Fh                                         */
/*and 80h--1FFh are not configured.                         */
/*--------------------------------------------------------  */

MEMORY
{
        PAGE 0:
        FLASH_VEC     : origin = 0h,  length = 40h
        FLASH         : origin = 40h, length = 3FC0h
        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
        .text         :         { }   > FLASH  PAGE 0
        .cinit        :         { }   > FLASH  PAGE 0
        .switch       :         { }   > FLASH PAGE 0
        blockb2       :         { }   > BLK_B22
        .bss          :         { }   > BLK_B0 PAGE 1
        .data         :         { }   > BLK_B0 PAGE 1
        table         :         { }   > BLK_B1 PAGE 1
        .sysmem       :         { }   > EXT_DATA PAGE 1
        .const        :         { }   > EXT_DATA PAGE 1
        .stack        :         { }   > EXT_DATA PAGE 1
}
```

Appendix C - Sine Look-up table

```
.word  0
.word 101
.word 201
.word 301
.word 401
.word 501
.word 601
.word 700
.word 799
.word 897
.word 995
.word 1092
.word 1189
.word 1285
.word 1380
.word 1474
.word 1567
.word 1660
.word 1751
.word 1842
.word 1931
.word 2019
.word 2106
.word 2191
.word 2276
.word 2359
.word 2440
.word 2520
.word 2598
.word 2675
.word 2751
.word 2824
.word 2896
.word 2967
.word 3035
.word 3102
.word 3166
.word 3229
.word 3290
.word 3349
.word 3406
.word 3461
.word 3513
.word 3564
.word 3612
.word 3659
.word 3703
.word 3745
.word 3784
.word 3822
.word 3857
.word 3889
.word 3920
.word 3948
.word 3973
.word 3996
.word 4017
.word 4036
.word 4052
.word 4065
.word 4076
.word 4085
```

```
        .word 4091
        .word 4095
        .word 4096
        .word 4095
        .word 4091
        .word 4085
        .word 4076
        .word 4065
        .word 4052
        .word 4036
        .word 4017
        .word 3996
        .word 3973
        .word 3948
        .word 3920
        .word 3889
        .word 3857
        .word 3822
        .word 3784
        .word 3745
        .word 3703
        .word 3659
        .word 3612
        .word 3564
        .word 3513
        .word 3461
        .word 3406
        .word 3349
        .word 3290
        .word 3229
        .word 3166
        .word 3102
        .word 3035
        .word 2967
        .word 2896
        .word 2824
        .word 2751
        .word 2675
        .word 2598
        .word 2520
        .word 2440
        .word 2359
        .word 2276
        .word 2191
        .word 2106
        .word 2019
        .word 1931
        .word 1842
        .word 1751
        .word 1660
        .word 1567
        .word 1474
        .word 1380
        .word 1285
        .word 1189
        .word 1092
        .word 995
        .word 897
        .word 799
        .word 700
        .word 601
        .word 501
        .word 401
        .word 301
        .word 201
        .word 101
        .word 0
```

```
.word 65435
.word 65335
.word 65235
.word 65135
.word 65035
.word 64935
.word 64836
.word 64737
.word 64639
.word 64541
.word 64444
.word 64347
.word 64251
.word 64156
.word 64062
.word 63969
.word 63876
.word 63785
.word 63694
.word 63605
.word 63517
.word 63430
.word 63345
.word 63260
.word 63177
.word 63096
.word 63016
.word 62938
.word 62861
.word 62785
.word 62712
.word 62640
.word 62569
.word 62501
.word 62434
.word 62370
.word 62307
.word 62246
.word 62187
.word 62130
.word 62075
.word 62023
.word 61972
.word 61924
.word 61877
.word 61833
.word 61791
.word 61752
.word 61714
.word 61679
.word 61647
.word 61616
.word 61588
.word 61563
.word 61540
.word 61519
.word 61500
.word 61484
.word 61471
.word 61460
.word 61451
.word 61445
.word 61441
.word 61440
.word 61441
.word 61445
.word 61451
```

```
        .word 61460
        .word 61471
        .word 61484
        .word 61500
        .word 61519
        .word 61540
        .word 61563
        .word 61588
        .word 61616
        .word 61647
        .word 61679
        .word 61714
        .word 61752
        .word 61791
        .word 61833
        .word 61877
        .word 61924
        .word 61972
        .word 62023
        .word 62075
        .word 62130
        .word 62187
        .word 62246
        .word 62307
        .word 62370
        .word 62434
        .word 62501
        .word 62569
        .word 62640
        .word 62712
        .word 62785
        .word 62861
        .word 62938
        .word 63016
        .word 63096
        .word 63177
        .word 63260
        .word 63345
        .word 63430
        .word 63517
        .word 63605
        .word 63694
        .word 63785
        .word 63876
        .word 63969
        .word 64062
        .word 64156
        .word 64251
        .word 64347
        .word 64444
        .word 64541
        .word 64639
        .word 64737
        .word 64836
        .word 64935
        .word 65035
        .word 65135
        .word 65235
        .word 65335
        .word 65435
```

Appendix D - User Interface Software

```
REM            ************************************************ *
REM            TEXAS INSTRUMENTS                                *
REM            Sensored Speed Field Orientated Control of an AC  *
REM            induction motor                                  *
REM            ************************************************ *
REM            USER INTERFACE program :  foc.BAS                *
REM            Author                 :  Riccardo Di Gabriele   *
REM            ************************************************ *

   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$(16); CHR$(18); CHR$(17); CHR$(19); :
   REM dac initialization
   PRINT #1, "3"; CHR$(0); CHR$(0); : REM flag run initialization to 0
   PRINT #1, "4"; CHR$(0); CHR$(16); CHR$(0); CHR$(1); CHR$(0); CHR$(1): REM current PI parameters
   initialization
   PRINT  #1,  "5";  CHR$(43);  CHR$(72);  CHR$(53);  CHR$(0);  CHR$(11);  CHR$(0):  REM  speed  PI
   parameters initialization

   flag = 0
   run$(0) = " N "
   run$(1) = " Y "
   speedref = 0
   da1 = 16: da2 = 18
   da3 = 17: da4 = 19
   Kpi = 1
   Ki = .0625
   Kcor = .0625

   Kpin = 4.51
   Kin = .0129
   Kcorn = .0028
   speedpu = 1500: REM mechanical base speed

   DIM daout$(200)
   daout$(0) = "ia"
   daout$(1) = "ib"
   daout$(2) = "ic"
   daout$(3) = "t1"
   daout$(4) = "t2"
   daout$(5) = "Vref1"
   daout$(6) = "Vref2"
   daout$(7) = "Vref3"
   daout$(8) = "VDC"
   daout$(9) = "taon"
   daout$(10) = "tbon"
   daout$(11) = "tcon"
   daout$(12) = "iSalfa"
   daout$(13) = "iSbeta"
   daout$(14) = "VSalfar"
   daout$(15) = "VSbetar"
   daout$(16) = "iSdref"
   daout$(17) = "iSqref"
   daout$(18) = "iSd"
   daout$(19) = "iSq"
   daout$(20) = "VSdref"
   daout$(21) = "VSqref"
   daout$(22) = "epiq"
   daout$(23) = "epid"
   daout$(24) = "xiq"
   daout$(25) = "xid"
   daout$(26) = "n"
   daout$(27) = "n_ref"
   daout$(28) = "epin"
   daout$(29) = "xin"
   daout$(30) = "X"
   daout$(31) = "Y"
   daout$(32) = "Z"
   daout$(33) = "sector"
   daout$(34) = "Teta_cm"
   daout$(35) = "sinTeta_cm"
   daout$(36) = "cosTeta_cm"
```

```
daout$(37) = "i_mr"
daout$(38) = "fs"

nDA = 12
1 CLS
FOR i = 0 TO nDA
COLOR 11
LOCATE (11 + i), 2: PRINT "("; : PRINT USING "##"; i; : PRINT ") "; daout$(i)
LOCATE (11 + i), 22: PRINT "("; : PRINT USING "##"; i + nDA + 1; : PRINT ") "; daout$(i + nDA +
1)
LOCATE (11 + i), 42: PRINT "("; : PRINT USING "##"; i + 2 * nDA + 2; : PRINT ") "; daout$(i + 2
* nDA + 2)
LOCATE (11 + i), 62: PRINT "("; : PRINT USING "##"; i + 3 * nDA + 3; : PRINT ") "; daout$(i + 3
* nDA + 3)
NEXT i
LOCATE 1, 12
COLOR 12: PRINT " Sensored Field Orientated Control of an AC Induction 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 " Run   (N=NoRun)      ("; run$(flag); ")"
COLOR 10: LOCATE 3, 50: PRINT "   <4>"; : 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 "   <5>"; : COLOR 2: PRINT " Kpin      ("; Kpin; "pu)"
COLOR 10: LOCATE 7, 50: PRINT "      "; : COLOR 2: PRINT " Kin       ("; Kin; "pu)"
COLOR 10: LOCATE 8, 50: PRINT "      "; : COLOR 2: PRINT " Kcorn     ("; Kcorn; "pu)"

COLOR 10: LOCATE 9, 10: PRINT "Choice : ";
DO
a$ = INKEY$
LOOP UNTIL ((a$ <= "5") 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$) / speedpu
     IF (speedrpu >= 7.999755859#) THEN speedrpu =
           7.999755859#
     IF (speedrpu <= -8) THEN speedrpu = -8
     speedrefpu = CLNG(speedrpu * 4096)
     IF (speedref < 0) THEN speedrefpu = 65536 +
       speedrefpu
     REM Send "Option" - "LSB" - "MSB"
     PRINT #1, "1"; CHR$(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$)
        END IF
        REM Send "Option" - "LSB" - "MSB"
        PRINT #1, "2"; CHR$(da1 AND 255); CHR$(da2 AND 255);
          CHR$(da3
          AND 255); CHR$(da4 AND 255)
        GOTO 1
    CASE "3"
        REM 4.12 format
        IF (flag = 1) THEN flag = 0 ELSE flag = 1
        flagpu = CLNG(flag * 4096)
        REM Send "Option" - "LSB" - "MSB"
        PRINT #1, "3"; CHR$(flagpu AND 255); CHR$((flagpu AND       65280) /       256)

    31
        GOTO 1
    CASE "4"

        REM 4.12 format
        PRINT a$; ") ";
        PRINT "Kpi ("; Kpi; " ) : ";
        INPUT Kpi$
        IF Kpi$ = "" THEN 41
        Kpi = VAL(Kpi$)
        REM Send "Option" - "LSB" - "MSB"
    41
        PRINT "                     Ki ("; Ki; " ) : ";
        INPUT Ki$
        IF Ki$ = "" THEN 42
        Ki = VAL(Ki$)
    42
        Kpipu = 4096 * Kpi
        Kip = 4096 * Ki
        Kcor = (Ki / Kpi)
        Kcorpu = 4096 * Kcor
        REM Send "Option" - "LSB" - "MSB"
        PRINT #1, "4"; CHR$(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 "5"

        REM 4.12 format
        PRINT a$; ") ";
        PRINT "Kpin ("; Kpin; " ) : ";
        INPUT Kpin$
        IF Kpin$ = "" THEN 51
        Kpin = VAL(Kpin$)
        REM Send "Option" - "MSB" - "LSB"
    51
        PRINT "                     Kin ("; Kin; " ) : ";
        INPUT Kin$
        IF Kin$ = "" THEN 52
        Kin = VAL(Kin$)
    52
        Kpinpu = 4096 * Kpin
        Kinpu = 4096 * Kin
        Kcorn = (Kin / Kpin)
        Kcornpu = 4096 * Kcorn
        REM Send "Option" - "LSB" - "MSB"
        PRINT #1, "5"; CHR$(Kpinpu AND 255); CHR$((Kpinpu AND       65280) /       256); CHR$(Kinpu
    AND 255); CHR$((Kinpu      AND 65280) / 256);     CHR$(Kcornpu AND 255);        CHR$((Kcornpu
    AND 65280) / 256)
        GOTO 1
    END SELECT
I.      CLOSE #1
```