Sensorless Control with Kalman Filter on TMS320 Fixed-Point DSP

Literature Number: BPRA057
Texas Instruments Europe
July 1997

IMPORTANT NOTICE

Texas Instruments (TI) reserves the right to make changes to its products or to discontinue any semiconductor product or service without notice, and advises its customers to obtain the latest version of relevant information to verify, before placing orders, that the information being relied on is current.

TI warrants performance of its semiconductor products and related software 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, INTENDED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS.

Inclusion of TI products in such applications is understood to be fully at the risk of the customer. Use of TI products in such applications requires the written approval of an appropriate TI officer. Questions concerning potential risk applications should be directed to TI through a local SC sales office.

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

TI assumes no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI 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.

Contents

1. Notation and Symbols 1
2. The Hardware 2
3. The Field Oriented Control Method 3
4. Motor Model Used in the Field Oriented Control 4
5. Implementation of the Field Oriented Control Method 6
6. Kalman Filter 9
7. Basics of Observers 10
8. Basics of Kalman Filters 12
9. Motor Model for the Kalman Filter 14
10. Simulation of the Kalman Filter 17
11. Simulation Results 18
12. Real-Time Implementation of the Kalman Filter 21
13. Real Time Results 27
14. Conclusions and Possible Development 29
References 30
Appendix Source Code of Programs 32

List of Figures

Figure 1: New Control Hardware System 3
Figure 2: Model 1 - Field Oriented Controller for the ASM 8
Figure 3: Reconstruction of the State Vector 10
Figure 4: Structure of the Luenberger Observer 11
Figure 5: Model 2 - Controller with Kalman Filter 18
Figure 6: Speed Reversal with FOC 19
Figure 7: Speed Reversal with EKF 19
Figure 8: Applying Load at 0 RPM with FOC 20
Figure 9: Applying Load at 0 RPM with EKF 20
Figure 10: Speed Reversal with Kalman Filter 27
Figure 11: Speed Reversal with Speed Measurement 28

List of Tables

Table 1: Summary of Notational Conventions....2
Table 2: Some Types of Matrix Multiplication Needed For Kalman Filter....23
Table 3: Scaling Factors of Variables....25
Table 4: Constant Values....25
Table 5: Memory Requirements of Kalman Filter....28

Sensorless Control with Kalman Filter on Fixed-Point DSP

ABSTRACT

The importance of Digital Motor Control (DMC) has grown gradually. As Digital Signal Processors have become cheaper, and their performance greater, it has become possible to use them for controlling electrical drives as a *cost effective* solution. Some relatively new methods such as speed sensorless field oriented control utilize this enhanced processing capacity. This document discusses the implementation of a *sensorless* field oriented control for induction motors using the Kalman Filter. First the theory of field oriented methodology, with and without speed sensor, is described. Then a simulation approach is given for both cases. Finally the real-time implementation issues of a sensorless control are discussed. The paper presents an evaluation of the results. The processing capability of the processor is used to 50% at the current cycle times, the memory requirement is approximately 6823 Word program, and 2564 Word data space, of which 1024 Words are C-stack. The appendix contains full source code of the sensorless control for the TMS320C50[1] DSP, which is source compatible to the other members of the Fixed-Point DSP family like 'C1x, 'C2xx.

1. Notation and Symbols

In this chapter the notational conventions will be summarized, as used in this document. Throughout this work notations used by [6] (University Paderborn), [7] (dSPACE) and [4] (Beierke) will be followed. The internal variable notations of the programs will not be discussed here. The meanings of the variables are documented in the corresponding programs.

[1] In this document we will use the following abbreviations: C14 for TMS320C14, C50 for the TMS320C50 etc.

Table 1: Summary of Notational Conventions

Meaning	Notation
Electrical and Mechanical Torque	m_e, m_L
Fluxes[2]	Ψ
Flux Angle	ρ or ε_{FS}
Flux Speed	ω_{mR} or ω_{FS}
Inertia	J
Leakage Factor	$\sigma = \frac{K_L}{L_S}$
Stator and Rotor Leakage Factor	σ_R, σ_S
Magnetic Pole Count	z_p or p
Magnetizing current	i_{mR}
Phase Currents	i_a, i_b, i_c
Rotor, Stator and Main Inductances	L_R, L_S, L_H
Rotor and Stator Resistances	R_R, R_S
Rotor and Stator Time Constants	T_R, T_S
Rotor Angle	ε or ε_{RS}
Rotor Currents Scalar Components	$i_{R\alpha}, i_{R\beta}$
Rotor Speed	ω or ω_{RS}
Stator and Rotor Currents	$\underline{i}_R, \underline{i}_S$
Stator Currents in Field Coordinates	i_{Sd}, i_{Sq}
Stator Currents Scalar Components	$i_{S\alpha}, i_{S\beta}$
Voltages[3]	u

Note, that in most of the articles about field oriented control rotor flux is regarded simply as “flux” and this document will follow this convention as well.

2. The Hardware

This chapter will give a very short overview of the hardware used in this project. This motor controller card (see Figure 1 on page 3) is based on a TMS320C50 DSP (Digital Signal Processor) manufactured by Texas Instruments. Other main elements of the system are a UART for serial communication with a PC or VT-100 terminal, an A/D converter with an analog multiplexer, which can be used to input up to 8 channels of analog signals in the range of 0-5V, a PWM generator, which is implemented inside an FPGA of Texas Instruments, a GPIO unit, with an integrated incremental encoder interface. All this features are now integrated in a new device called DSP-Controller TMS320C240.

[2]Indices are just the same as for currents
[3]Indices are just the same as for currents

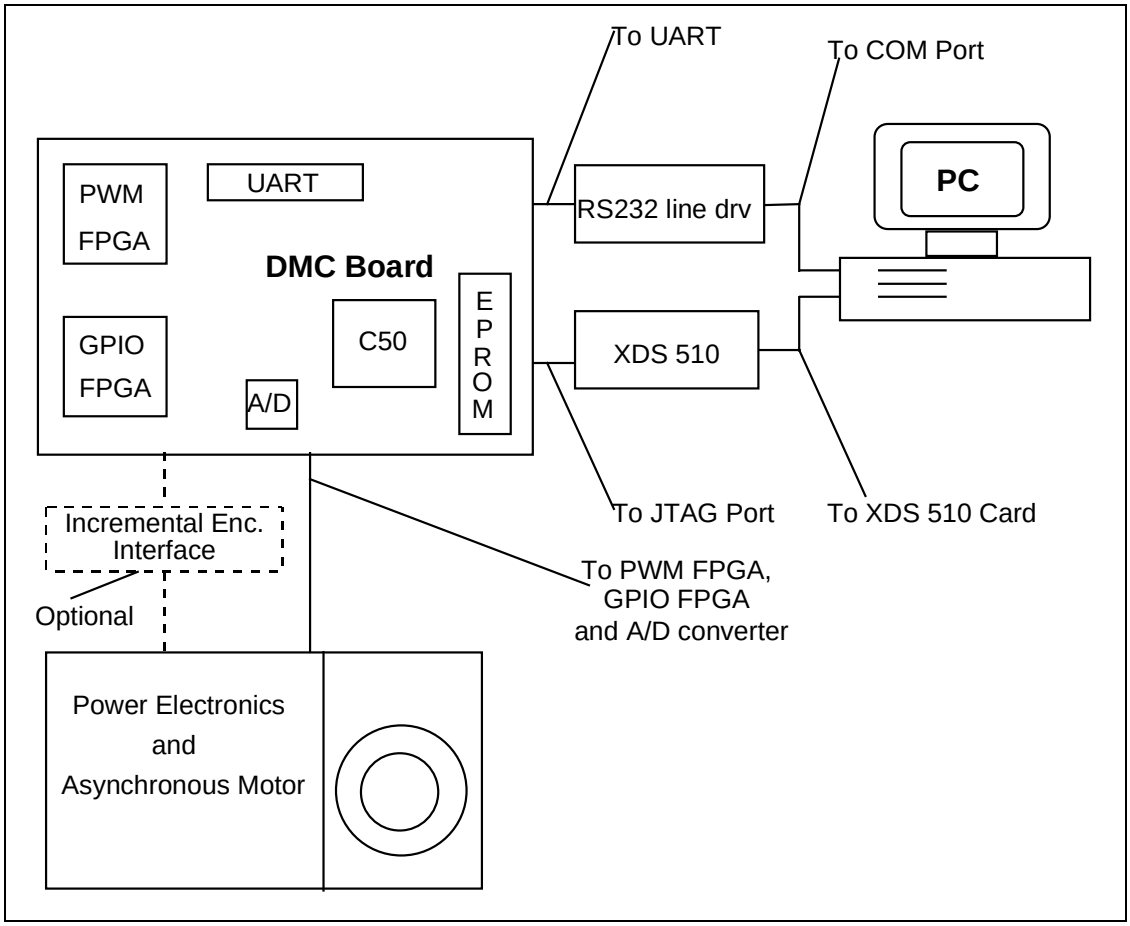

Figure 1: New Control Hardware System

3. The Field Oriented Control Method

In the case of an asynchronous 3 phase motor, sometimes regarded as an induction cage motor, a very elegant control method, the field oriented control is available. The main feature of this method is that all variables are converted to the coordinate system of the magnetic field of the rotor, called the rotor flux. The flux is held constant using the current component parallel to the rotor flux (isd), the torque is controlled by the other current component (isq). This method is basically the same as controlling separately excited DC motors. The control method is not very complicated, however the calculation of the rotor flux and a conversion of the variables from the stator system to the flux system requires high processor capacity, since a conversion between field coordinates and stator coordinates is needed in both directions in each controlling cycle. The C14 DSP was completely capable of doing this job in real time. The motor and the controlling hardware is relatively cheap, on the other hand the software is quite complicated. But this complicated software can be used in a quite flexible way, once it is developed.

The main reason for using this method is its dynamic performance. Specifically, it offers good lead performance, and resistance against disturbances such as changes of the load torque. These properties can be achieved by a decoupling of the flux and the torque, which is possible with a field oriented model. In this case, as was mentioned before, not only the structure of the control will be the same as the separately excited DC motor but we also get a similarly good dynamic control behaviour.

4. Motor Model Used in the Field Oriented Control

The model of the AC drive will be described in field coordinates[4]. As a basis, the equations of the motor as described in [4] will be used. This system of equations is nonlinear. The indices "R" and "S" mean rotor and stator respectively.

$$\underline{u}_S(t) = R_S \underline{i}_S(t) + \frac{d}{dt}\underline{\Psi}_S(t) = R_S \underline{i}_S(t) + L_S \frac{d}{dt}\underline{i}_S(t) + L_H \frac{d}{dt}(\underline{i}_R(t) e^{j\varepsilon(t)}) \tag{1}$$

$$0 = R_R \underline{i}_R(t) + \frac{d}{dt}\underline{\Psi}_R(t) = R_R \underline{i}_R(t) + L_R \frac{d}{dt}\underline{i}_R(t) + L_H \frac{d}{dt}(\underline{i}_S(t) e^{-j\varepsilon(t)}) \tag{2}$$

$$m_d(t) = \frac{2}{3} z_p L_H \operatorname{Im}\left[\underline{i}_S(t)(\underline{i}_R(t) e^{j\varepsilon(t)})^*\right] \tag{3}$$

$$\frac{d\omega(t)}{dt} = \frac{z_p}{J}\left(m_d(t) - m_L(t)\right) \tag{4}$$

$$\frac{d\varepsilon(t)}{dt} = \omega(t) \tag{5}$$

The inductances in these equations are defined as:

$$L_S = L_{\sigma S} + L_H = (1+\sigma_S) L_H \qquad L_R = L_{\sigma R} + L_H = (1+\sigma_R) L_H \tag{6}$$

The total leakage factor is defined as follows:

$$\sigma = 1 - \frac{1}{(1+\sigma_S)(1+\sigma_R)} \tag{7}$$

L_H is the main inductance of the motor, R_S and R_R are the resistances of the windings.

We will now define the model of the asynchronous motor in field coordinates. This will make it possible to implement the controller in field coordinates. Field orientation is described in detail in [1]. The basic principle is, that we convert all values to the coordinate system of the magnetic field, decompose the stator current vector into a field

[4] In none of these models of the Asynchronous Motor do we take the mechanical losses into account.

generating, and a torque generating (i_{Sd} and i_{Sq} respectively) component. Once this is done, the actual control becomes very simple. Since our program does not reach the field weakening range, we will keep the field generating component at a constant value to generate a necessary field, and control the torque generating component according to the speed. This means, that the output of the speed controller is the reference for the i_{Sq} controller.

We will eliminate the use of flux in the model, and we will use the magnetizing current instead. The connection between these two variables is as follows:

$$\underline{i}_{mR}(t) = (1+\sigma_R)\underline{i}_R(t)e^{j\varepsilon(t)} + \underline{i}_S(t) = \underline{i}_{mR}(t)e^{j\rho(t)} = \frac{1}{L_H}\underline{\Psi}_R(t)e^{j\varepsilon(t)} \tag{8}$$

The transformation between the systems will be done by the following transformation:

$$\underline{u}_S e^{-j\rho} = u_{Sd} + ju_{Sq} \tag{9}$$

$$\underline{i}_S e^{-j\rho} = i_{Sd} + ji_{Sq} \tag{10}$$

Let us now consider the equations of the model in field coordinate system. Note that in these equations $T_R = \frac{L_R}{R_R}$. Note also, that the rotor based variables are also completely eliminated.

$$u_{Sd}(t) = R_S i_{Sd}(t) + \sigma L_S \frac{d}{dt} i_{Sd}(t) - \sigma L_S \omega_{mR}(t) i_{Sq}(t) + (1-\sigma) L_S \frac{d}{dt} i_{mR}(t) \tag{11}$$

$$u_{Sq}(t) = R_S i_{Sq}(t) + \sigma L_S \frac{d}{dt} i_{Sq}(t) + \sigma L_S \omega_{mR}(t) i_{Sd}(t) + (1-\sigma) L_S \omega_{mR}(t) i_{mR}(t) \tag{12}$$

$$i_{Sd}(t) = i_{mR}(t) + T_R \frac{d}{dt} i_{mR}(t) \tag{13}$$

$$i_{Sq}(t) = (\omega_{mR}(t) - \omega(t)) T_R i_{mR}(t) \tag{14}$$

$$m_d(t) = \frac{2}{3} z_p (1-\sigma) L_S i_{mR}(t) i_{Sq}(t) \tag{15}$$

$$\frac{J}{z_p} \frac{d\omega(t)}{dt} = m_d(t) - m_L(t) \tag{16}$$

This model has been the basis for the field oriented control. Note, that the Kalman filter in the next section is based on another model, which uses rotor fluxes and stator currents as state variables. These two models are equivalent. The other model will be presented where the Kalman filter is introduced.

5. Implementation of the Field Oriented Control Method

Using the field oriented model of the motor, it has theoretically become possible to realize a control in field orientation, which makes it extremely easy to control speed by controlling i_{Sq}. However our system of equations is nonlinear. We will eliminate this non-linearity by a decoupling of the nonlinear terms. This method is taken from [4].

$$u_{Sd} = u_{Sd}^{Lin} + u_{Sd}^{Couple} = \left[R_S i_{Sd} + \sigma L_S \frac{d}{dt} i_{Sd} \right] + \left[-\sigma L_S \omega_{mR} i_{Sq} + (1-\sigma) L_S \frac{d}{dt} i_{mR} \right] \tag{17}$$

$$u_{Sq} = u_{Sq}^{Lin} + u_{Sq}^{Couple} = \left[R_S i_{Sq} + \sigma L_S \frac{d}{dt} i_{Sq} \right] + \left[\sigma L_S \omega_{mR} i_{Sq} + (1-\sigma) L_S \omega_{mR} i_{mR} \right] \tag{18}$$

From these equations we derive the formula for the linearized voltages, which will make our equations linear.

$$u_{Sd}^{Lin} = u_{Sd} - u_{Sd}^{Couple} = \left[R_S i_{Sd} + \sigma L_S \frac{d}{dt} i_{Sd} \right] \tag{19}$$

$$u_{Sq}^{Lin} = u_{Sq} - u_{Sq}^{Couple} = \left[R_S i_{Sq} + \sigma L_S \frac{d}{dt} i_{Sq} \right] \tag{20}$$

Now we will presume, that $i_{mR} = i_{mR}^{ref}$, and that it is constant. This will remove the last non linearity by making K_J constant.

$$\frac{d}{dt}\begin{bmatrix} i_{Sd} \\ i_{mR} \\ i_{Sq} \\ \omega \\ \varepsilon \end{bmatrix} = \begin{bmatrix} -\frac{R_S}{\sigma L_S} & 0 & 0 & 0 & 0 \\ \frac{1}{T_R} & -\frac{1}{T_R} & 0 & 0 & 0 \\ 0 & 0 & -\frac{R_S}{\sigma L_S} & 0 & 0 \\ 0 & 0 & K_J & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} i_{Sd} \\ i_{mR} \\ i_{Sq} \\ \omega \\ \varepsilon \end{bmatrix} + \begin{bmatrix} \frac{1}{\sigma L_S} & 0 & 0 \\ 0 & 0 & 0 \\ 0 & \frac{1}{\sigma L_S} & 0 \\ 0 & 0 & -\frac{z_p}{J} \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} u_{Sd}^{Lin} \\ u_{Sq}^{Lin} \\ m_L \end{bmatrix} \tag{21}$$

where

$$K_J = \frac{2}{3}\frac{z_p^2}{J}(1-\sigma) L_S i_{mR}^{ref} \tag{22}$$

This system can be split into two systems, the d and the q subsystem. This will enable us to build up a separate controller for both parts. The d and the q subsystems are the following:

$$\frac{d}{dt}\begin{bmatrix} i_{Sd} \\ i_{mR} \end{bmatrix} = \begin{bmatrix} -\frac{R_S}{\sigma L_S} & 0 \\ \frac{1}{T_R} & -\frac{1}{T_R} \end{bmatrix}\begin{bmatrix} i_{Sd} \\ i_{mR} \end{bmatrix} + \begin{bmatrix} \frac{1}{\sigma L_S} \\ 0 \end{bmatrix} u_{Sd}^{Lin}; \qquad y_q = \begin{bmatrix} 0 & 1 \end{bmatrix}\begin{bmatrix} i_{Sq} \\ i_{mR} \end{bmatrix} \tag{23}$$

$$\frac{d}{dt}\begin{bmatrix} i_{Sq} \\ \omega \\ \varepsilon \end{bmatrix} = \begin{bmatrix} -\frac{R_S}{\sigma L_S} & 0 & 0 \\ K_J & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}\begin{bmatrix} i_{Sq} \\ \omega \\ \varepsilon \end{bmatrix} + \begin{bmatrix} \frac{1}{\sigma L_S} & 0 \\ 0 & -\frac{z_p}{J} \\ 0 & 0 \end{bmatrix}\begin{bmatrix} u_{Sq}^{Lin} \\ m_L \end{bmatrix}; \qquad y_d = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}\begin{bmatrix} i_{Sq} \\ \omega \\ \varepsilon \end{bmatrix} \tag{24}$$

After this it is possible to control the motor, provided that we are able to convert all values to field coordinates. This is made by the flux model. The form of the flux model, which is realized here is based on [1] and [7].

After realizing the flux model it is possible to control. The first branch of the control looks like this: There is a controller for i_{mR}, which is in fact a controller for Ψ, the flux. There is a controller for i_{Sd}, which will force the correct i_{mR}. The second branch begins with a controller for ω, which is in turn followed by a controller for i_{Sq}, which will force the correct torque and will realize the correct velocity. The overall control structure is given in Figure 2, with the controllers shown in the upper left corner.

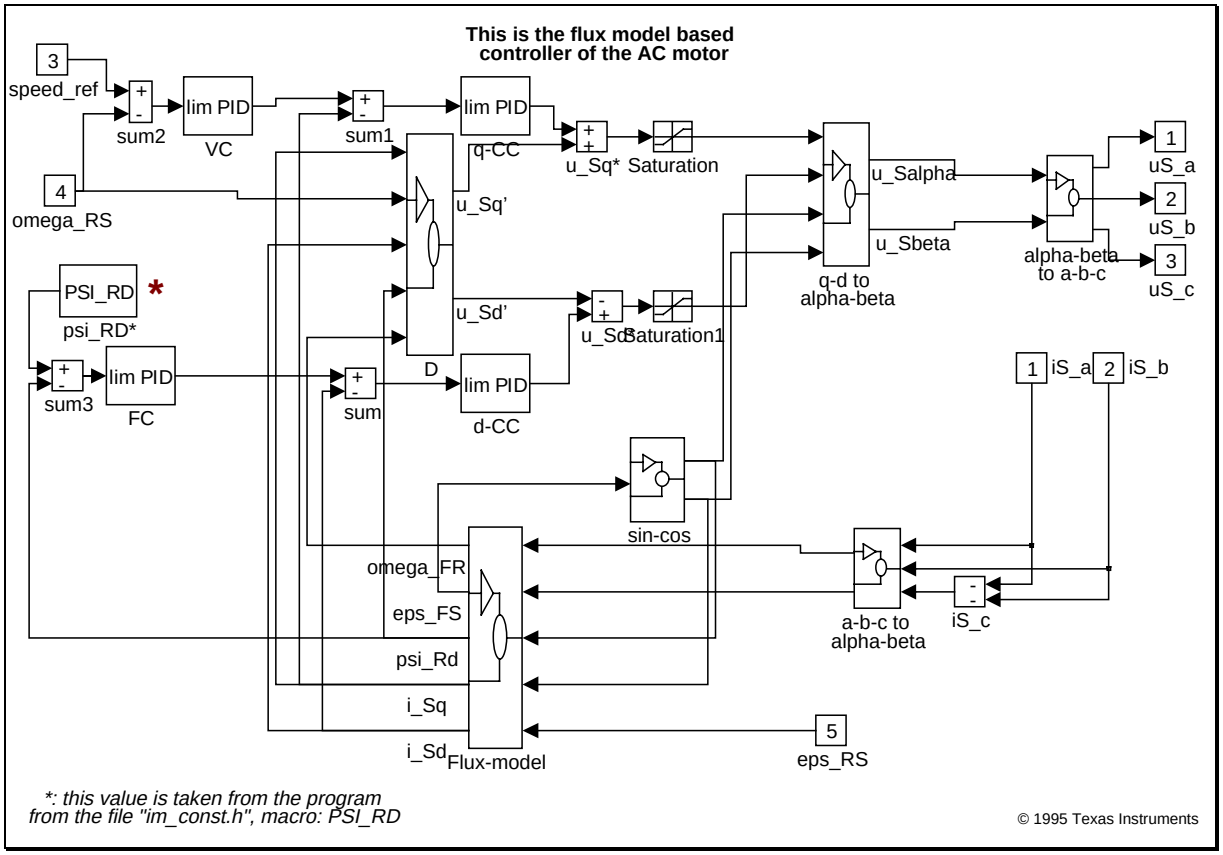

Figure 2: Model 1 - Field Oriented Controller for the ASM

The Field oriented control has been implemented as a simulation in Simulink, and also in real time on a C50 processor. A detailed documentation of this implementation can be found in [5].

6. Kalman Filter

The system we have considered up to now uses a sensor to measure position of the rotor. In many cases it is impossible to use sensors for speed measurement, perhaps because it is either technically impossible or extremely expensive. As an example, we can mention the pumps used in oil rigs to pump out the oil. These have to work under the surface of the sea, sometimes at depths of 50 meters, and getting the speed measurement data up to the surface means extra cables, which is extremely expensive. Cutting down the number of sensors and measurement cables provides a major cost reduction.

Lately, there have been many proposals addressing this problem, and it has turned out that speed can be calculated from the current and voltage values of the AC motor. Some of these proposals are open loop solutions, which give some estimation of speed, but these solutions normally have a large error. For better results we need an observer or a filter. The Kalman filter has a good dynamic behaviour, disturbance resistance, and it can work even in a standstill position. See [11] for a comparison of the performances of an observer, a Kalman filter (KF) and an Extended Kalman filter (EKF).

Implementing a filter is a very complex problem, and it requires the model of the AC motor to be calculated in real time. Also, the Filter equations must be calculated, which normally means many matrix multiplications and one matrix inversion. Nevertheless, these requirements can be fulfilled by a processor with high calculation performance. A DSP is especially well suited for this purpose, because of its good calculation-performance/price ratio. In low cost applications *fixed point* DSPs are desirable.

The chosen solution is a Kalman filter, which is a statistically optimal observer (see exact details in [2]), if the statistical characteristics of the various noise elements are known. For the implementation of the Kalman filter we need a much greater calculating capacity than the C14 used in [6] to realize the field oriented control, so the C50 DSP has been chosen. This makes things more complicated of course, since the variables have to be scaled, which would be unnecessary with a floating point processor.

For information on the hardware used in this project see [5].

At this point, the question of the portability has to be mentioned. The portability of this software is very limited, if we look at the processor side only. It is however a special purpose software for DSPs, so it makes sense to look at the portability only to other DSPs. The fixed point DSPs of Texas Instruments are upward source code compatible, which makes it easy to port the software to newer DSPs. Porting to older versions is also quite simple, the special C50 instructions must be substituted. The software is modular, so porting it to another hardware platform with the same processor but other peripherals only means substituting the I/O routines, and setting the parameters. So we can justly claim that the portability of the software inside the fixed point DSP family is very good.

First a short introduction to the theory of Kalman filters will be presented. This introduction will be based partly on [2], partly on [3].

The Kalman filter is a special kind of observer, which provides optimal filtering of the noises in measurement and inside the system if the covariance matrices of these noises are known. So let us first see what an observer is.

7. Basics of Observers

The problem of the observers is the following. Take a system which has some internal states: these state variables are normally not measurable so we usually measure only substitute variables. If we want to know these internal state variables for some reason, for example, we want to be able to control them, then we have to calculate them. It is not always possible to calculate these variables directly from the measured outputs.

Consider a system with the following form. (Note, that all symbols that denote matrices or vectors are underlined.)

$$\underline{\dot{x}} = \underline{A}\,\underline{x} + \underline{B}\,\underline{u} \tag{25}$$

$$\underline{y} = \underline{C}\,\underline{x} \tag{26}$$

With a very simple approach we can realize a system, that runs parallel to the real system, and it calculates the state vector, as seen in.Figure 3. This is based on the quite reasonable assumption, that we know the input values of the system.

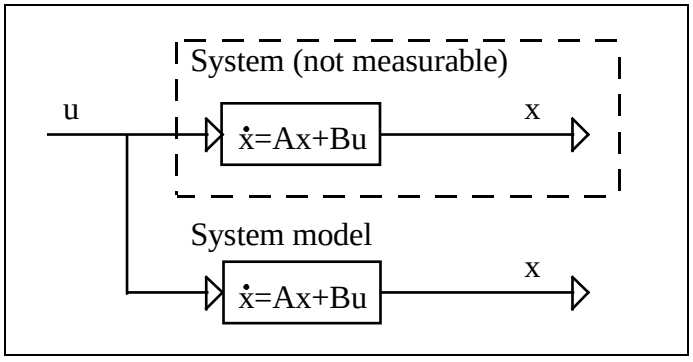

Figure 3: Reconstruction of the State Vector

This approach however does not take into account that the starting condition of the system is unknown, which is true in practically every case. This causes the state variable vector of the system model to be different from that of the real system.

The problem can be overcome by using the principle that that the estimated output vector is calculated based on the estimated state vector,

$$\underline{\hat{y}} = \underline{C}\,\underline{\hat{x}} \tag{27}$$

which may then be compared with the measured output vector. The difference will be used to correct the state vector of the system model. This is called the Luenberger observer, and it can be seen in Figure 4.

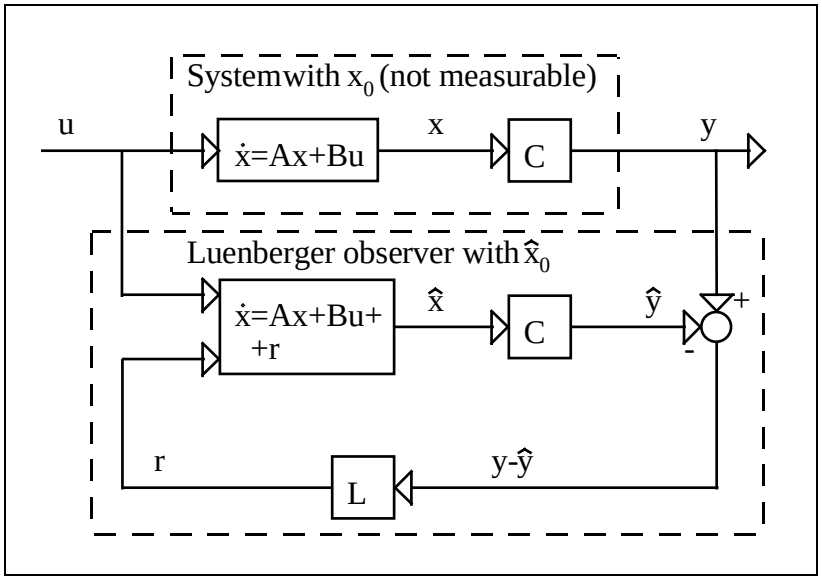

Figure 4: Structure of the Luenberger Observer

Now we can set up the state equation of the Luenberger Observer as the following:

$$\dot{\hat{\underline{x}}} = (\underline{A} - \underline{L}\underline{C})\hat{\underline{x}} + \underline{B}\underline{u} + \underline{L}\underline{y} \tag{28}$$

Now we can ask how the matrix L must be set in order to make the error go to zero. This is done by setting up a state equation for the error as follows:

$$\dot{\tilde{\underline{x}}} = (\underline{A} - \underline{L}\underline{C})\tilde{\underline{x}} \tag{29}$$

where:

$$\tilde{\underline{x}} = \underline{x} - \hat{\underline{x}} \tag{30}$$

If we now transpose the matrix of the error differential equation (29), we get a form which is very similar to a controller structure:

$$\dot{\underline{x}}_f = (\underline{A}^T - \underline{C}^T \underline{L}^T)\underline{x}_f \tag{31}$$

The effectiveness of such an observer greatly depends on the exact setting of the parameters, and the exact measurement of the output vector. In the case of a real system, none of these criteria can be taken for granted. In the event of relatively great disturbances in the measurement, great parameter differences, or internal noises in the system, the Luenberger observer cannot work anymore and we have to turn to the Kalman filter.

8. Basics of Kalman Filters

The Kalman filter provides a solution that directly cares for the effects of the disturbance noises. The errors in the parameters will normally also be handled as noise. Let us assume a system with the following equations.

$$\dot{\underline{x}} = \underline{A}\underline{x} + \underline{B}\underline{u} + \underline{r} \qquad \text{(System)} \qquad (32)$$

$$\underline{y} = \underline{C}\underline{x} + \underline{\rho} \qquad \text{(Measurement)} \qquad (33)$$

Where r and ρ are the system and the measurement noise. Now we assume, that these noises are stationary, white, uncorrelated and Gauss noises, and their expectation is 0. Let us now define the covariance matrices of these noises:

$$\mathrm{cov}(\underline{r}) = E\left\{\underline{r}\underline{r}^T\right\} = Q \qquad (34)$$

$$\mathrm{cov}(\underline{\rho}) = E\left\{\underline{\rho}\underline{\rho}^T\right\} = R \qquad (35)$$

Where $E\{.\}$ denotes expected value.

The overall structure of the Kalman filter is the same as that of the Luenberger observer in Figure 4. The system equations are also the same:

$$\dot{\hat{\underline{x}}} = (\underline{A} - \underline{K}\underline{C})\hat{\underline{x}} + \underline{B}\underline{u} + \underline{K}\underline{y} \qquad (36)$$

We will follow the notations of [2], and denote the matrix of the Kalman filter by K. The only real difference between the Luenberger observer and the Kalman filter is the setting of the matrix K. This will be done based on the covariance of the noises. We will first build the measure of the goodness of the observation, which is the following:

$$J = \sum_{i=1}^{n} E\left\{\tilde{x}_i^2\right\} \qquad (37)$$

This depends on the choice of K. K has to be chosen to make J minimal. The solution of this is the following (see [11]):

$$\underline{K} = \underline{P}\underline{C}^T\underline{R}^{-1} \qquad (38)$$

Where P can be calculated from the solution of the following equation:

$$\underline{P}\underline{C}^T\underline{R}^{-1}\underline{C}\underline{P} - \underline{A}\underline{P} - \underline{P}\underline{A}^T - \underline{Q} = \underline{0} \qquad (39)$$

Q and R have to be set up based on the stochastic properties of the corresponding noises. Since these are usually unknown they are used as weight matrices in most cases. They are often set equal to the unit matrix, avoiding the need of the deeper knowledge of noises.

In [2] a recursive algorithm is presented for the discrete time case to provide the solution for this equation. This algorithm is in fact the EKF (Extended Kalman Filter) algorithm, because the matrix of the Kalman filter, K, will be on-line calculated. The EKF is also capable of handling non-linear systems, such as the induction motor. In this case we do not have the optimum behaviour, which means the minimum variance, and it is also impossible to prove the convergence of the model. (See [11]).

Let us now see the recursive form of the EKF as in [9]. (This is basically the same as in [2], but with slightly different notation):

All symbols in the following formulas denote matrices or vectors. They are not denoted with a special notation, because there is no possibility of mixing them up with scalars.

$$x_{k|k} = x_{k|k-1} + K_k(y_k - h(x_{k|k-1}, k)) \tag{40}$$

$$P_{k|k} = P_{k|k-1} - K_k \left.\frac{\partial h}{\partial x}\right|_{x=x_{k|k-1}} P_{k|k-1} \tag{41}$$

$$K_k = P_{k|k-1} \left.\frac{\partial h^T}{\partial x}\right|_{x=x_{k|k-1}} \left(\left.\frac{\partial h}{\partial x}\right|_{x=x_{k|k-1}} P_{k|k-1} \left.\frac{\partial h^T}{\partial x}\right|_{x=x_{k|k-1}} + R \right)^{-1} \tag{42}$$

$$x_{k+1|k} = \Phi(k+1, k, x_{k|k-1}, u_k) \tag{43}$$

$$P_{k+1|k} = \left.\frac{\partial \Phi}{\partial x}\right|_{x=x_{k|k}} P_{k|k} \left.\frac{\partial \Phi^T}{\partial x}\right|_{x=x_{k|k}} + \Gamma_k Q \Gamma_k^T \tag{44}$$

Where

$$\Phi(k+1, k, x_{k|k-1}, u_k) = A_k(x_{k|k}) x_{k|k} + B_k(x_{k|k}) u_k \tag{45}$$

$$h(x_{k|k-1}, k) = C_k(x_{k|k-1}) x_{k|k-1} \tag{46}$$

These are the system vector and the output vector respectively, and they can be explicitly calculated.

The matrix K is the feedback matrix of the EKF. This matrix determines how the state vector of the EKF is modified after the output of the model is compared with the real output of the system.

At this point it is important to mention that this system of equations contains many matrix operations, which can be difficult to implement in real-time.

To implement this recursive algorithm of course we will need the Model of the motor, which means the matrices A, B and C , from which we have to calculate the matrices Φ and h. So let us see the Motor Model.

9. Motor Model for the Kalman Filter

As shown in the previous section we need a model of the motor for the implementation of the KF. For this purpose two models have been tested, one of Brunsbach [9] and one of Vas [10]. The model of Vas has shown a more stable behaviour, and that is why we use it later for the implementation.

First, let us have a look at the model of [9].

$$\frac{d}{dt}\begin{bmatrix} i_{Sd} \\ i_{mR} \\ i_{Sq} \\ \omega \end{bmatrix} = \begin{bmatrix} -\frac{1}{\sigma}\frac{1}{T_S} - \frac{1-\sigma}{\sigma}\frac{1}{T_R} & \frac{1-\sigma}{\sigma}\frac{1}{T_R} & \omega_{mR} & 0 \\ \frac{1}{T_R} & -\frac{1}{T_R} & 0 & 0 \\ -\omega_{mR} & -\frac{1-\sigma}{\sigma}\omega_{mR}T & -\frac{1}{\sigma T_S} & 0 \\ 0 & 0 & \frac{2}{3}\frac{z_p^2}{J}(1-\sigma)L_S i_{mR} & 0 \end{bmatrix} \begin{bmatrix} i_{Sd} \\ i_{mR} \\ i_{Sq} \\ \omega \end{bmatrix} + \frac{1}{\sigma T_S R_S}\begin{bmatrix} \cos(\varepsilon) & \sin(\varepsilon) \\ 0 & 0 \\ -\sin(\varepsilon) & \cos(\varepsilon) \\ 0 & 0 \end{bmatrix}\begin{bmatrix} u_{S\alpha} \\ u_{S\beta} \end{bmatrix} \tag{47}$$

$$\begin{bmatrix} i_{S\alpha} \\ i_{S\beta} \\ \omega \end{bmatrix} = \begin{bmatrix} \cos(\varepsilon) & 0 & -\sin(\varepsilon) & 0 \\ \sin(\varepsilon) & 0 & \cos(\varepsilon) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} i_{Sd} \\ i_{mR} \\ i_{Sq} \\ \omega \end{bmatrix} \tag{48}$$

Note that in this model ω is part of the output vector. This does not mean that we measure it, but it must be estimated roughly and this estimated value must be substituted into the Kalman filter where this output vector is needed. The substitution is made based on the following formula:

$$\omega = \omega_{mR} - \frac{i_{Sq}}{T_R i_{mR}} \tag{49}$$

This expression has to be calculated each time the model has to be evaluated. To evaluate this formula we need the speed of the flux, ω_{mR} . This means, that we require our flux model just as before. The flux model is also needed to calculate the angle of the flux, ε .

Now we will examine the other model proposed in [10].

$$\frac{d}{dt}\begin{bmatrix} i_{S\alpha} \\ i_{S\beta} \\ \Psi_{R\alpha} \\ \Psi_{R\beta} \\ \omega \end{bmatrix} = \begin{bmatrix} -\frac{K_R}{K_L} & 0 & \frac{L_H R_R}{L_R^2 K_L} & \frac{L_H \omega}{L_R K_L} & 0 \\ 0 & -\frac{K_R}{K_L} & -\frac{L_H \omega}{L_R K_L} & \frac{L_H R_R}{L_R^2 K_L} & 0 \\ \frac{L_H}{T_R} & 0 & -\frac{1}{T_R} & -\omega & 0 \\ 0 & \frac{L_H}{T_R} & \omega & -\frac{1}{T_R} & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} i_{S\alpha} \\ i_{S\beta} \\ \Psi_{R\alpha} \\ \Psi_{R\beta} \\ \omega \end{bmatrix} + \frac{1}{K_L}\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} u_{S\alpha} \\ u_{S\beta} \end{bmatrix} \tag{50}$$

$$\begin{bmatrix} i_{S\alpha} \\ i_{S\beta} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} i_{S\alpha} \\ i_{S\beta} \\ \Psi_{R\alpha} \\ \Psi_{R\beta} \\ \omega \end{bmatrix} \tag{51}$$

This model has a disadvantage; its order is higher. This will be a drawback when the EKF algorithm has to be implemented in real-time. One great advantage of this model, however, is that it does not assume that the speed is measured, so neither ω_{mR} nor ω has to be known. The other is that the flux model can be omitted, since this model also estimates the flux, and so the angle of the flux and any other parameters can be directly calculated. The model is also much simpler than the first one, since it does not contain conversions between the stator and the field coordinate system, and thus the nonlinear sine terms disappear from the input and output matrices.

In both cases we will need a discrete version of the systems. The conversion will be done by the following approximate formulas based on [9]:

$$A' = e^{AT} \approx I + AT \tag{52}$$

$$B' = \int_0^T e^{A\xi} B d\xi \approx BT \tag{53}$$

$$C' = C \tag{54}$$

Where we denoted the system matrix, the input and output matrices of the continuous system with A, B and C, and those of the discrete system with A', B' and C'. We assumed that our sampling time is very short compared with the dynamics of the system. From now on we will use the model presented in [10], so let us see the discrete form of this model.

$$\begin{bmatrix} i_{S\alpha} \\ i_{S\beta} \\ \Psi_{R\alpha} \\ \Psi_{R\beta} \\ \omega \end{bmatrix}_{k+1} = \begin{bmatrix} 1-T\frac{K_R}{K_L} & 0 & T\frac{L_H R_R}{L_R^2 K_L} & T\frac{L_H \omega}{L_R K_L} & 0 \\ 0 & 1-T\frac{K_R}{K_L} & -T\frac{L_H \omega}{L_R K_L} & T\frac{L_H R_R}{L_R^2 K_L} & 0 \\ T\frac{L_H}{T_R} & 0 & 1-T\frac{1}{T_R} & -T\omega & 0 \\ 0 & T\frac{L_H}{T_R} & T\omega & 1-T\frac{1}{T_R} & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} i_{S\alpha} \\ i_{S\beta} \\ \Psi_{R\alpha} \\ \Psi_{R\beta} \\ \omega \end{bmatrix}_k + T\frac{1}{K_L} \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} u_{S\alpha} \\ u_{S\beta} \end{bmatrix}_k \tag{55}$$

$$\begin{bmatrix} i_{S\alpha} \\ i_{S\beta} \end{bmatrix}_k = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} i_{S\alpha} \\ i_{S\beta} \\ \Psi_{R\alpha} \\ \Psi_{R\beta} \\ \omega \end{bmatrix}_k \tag{56}$$

After this point we will only look at the second model because the delivered results were much more stable.

10. Simulation of the Kalman Filter

Now that we have the discrete form of the model, we can calculate the necessary matrices and vectors for the recursion. This will be the last step to enable both real-time implementation and simulation.

$$\Phi = \begin{bmatrix} (1-T\frac{K_R}{K_L})i_{S\alpha} + T\frac{L_H R_R}{L_R^2 K_L}\Psi_{R\alpha} + T\frac{L_H \omega}{L_R K_L}\Psi_{R\beta} + T\frac{1}{K_L}u_{S\alpha} \\ (1-T\frac{K_R}{K_L})i_{S\beta} - T\frac{L_H \omega}{L_R K_L}\Psi_{R\alpha} + T\frac{L_H R_R}{L_R^2 K_L}\Psi_{R\beta} + T\frac{1}{K_L}u_{S\beta} \\ T\frac{L_H}{T_R}i_{S\alpha} + (1-T\frac{1}{T_R})\Psi_{R\alpha} - T\omega\Psi_{R\beta} \\ T\frac{L_H}{T_R}i_{S\beta} + T\omega\Psi_{R\alpha} + (1-T\frac{1}{T_R})\Psi_{R\beta} \\ \omega \end{bmatrix} \tag{57}$$

$$h = Cx = \begin{bmatrix} i_{S\alpha} \\ i_{S\beta} \end{bmatrix} \tag{58}$$

$$\frac{\partial \Phi}{\partial x} = \begin{bmatrix} 1-T\frac{K_R}{K_L} & 0 & T\frac{L_H R_R}{L_R^2 K_L} & T\frac{L_H \omega}{L_R K_L} & T\frac{L_H}{L_R K_L}\Psi_{R\beta} \\ 0 & 1-T\frac{K_R}{K_L} & -T\frac{L_H \omega}{L_R K_L} & T\frac{L_H R_R}{L_R^2 K_L} & -T\frac{L_H}{L_R K_L}\Psi_{R\alpha} \\ T\frac{L_H}{T_R} & 0 & 1-T\frac{1}{T_R} & T\omega & T\Psi_{R\beta} \\ 0 & T\frac{L_H}{T_R} & T\omega & 1-T\frac{1}{T_R} & T\Psi_{R\alpha} \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{59}$$

$$\frac{\partial h}{\partial x} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix} \tag{60}$$

Reaching this point the realization of the model in Matlab/Simulink can begin. Realizing the complete Kalman filter as a Simulink model would have been a very complicated model, and it seemed easier to implement it simply as a Matlab language file. Another advantage of this is that the Matlab language file can be more easily converted into an assembly program. A subsystem has been inserted into the system that contains the Kalman filter, which is then bound into the model as an S-function.

The overall structure of the controller is not changed very much. See Figure 5: Model 2. The filter is in the subsystem called KF. Its output depends on which model we use. The picture shows the case of the EKF with the second motor model (from [10]). In this case,

the outputs of the system are all state variables, which is the rotor fluxes and stator currents and rotor speed. The inputs are measured rotor currents and rotor voltages. In the other case the output would be only rotor speed, but the estimated rotor speed would be needed as well, which would be calculated by a flux model.

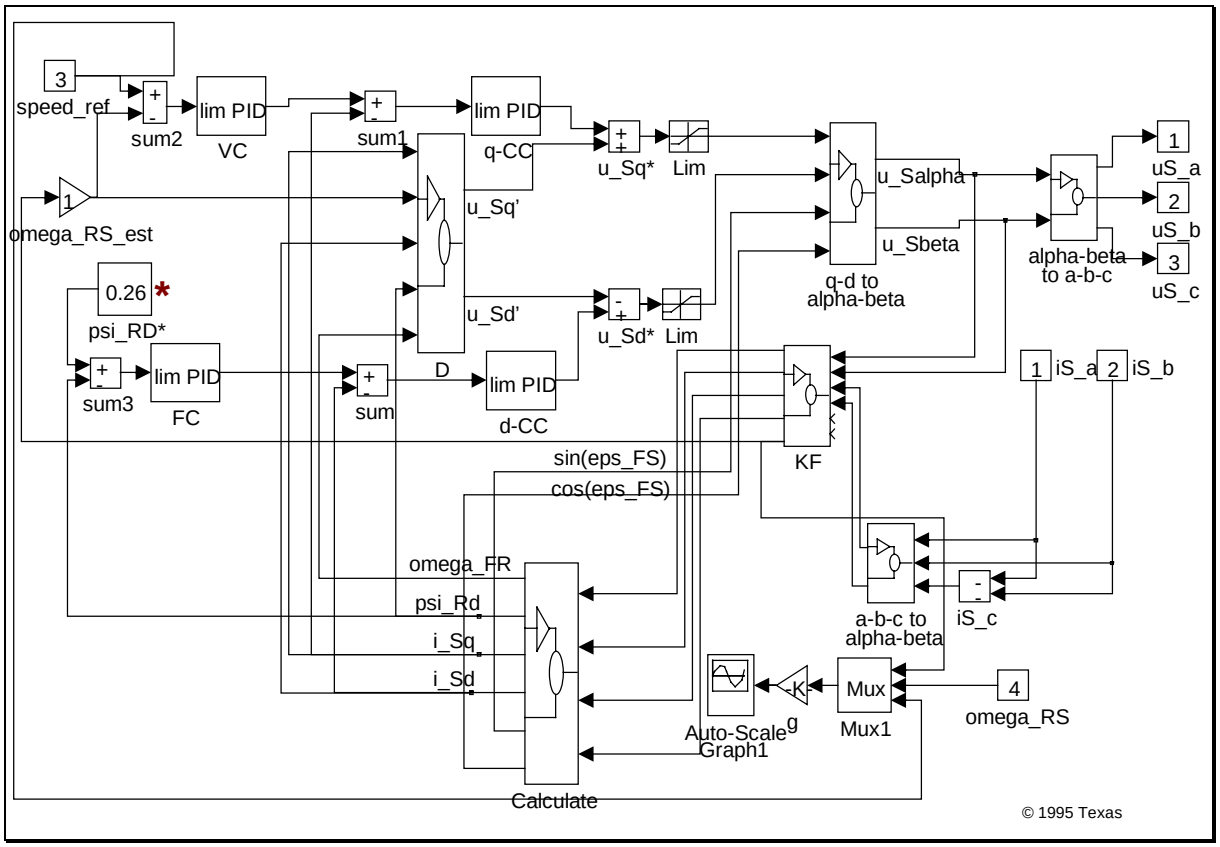

Figure 5: Model 2 - Controller with Kalman Filter

These tests reveal that the model of Vas (Presented in [10]) has a much more stable behaviour. In the Appendix the Extended Kalman Filter S-functions are presented for both motor models. For test results with the motor model please refer to the section "11 Simulation Results".

After achieving the required simulation results the real-time implementation could begin, again based on the second model.

11. Simulation Results

In this section the simulation results of the Field Oriented Control (FOC) and the EKF (Model of Vas) will be compared.

As a first test, let us compare the speed reversal of an FOC and an EKF. In both cases we apply a speed command of 2000RPM, and -2000RPM at 0.1 and 2.2s respectively. Also a constant load is applied at 1s in positive, and at 3s in negative direction.

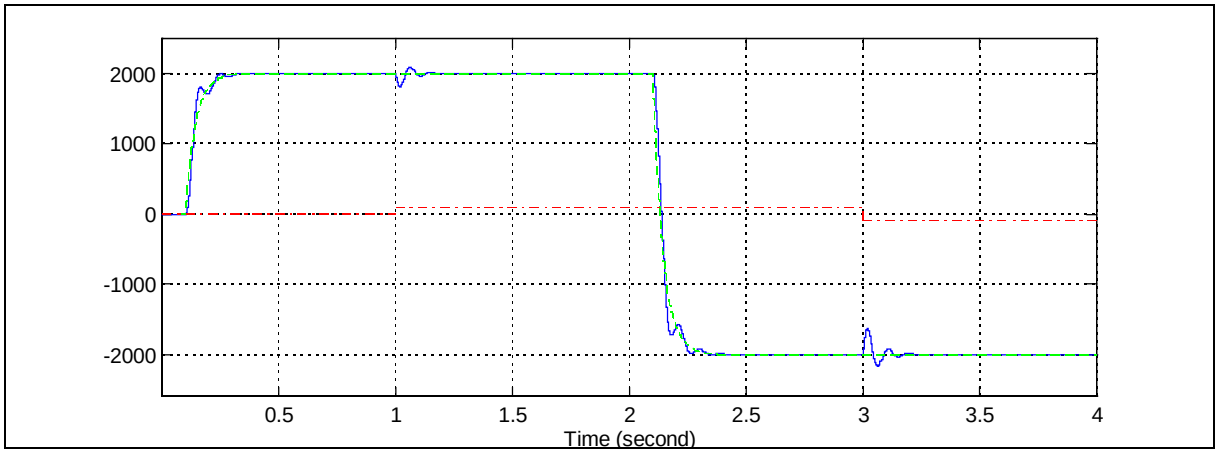

Figure 6: Speed Reversal with FOC

The figure shows the speed reference, speed and the load torque

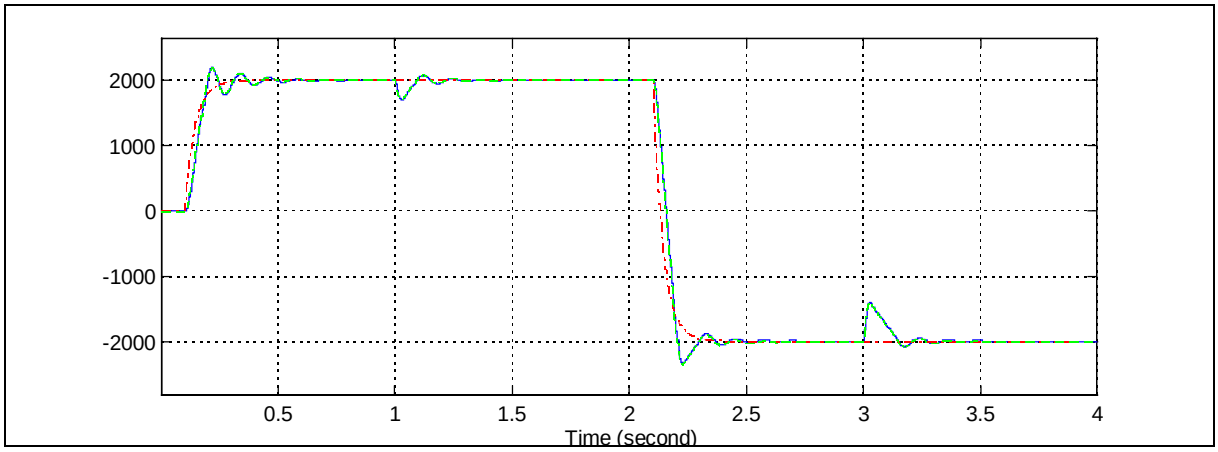

Figure 7: Speed Reversal with EKF

The figure shows the speed reference, estimated speed and the speed.

We can observe that the response of the EKF is worse than of the FOC, which is natural because we do not have an exact speed measurement, but the difference is not that great.

The next experiment shows load torque pulses applied to the motor in a standstill position. The torque is very large, so it causes very big changes in the speed. Let us first see, how the FOC behaves.

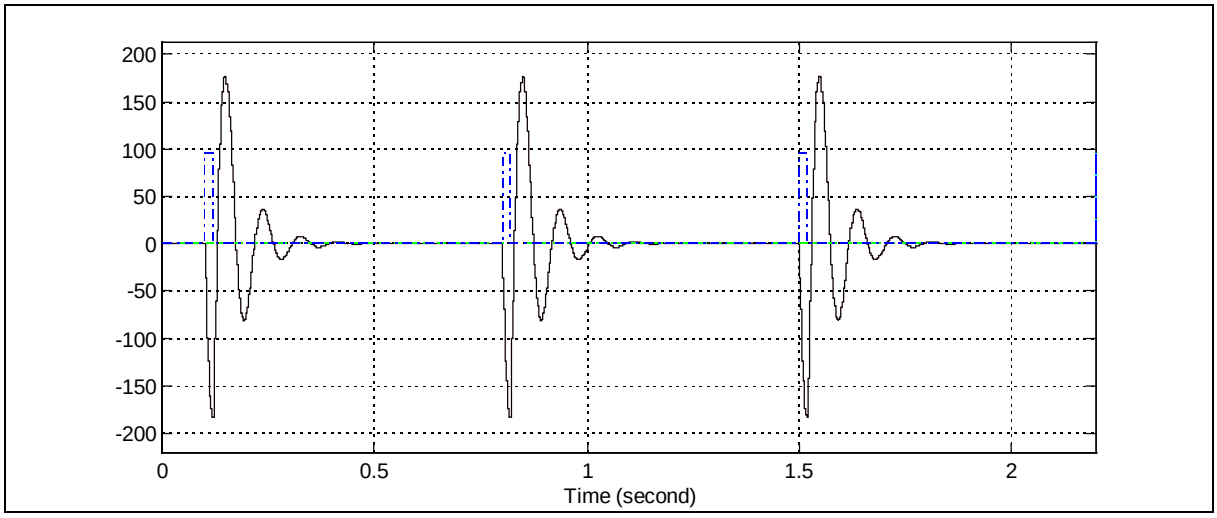

Figure 8: Applying Load at 0 RPM with FOC

The figure shows speed, torque and speed reference (always 0).

Let us now see the same test with the EKF.

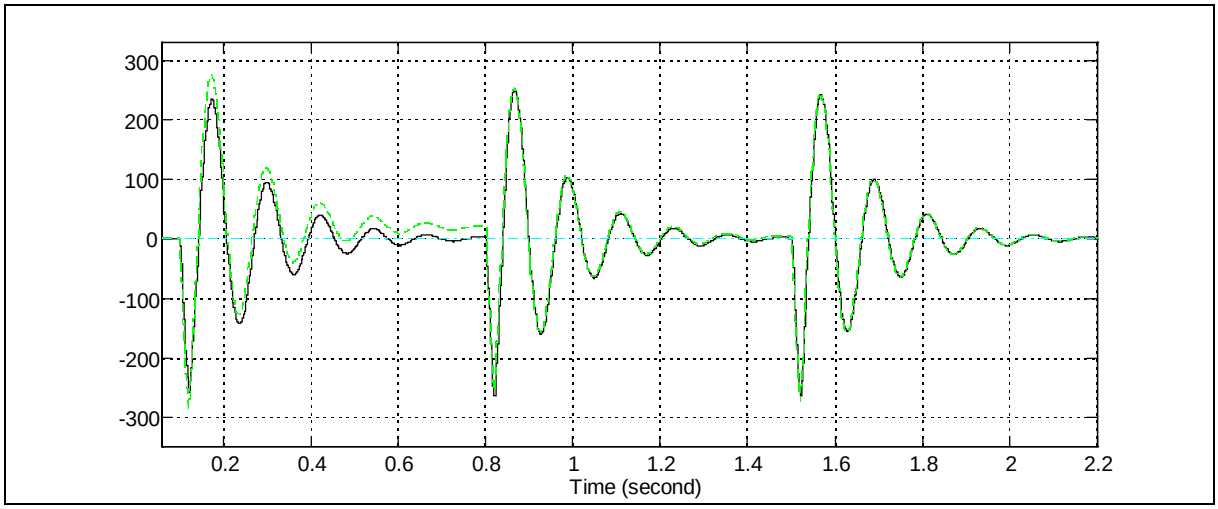

Figure 9: Applying Load at 0 RPM with EKF

It is clear that the offset gets very small after the second, and even smaller after the third, load impulse. This is due to the fact that the EKF parameters must settle before it can deliver good results and it needs some changes to be able to settle. This settling means that the K matrix reaches a point where it is more or less constant and its performance is close to optimal. Typical values of the K matrix after the first torque impulse:

```
K =
    0.1217   -0.0245
```

```
   -0.0245    0.0049
   -0.0027    0.0005
    0.0005   -0.0001
    9.1643   -1.8304
```

This value leaves quite a big offset in the speed.

After the second torque impulse the offset will be very small, and K gets the following value:

```
K =
    0.0670    0.0632
    0.0632    0.0597
   -0.0015   -0.0014
   -0.0014   -0.0013
    6.8152    6.3945
```

This shows that there are great changes in K. As the system is nonlinear, we have to see that K will change all the time and will adapt the actual system conditions.

As a conclusion we can say that the simulation of the EKF shows a stable behaviour after a certain time has passed for settling. The torque disturbance rejection is very good, and comparable to the FOC. The simulations provide a basis for the real time implementation.

12. Real-Time Implementation of the Kalman Filter

In this chapter the real-time implementation using the TMS320C50 DSP will be presented.

The calculations look quite simple in the Matlab S-function, but we should not forget that all operations are matrix operations. Additionally there is a matrix inversion in the calculation process, which is very complex in the general case. These manipulations have to be ported to the assembly language of the C5x. Since the language has a very good macro support, these functions have been implemented with the help of macros.

To implement the matrix calculations, some matrix manipulation macros are needed. The most frequently needed was matrix multiplication. Let us see the matrix multiplication macro as an example.

```
; Matrix Multiplication Macro
;               Author :    Balazs Simor
;               Date :      09. 1995.
; matrix multiplication for fractional matrices
; the macro does mat1*mat2=reslt.
; sizes of the matrices:
; mat1: [s11 x s21], mat2: [s21 x s22], reslt: [s11 x s22]
```

```
mmfra  .macro  mat1, mat2, reslt, s11, s21, s22
       setc ovm                ; saturation mode on.
       spm  1                  ; product shifted left by 1
       .asg 0,i
       .loop
       .asg 0,j
          .loop
             zap
             .asg 0,k
             .loop
                lta mat1+i*s21+k
                mpy mat2+k*s22+j
                .eval k+1,k
                .break(k == s21)
             .endloop
             apac
             sach reslt+s22*i+j
             .eval j+1,j
             .break(j == s22)
          .endloop
          .eval i+1,i
          .break (i == s11)
       .endloop
       clrc ovm                ; saturation mode off
       spm 0                   ; shifting off
       .endm
```

This macro implements a very general [nxm]*[mxl] matrix multiplication in a macro language. This has a great advantage, the cycles are generated and expanded by the compiler, and they do not take computation time. This macro relies basically upon the macro support of the Fixed Point DSP Assembler, for more information of the macro language see [13]. Note, this is a simplified version of the macro, but it shows the basic idea. For the complete macro see the Appendix.

The main reason why this function is implemented as a macro is that we need to multiply many different sized matrices. There is a list of the matrix multiplication operations needed in the following table. Since there are so many of these it did not make sense to implement each multiplication separately, and using loops would have made the calculations much slower. The macro language can handle the loops at compile time and achieve faster calculation.

Table 2: Some Types of Matrix Multiplication Needed For Kalman Filter

Matrix1 * Matrix2	Result
[5x5]*[5x5]	[5x5]
[2x5]*[5x5]	[2x5]
[2x5]*[5x2]	[2x2]
[5x2]*[2x2]	[5x2]
[5x5]*[5x2]	[5x2]
[5x2]*[2x1]	[5x1]
[5x2]*[2x5]	[5x5]

As an example for the speed of this macro we take a simple case of a [3x3]*[3x2] matrix multiplication. The macro will be expanded into 54 instructions, and the time needed for the execution will be 65 cycles. This is a measured value with the emulator. To see how to measure speed of programs with the TI tools, see [12] ("runb" debugger command).

Other matrix operations that are needed are: Inverting, Addition, Transposition, Vector normalization.

Inverting the matrix will be done by the Cramer rule:

$$A^{-1} = \frac{1}{det(A)} adj(A) \tag{61}$$

Since we need the inverse of only a [2x2] matrix, the values can be explicitly calculated. Inverting has also been implemented in macros.

The macros available for inversion of a matrix are "madjG" for calculating the adjunct matrix, and "detG" for calculating the determinant of a matrix. To divide by the determinant, it is advisable to use the "mdsclG" macro. This macro divides a matrix by a scalar. Note, that the inverse of a matrix with elements in (-1,1) usually has elements out of this range. This means special care must be taken when calculating inverse.

The “mdsclG” macro makes a division of each element of a matrix by a scalar. This is usually a manipulation, where we lose a lot of accuracy, sometimes the complete result is incorrect. To overcome this problem, a “quasi floating point” division has been implemented. This means the scalar number is scaled so that its absolute value is in the range (0.5,1). Let us say that the scaling factor is 2^k. Then its “reciprocal” is calculated by dividing 0.5 by this scaled value, which is in fact $\frac{0.5}{2^k scl}$ and in the range of (0.5,1), which means it is well scaled. The elements of the matrix are then calculated with this manipulated reciprocal and then left shifted by k+1. This method has the advantage that the “reciprocal” has the maximum accuracy achievable with 16 bits. The whole process can be expressed by the following formula:

$$\frac{a_{i,j}}{scl} = \frac{0.5}{2^k scl} a_{i,j} 2^{k+1} \tag{62}$$

where scl is the scalar, and $a_{i,j}$ are the elements of the matrix.

This method can be called “quasi floating point”, because the reciprocal is in a form, where $\frac{0.5}{2^k scl}$ is its mantissa, and k+1 its exponent.

The addition and transposition are also realized in very simple macros. There are versions of these macros, that work with loops to make the need of program memory smaller.

There is a significant problem in storing the matrices. The macros can operate correctly only if the operand(s) and the result are on the same data page. We have quite a few matrices and a [5x5] matrix takes up as much as 25 words place. A data page has the size of 128 words, and we have 12 vectors and matrices of various sizes, which do not fit this page. The necessary space can be reduced if the transposed matrices are not calculated explicitly but a multiplication by transposed macro is created. This way diff_FI transposed ($\frac{\bar{\sigma}\ \Phi^T}{\bar{\sigma}\ x}$), and diff_h transposed ($\frac{\bar{\sigma}\ h^T}{\bar{\sigma}\ x}$) do not have to be calculated. This also reduces the time needed for calculation. But the space taken up is still more than can fit on one page, so macros had to be created that can operate across pages. These macros use auxiliary registers of the DSP rather than direct addressing.

An interesting problem is normalizing vectors. This problem is encountered as the program has to calculate the transformation to field coordinates from the components of the rotor flux. The “vnormG” macro is available for this purpose. The macro works only with [2x1] vectors. The macro uses “quasi floating point” operations similar to the “mdsclG”. For this calculation the Newton-Raphson approximation of the square root is used, see [19] for short practical information.

Another important problem is that all the calculations have to be converted to fractional. This means that the ranges of the various parameters have to be known and scaled into the range of (-1,1). This is a difficult problem to handle in the case of the matrices which can have elements of very different ranges. Let us now see the scaling values for the various values in the matrices.

Table 3: Scaling Factors of Variables

Variable	Scaling Factor and Dimension
$i_{Sd}, i_{mR}, i_{Sq}, i_\alpha, i_\beta$	22.5 [A]
$\omega(=\omega_{RS}), \omega_{mR}(=\omega_{FS}), \omega_{FR}$ (electrical speed)	$2\pi\frac{n_{\max}}{60}z_p\left[\frac{rad}{s}\right]$[5]
ω_M, ω_{M_m} (mechanical speed)	$2\pi\frac{n_{\max}}{60}\left[\frac{rad}{s}\right]$
$\varepsilon_{RS}, \varepsilon(=\varepsilon_{FS}), \varepsilon_{FR}, \varepsilon_M$	$\pi[rad]$
$u_{Sd}, u_{Sq}, u_\alpha, u_\beta$	155 [V]

The constants contained in the equations are presented in the following table.

Table 4: Constant Values

Constant	Value
T	500 µs
T_S	74.4 ms
T_R	40 ms
σ	0.084
R_S	1.68 Ω
L_S	0.125 H
J	0.00028 kg·m²

Using these values the scaled vector Φ and the matrix $\frac{\partial \Phi}{\partial x}$ can be calculated. All the constants have to be pre-calculated to make the execution optimal. Here are these matrices:

$$\frac{\partial\Phi}{\partial x}=\begin{bmatrix}0.6067 & 0 & 0.0562 & 0.7063\omega & 0.7063\Psi_{R\beta}\\ 0 & 0.6067 & -0.7063\omega & 0.0562 & -0.7063\Psi_{R\alpha}\\ 0.0559 & 0 & 0.9875 & -0.1571\omega & -0.1571\Psi_{R\beta}\\ 0 & 0.0559 & 0.1571\omega & 0.9875 & 0.1571\Psi_{R\alpha}\\ 0 & 0 & 0 & 0 & 1\end{bmatrix} \quad (63)$$

[5] With $n_{max}=3000\frac{1}{s}$ and $z_p=2$.

We can see, that $\frac{\partial \Phi}{\partial x}$ is almost constant, and only 8 of its 25 elements have to be calculated each time. Since $\frac{\partial \Phi}{\partial x}$ contains the constants needed to calculate Φ, first $\frac{\partial \Phi}{\partial x}$ is calculated, and then Φ. This saves space since we have to reserve space for these constants only once - and also time, since products such as 0.7063ω have to be calculated only once in one recursion step.

13. Real Time Results

In this section the real-time results of the field oriented control and the EKF will be presented.

The program was tested with a method where all calculations have been compared with the formulas, thus checking the correctness of the implementation. After this, a 1 to 1 test has been made to compare the results with the ones without Kalman filter. The results were quite similar.

The results are quite similar to the ones like the simulated results. In this case also the standard test was used, the speed reversal test. Speeds mean electrical speed, so 1000 RPM corresponds to 500 RPM mechanical with a machine with two pole pairs (p=2).

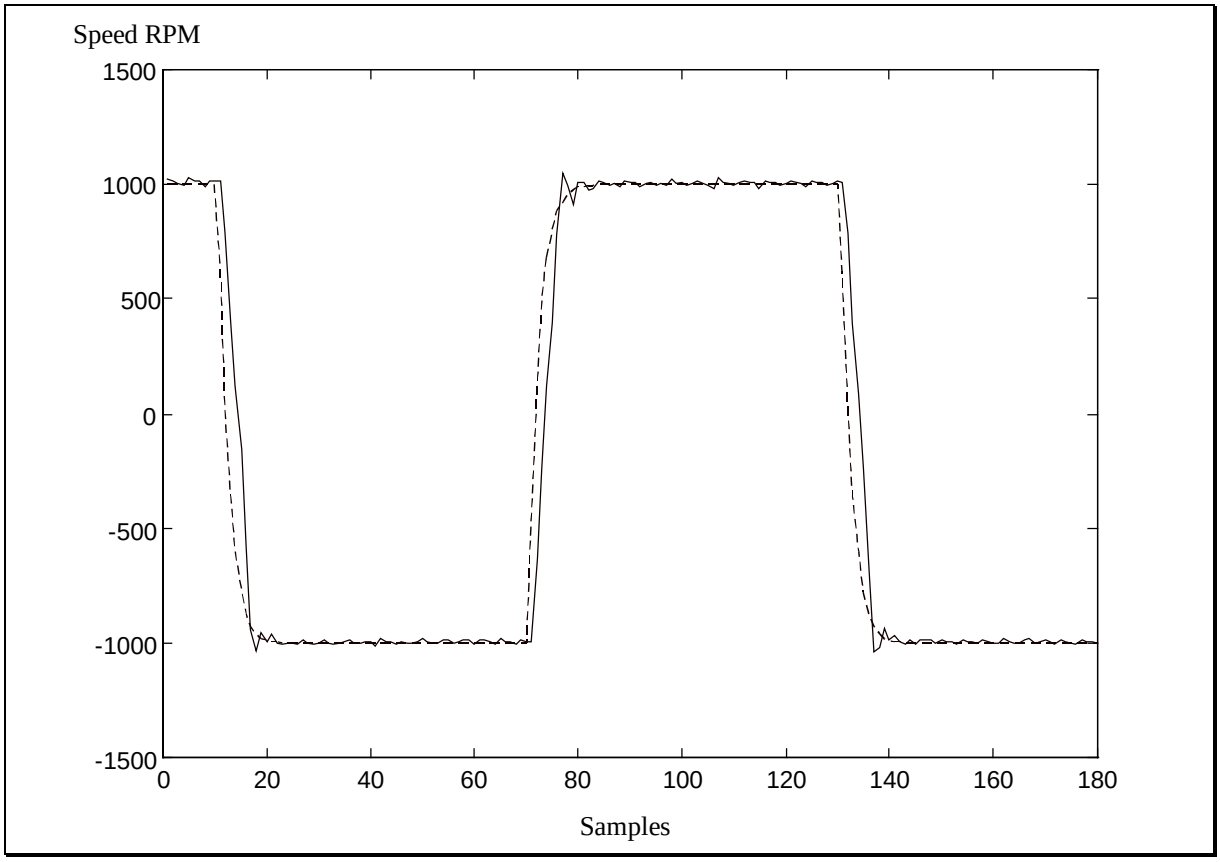

Figure 10: Speed Reversal with Kalman Filter

Note, the speed has a ripple on it, which shows that the behaviour should be improved. This may be achieved by tuning the Q and R covariance matrices. Another important factor is that the motor parameters have not been identified with the necessary tolerance, and much improvement can be expected here. Also note, this test, just as in case of the field oriented control, is without load torque.

As a comparison let us see the same experiment with the same controller settings with speed measurement.

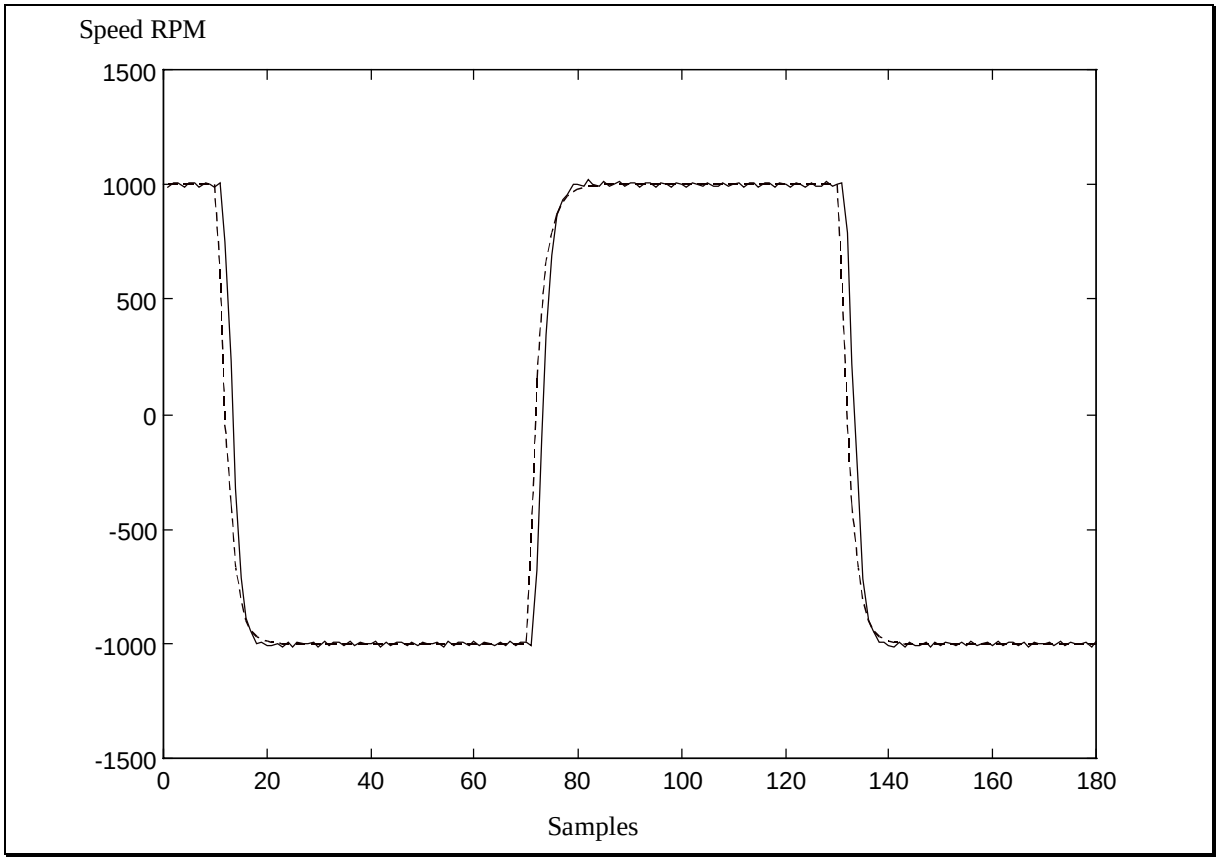

Figure 11: Speed Reversal with Speed Measurement

The program needs relatively little memory, the following table summarizes the needs:

Table 5: Memory Requirements of Kalman Filter

Program Part	Program Size (Words)	Data Size (Words)
Control+Kalman Filter	3641	631
Monitor Program	1577	603
Libraries	1605	309
Stack	0	1024
Σ Memory requirement	6823	2564

The Processor has a computing power of 20 MIPS at 20 MHz, which means a cycle time of 50 ns. The computation of the control happens in a 500 μs cycle, so the processor has 10,000 cycles available. The processor is currently using about 4400-4700 cycles and this means that it has time to perform foreground tasks, such as Monitor programs, or

other communications. The processor computing capacity is used to about 50%, but the cycle times could also be reduced.

It should be mentioned here that further optimization in memory requirement and speed can be made by transferring more code directly to assembly language.

14. Conclusions and Possible Development

A model for the system has been prepared for comparison with the on-line results, and further development purposes. This model has been tested.

After that, the Kalman filter has been integrated to the simulation, in the form of a Matlab function, and it has been tested. The nonlinear Kalman filter algorithm based on [9] has been tested in the model. After the correct system model was chosen for the filter, the results were satisfactory.

An I/O library has been written to support the programming on board, which has then been extensively used in all software development.

The implementation of the Kalman filter has been done after this. Here the greatest problem was to keep the size of the program reasonable, and still reach a good performance. This was achieved by further optimizing the model with hand calculations to get a form of the EKF (Extended Kalman Filter) algorithm which can be implemented with relatively few instructions.

A further phase of the development could be fine tuning the Q and R matrices, which are the covariance matrices of the state and measurement noises and are used in the EKF. This fine tuning is usually done by experiments, see [11]. Another important improvement should be the possibility of exact identification of the system parameters.

There would be a need to convert this software partly into standard libraries. The libraries could provide the possibility to implement various control methods quickly, without low-level programming. It would be possible to overcome the slowness of C by using hand optimized assembly routines for frequent tasks, such as filtering or PID controllers. A library of fast and effective matrix manipulation in form of C callable functions or assembly macros would also be a very important step, which could be done enhancing the matrix manipulation macros in MATG.INC presented in the Appendix.

References

1. Leonhard: Control Of Electrical Drives, Springer-Verlag 1985
2. Brammer, Siffling: Kalman-Bucy Filter, Deterministische Beobachtung und stochastische Filterung. R. Oldenbourg Verlag Muenchen, Wien 1994.
3. Otto Föllinger: Regelungstechnik. Hüthig Buch Verlag GmbH Heidelberg, 1992.
4. Stefan Beierke: Vergleichende Untersuchungen von unterschiedlichen feldorientierten Lagereglerstrukturen fuer Asynchron-Servomotoren mit einem Multi-Transputer-System. Dissertation, Technische Universitaet Berlin 1992.
5. Balázs Simor: Design and Implementation of a Kalman Observer for an Induction Motor Based on a DSP System. Technical University of Budapest, 1995.
6. Dipl.-Ing. R. Neufeld, Dipl.-Ing. F.-J. Burs, Dr.-Ing. P.Krafka: Regelung einer Asynchronmaschine unter Verwendung des Signalprozessors TMS320E14, Universitaet Paderborn, 1993.
7. dSpace GmbH: Induction Motor Control with SIMULINK, dSpace, 1994 Preliminary Document Version 1.0
8. Dierk Schroeder, Clemens Schaeffner, Ulrich Lenz: Neural Net Based Observers for Sensorless Drives. IECON 1994, Volume 3.
9. B.-J. Brunsbach, G. Henneberger: Einsatz eines Kalman-Filters zum feldorientierten Betrieb einer Asynchronmaschine ohne mechanische Sensoren. Archiv fuer Elektrotechnik 1990 (Springer Verlag).
10. T. Du, P. Vas, A.F. Stronach, M.A. Brdys: Application of Kalman Filters and Extended Luenberger Observers in Induction Motor Drives. Intelligent Motion Proceedings, 1994.
11. C. Manes, F. Parasiliti, M. Tursini: Comparative Study of Rotor Flux Estimation in Induction Motors with a Nonlinear Observer and the Extended Kalman Filter. IECON 1994.
12. TMS320C5x C Source Debugger. User's Guide. Texas Instruments, 1991.
13. TMS320 Fixed-Point DSP Assembly Language Tools. User's Guide. Texas Instruments, 1991.
14. TMS320C2x/C5x Optimizing C Compiler. User's Guide. Texas Instruments, 1991.
15. TMS320C5x User's Guide. Texas Instruments, 1991.
16. SIMULINK (V1.3c). User's Guide. The Math Works, Inc. 1993.
17. MATLAB (V4.2c1). User's Guide. The Math Works, Inc. 1993.
18. Linear Circuits. Data Book Volume 2, Texas Instruments, 1992.
19. TMS320C5x User's Guide. Texas Instruments, 1995.

Appendix Source Code of Programs

Extended Kalman Filter in MATLAB Language

```
; Extended Kalman Filter as a Simulink S-function
;     Author : Balazs Simor
;     Date :      November, 1995
```

```
function [sys, x0] = kalfil(t,x,u,flag, T)
%kalman filter as an S-function (m-file)
global Tr Ts Lr Ls Lh Kl Kr Rs Rr p J GAM Q R;
global sig sigs out;
global Tr_K Ts_K Lr_K Ls_K Lh_K Kl_K Kr_K Rs_K Rr_K p_K J_K;
global sig_K sigs_K;
global x_1 P_1 K P h FI Y omrs epsmr epsmr1 ommr co ommrfil isq omrsest;
if flag == 0
      kalini5
         x0 = zeros(5,1);
         sys = [0, 5, 5, 6, 0, 0];
elseif flag == 2
         % calc. inp and out vec of kalman filter
         U = [u(1); u(2)];
         Y = [u(3); u(4)];
         %prediction
         diff_FI=[1-Kr_K/Kl_K*T, 0, Lh_K*Rr_K/Lr_K/Lr_K/Kl_K*T, Lh_K/Lr_K*x(5)/Kl_K*T,
Lh_K/Lr_K*x(4)/Kl_K*T;
                  0, 1-Kr_K/Kl_K*T, -Lh_K/Lr_K*x(5)/Kl_K*T, Lh_K*Rr_K/Lr_K/Lr_K/Kl_K*T,-
Lh_K/Lr_K*x(3)/Kl_K*T;
                  Lh_K/Tr_K*T, 0, 1-T/Tr_K, -x(5)*T, -x(4)*T;
                  0, Lh_K/Tr_K*T, x(5)*T, 1-T/Tr_K, x(3)*T;
                  0, 0, 0, 0, 1];
         x_1=[diff_FI(1,1)*x(1)+diff_FI(1,3)*x(3)+diff_FI(1,4)*x(4);
             diff_FI(2,2)*x(2)+diff_FI(2,3)*x(3)+diff_FI(2,4)*x(4);
             diff_FI(3,1)*x(1)+diff_FI(3,3)*x(3)+diff_FI(3,4)*x(4);
             diff_FI(4,2)*x(2)+diff_FI(4,3)*x(3)+diff_FI(4,4)*x(4);
             diff_FI(5,5)*x(5)]...
           +T*[u(1)/Kl_K; u(2)/Kl_K; 0; 0; 0];
         P_1=diff_FI*P*diff_FI' + GAM*Q*GAM'; %P[k|k-1] is ready
         % calculation of h, diff_h
         h=[x(1); x(2)];
         diff_h=[1 0 0 0 0
                 0 1 0 0 0];
```

```
        % system of Kalman filter
        K=P_1*diff_h'* inv(diff_h*P_1*diff_h' +R); %K[k]
        out = x_1+K*(Y-h);      %x[k] is ready
        sys = out;
        P=P_1-K*diff_h*P_1;     %P[k] is ready
elseif flag == 3
        sys = out;
elseif flag == 4
        sys = (round(t/T)+1)*T;
else
        sys =[];
end
```

Full Source Code of Motor Control with EKF for C50

```
; Kalman Filter for the TMS320C5x
; Makefile
;     Author : Balazs Simor
;     Date :      June, 1996
kalman.out : vel_ctr.obj ctr.obj sin.obj
       dsplnk $** link.cmd -o $@ -m map

matp.out : matp.obj
       dsplnk $** link.cmd -o $@ -m map

.c.asm :
       dspcl -O -g -n $*

.asm.obj :
       dspa -l -v50 $*
```

```
; Kalman Filter for the TMS320C5x
; matg.inc
;     Author : Balazs Simor
;     Date :     June, 1996
; Matrix operation macros for C5x with Global mem access (With auxilliary
; registers of the processor)
; 1995 Balazs Simor

       .mmregs

;matrix multiplication for fractional matrices
; the macro does mat1*mat2=reslt.
; reslt must not be mat1 or mat2.
; sizes: mat1: [s11 x s21], mat2: [s21 x s22], reslt: [s11 x s22]
mmfraG  .macro  mat1, mat2, reslt, s11, s21, s22
       setc ovm        ; saturation mode on.
       spm  1          ;product shifted left by 1
       lar     AR0,#s22
       mar     *,AR2
       .if (s21 >= 3)
       .asg    0,i
       .loop
          .asg       0,j
          .loop
             lar     AR2, #mat1+i*s21+0
             lar     AR3, #mat2+0*s22+j
             lt      *+,AR3
             mpy     *0+,AR2
             ltp     *+,AR3
             mpy     *0+,AR2
             ;lt  mat1+i*s21+0
             ;mpy mat2+0*s22+j
             ;ltp mat1+i*s21+1
             ;mpy mat2+1*s22+j
             .asg 2,k
             .loop
                lta *+,AR3
                mpy *0+,AR2
                ;lta mat1+i*s21+k
                ;mpy mat2+k*s22+j
                .eval k+1,k
                .break(k == s21)
```

```
        .endloop
        apac
        lar     AR2,#reslt+s22*i+j
        add     #16384,1
        sach    *
        .eval j+1,j
        .break(j == s22)
    .endloop
    .eval i+1,i
    .break (i == s11)
.endloop
.else
.asg    0,i
.loop
    .asg        0,j
    .loop
        zap
        lar     AR2, #mat1+i*s21+0
        lar     AR3, #mat2+0*s22+j
        .asg 0,k
        .loop
            lta *+,AR3
            mpy *0+,AR2
            .eval k+1,k
            .break(k == s21)
        .endloop
        apac
        lar     AR2,#reslt+s22*i+j
        add     #16384,1
        sach *
        .eval j+1,j
        .break(j == s22)
    .endloop
    .eval i+1,i
    .break (i == s11)
.endloop
.endif
clrc ovm        ; saturation mode off
spm 0           ;shifting off
.endm
```

;matrix multiplication for fractional matrices with transposition

```
; the macro does mat1*mat2'=reslt.
; reslt must not be mat1 or mat2.
; sizes: mat1: [s11 x s21], mat2: [s22 x s21], reslt: [s11 x s22]
mmtfraG  .macro  mat1, mat2, reslt, s11, s21, s22
       setc ovm        ; saturation mode on.
       spm  1          ;product shifted left by 1
       mar     *,AR2
       .if (s21 >= 3)
       .asg    0,i
       .loop
          .asg       0,j
          .loop
             lar     AR2,#mat1+i*s21+0
             lar     AR3,#mat2+j*s21+0
             lt      *+,AR3
             mpy     *+,AR2
             ltp     *+,AR3
             mpy     *+,AR2
             ;lt  mat1+i*s21+0
             ;mpy mat2+j*s22+0
             ;ltp mat1+i*s21+1
             ;mpy mat2+j*s22+1
             .asg 2,k
             .loop
                lta     *+,AR3
                mpy     *+,AR2
                ;lta mat1+i*s21+k
                ;mpy mat2+j*s22+k
                .eval k+1,k
                .break(k == s21)
             .endloop
             apac
             lar     AR2,#reslt+s22*i+j
             add     #16384,1
             sach *
             .eval j+1,j
             .break(j == s22)
          .endloop
          .eval i+1,i
          .break (i == s11)
       .endloop
       .else
       .asg    0,i
```

```
        .loop
           .asg       0,j
           .loop
              zap
              lar AR2,#mat1+i*s21
              lar AR3,#mat2+j*s21
              .asg 0,k
              .loop
                 lta *+, AR3
                 mpy *+, AR2
                 ;lta mat1+i*s21+k
                 ;mpy mat2+j*s22+k
                 .eval k+1,k
                 .break(k == s21)
              .endloop
              apac
              lar AR2, #reslt+s22*i+j
              add     #16384,1
              sach *
              .eval j+1,j
              .break(j == s22)
           .endloop
           .eval i+1,i
           .break (i == s11)
        .endloop
        .endif
        clrc ovm        ; saturation mode off
        spm 0           ;shifting off
        .endm



; matrix determinant for 2x2 matrices.
; input is fractional matrix, output is fractional
detG    .macro  mat, reslt
        setc    ovm     ; saturation mode on
        spm     1       ; product sh mode 1
        lar     AR2, #mat
        lar     AR3, #mat+3
        mar     *,AR2
        lt      *+,AR3  ;a
        mpy     *-,AR2  ;d
        ltp     *,AR3   ;b
```

```
        mpy     *,AR2   ;c

        spac            ;ad-bc
        lar     AR2,#reslt
        add     #16384,1
        sach    *
        clrc    ovm     ; saturation mode off
        spm     0       ; product sh mode 0
        .endm


;matrix adjunct for 2x2 fractional matrices
; reslt must not be mat.
madjG   .macro  mat, reslt
        setc    ovm     ; saturation mode on
        spm     1       ; product sh mode 1

        lar     AR2, #mat
        lar     AR3, #reslt+3
        mar     *,AR2
        lacc    *+,AR3  ;a
        sacl    *,AR2
        lacc    *+,AR3  ;b
        sbrk    2
        neg
        sacl    *+,AR2
        lacc    *+,AR3  ;c
        neg
        sacl    *,AR2
        lacc    *, AR3  ;d
        sbrk    2
        sacl    *,AR2

        clrc    ovm     ; saturation mode off
        spm     0       ; product sh mode 0
        .endm

;matrix addition
; reslt = mat1 + mat2
; reslt can be mat1, or mat2 as well.
maddG   .macro  mat1, mat2, reslt, s0, s1
        setc    ovm
        lar     AR2, #mat1
```

```
        lar     AR3, #mat2
        lar     AR4, #reslt
        mar     *,AR2
        .asg 0,i
        .loop
           lacc   *+,16,AR3
           add    *+,16,AR4
           sach   *+,AR2
           .eval i+1, i
           .break(i == s0*s1)
        .endloop
        clrc    ovm
        .endm



;matrix subtraction
; reslt = mat1-mat2;
; reslt can be mat1, or mat2 as well.
msubG   .macro  mat1, mat2, reslt, s0, s1
        setc    ovm
        lar     AR2, #mat1
        lar     AR3, #mat2
        lar     AR4, #reslt
        mar     *,AR2
        .asg 0,i
        .loop
           lacc   *+,16,AR3
           sub    *+,16,AR4
           sach   *+,AR2
           .eval i+1, i
           .break(i == s0*s1)
        .endloop
        clrc    ovm
        .endm



;matrix addition with diagonal matrix
; mat1 = mat1 + mat2
; where mat2 is a diagonal matrix, and it is stored as a vector.
madiG   .macro  mat1, mat2, s0
```

```
        setc    ovm
        lar     AR2, #mat1
        lar     AR3, #mat2
        lar     AR0, #s0+1
        mar     *,AR2

        .asg 0,i
        .loop
           lacc   *,16,AR3
           add    *+,16,AR2
           sach   *0+
           .eval i+1, i
           .break(i == s0)
        .endloop
        clrc    ovm
        .endm

;matrix multiplied by a scalar (fractional)
; reslt = mat*scl;
; reslt can be mat as well
mmsclG  .macro  mat, scl, reslt, s0, s1
        setc    ovm
        spm     1
        lar     AR2, #scl
        mar     *,AR2
        lt      *
        lar     AR2, #mat
        lar     AR3, #reslt
        .asg 0,i
        .loop
           mpy    *+,AR3
           pac
           add    #16384,1
           sach   *+,AR2
           .eval i+1, i
           .break(i == s0*s1)
        .endloop
        clrc    ovm
        spm     0
        .endm
```

```
; matrix divided by a scalar. (Quasi Floating Point division)
; reslt = mat/scl;
; reslt and mat are [s0 x s1], scl is scalar
; t0 and t1 are temporary variables
mdsclG   .macro mat, scl, reslt, s0, s1, t1

      lar     AR4,#0ffffh

      lar     AR2,#scl
      mar     *,AR2
      lacc    *, 16, AR4
      bcnd    epi?, eq        ;scl must not be 0!
      bcnd    neg?, lt
x?
      sfl
      mar     *+              ;counting exponent in AR4
      bcnd    x?,geq

      ror                     ; rolling back to previous
      lar     AR2,#t1
      mar     *,AR2
      sach    *
      lacc    *               ;elimination of t1==16384
      sub     #16384
      bcnd    ok?, NEQ
      lacc    #16385
      sacl    *
ok?
      lacc    #16384,15
      rpt     #15
      subc    *
      and     #0ffffh

      b       conti?


neg?    neg
      mar     *,AR4
x1?
      sfl
      mar     *+
      bcnd    x1?,geq
```

```
      ror
      lar     AR2,#t1
      mar     *,AR2
      sach    *
      lacc    *             ;elimination of t1==16384
      sub     #16384
      bcnd    ok?, NEQ
      lacc    #16385
      sacl    *
ok1?
      lacc    #16384,15
      rpt     #15
      subc    *
      and     #0ffffh
      neg

conti?

      sacl    *
      setc    ovm
      spm     1
      lt      *
      sar     AR4,*, AR3
      lar     AR3,#mat
      .asg 0,i
      .loop
         mpy    *+,AR2
         pac
         lar    AR4, *, AR4
loop:i:?              ; left shifting with saturation.
         sacb
         addb
         banz       loop:i:?

         ldp    #reslt+i
         add     #16384,1
         sach   reslt+i
         mar    *,AR3
         .eval i+1, i
         .break(i == s0*s1)
      .endloop

epi?
```

```
        clrc    ovm
        spm     0
        .endm


; Vector normalization.
; reslt = vec/abs(vec);
; reslt and vec are [2 x 1] vectors
; t1 is a temporary store cell
vnormG  .macro vec, reslt, t1
        lar     AR2, #vec
        lar     AR3, #reslt
        lar     AR4,#0
        lar     AR5,#t1   ;AR5->temp
        mar     *,AR2
        spm     0
        zap
        sqra     *+
        sqra     *-,AR4
        apac              ;ACCH= (abs(vec))^2/2

        rpt     #15
        norm    *+

        nop             ;pipeline protection
        nop

        mar     *,AR3
        sach    *+          ;Store normalized value of (abs(vec))^2/2 in reslt
        splk    #27969, *-      ;Beginning value for the iteration


        .loop   3
        lt      *+              ;N/2
        mpy     *,AR5           ;R[i-1]
        pac
        sach    *,1
        lt      *,AR3           ;T=N/2*R[i-1]
        mpy     *,AR5             ;R[i-1]
        pac
        neg
        add     #32767,15
        add     #16385,15       ;ACCH=(1.5-(N/2)*R[i-1]^2)>>1
```

```
      sach   *
      lt     *,AR3
      mpy    *             ;R[i-1]
      pac
      sach   *-,2          ;Storing new R[i]
      .endloop


      mar    *+
      lt     *-,AR2       ;T=1/Root
      mpy    *+,AR3
      pac
      sach   *+,1,AR2
      mpy    *,AR3
      pac
      sach   *-,1,AR5

      ;denorming the values
      sar    AR4,*
      lacc   *
      bcnd   nodenorm1?,EQ   ;no denorming if shiftcount is 0
      ror
      sacl   *
      lar    AR4,*,AR4
      mar    *-             ;decrementing AR4 for looping
      bcnd   even?,NC
      mar    *+, AR3        ;if odd, it must be incremented
      lt     *
      mpy    #23170         ;sqrt(2)/2
      pac
      sach   *+,1
      lt     *
      mpy    #23170         ;sqrt(2)/2
      pac
      sach   *-,1

even?
      mar    *,AR3
loop?
      lacc   *,16
      add    *,16
      sach   *+
      lacc   *,16
```

```
        add     *,16
        sach    *-,AR4
        banz    loop?,AR3

nodenorm1?

        .endm
```

```
; Kalman Filter for the TMS320C5x
;  ctr.asm
;     Author : Balazs Simor
;     Date :     June, 1996

cls_lp  .set    1       ;1 if kalman filter is in the closed loop,
                        ;0 for open loop
;-----------------------------Globals-----------------------------------
                ;---Functions---
          .globl   _sine
          .globl  _c_int0
          .globl  _c_int4
          .globl   timer
          .globl   _get_incr   ;incremental encoder IF
          .globl  _send_to_pwm

          ;---Variables----
        .globl   _i_sq_max
        .globl   _i_sd_ref
        .globl   _omega_m_ref
        .globl   _omega_m_ref_delayed
        .globl  _u_a
        .globl  _u_b
        .globl  _u_c
        .globl  _i_sd
        .globl  _d_decouple
        .globl  _phase_current
        .globl  _i_sd_m
        .globl  _omega_fr
        .globl  _omega_m
        .globl  _omega_m_est
        .globl  _omega_m_m
        .globl  _eps_m
        .globl  _eps_m_est
        .globl  _i_sq_m
        .globl  _psi_rd
        .globl  _x_psi
        .globl  _eps_fs
        .globl  _omega_fs
        .globl  _omega_rs
        .globl  _i_sq_scal
```

```
        .globl  _i_sd
        .globl  _d_decouple
        .globl  _i_sq_ref
        .globl  _i_sq
        .globl  _q_decouple
        .globl  _psi_decouple
        .globl  _u_a
        .globl  _u_b
        .globl  _u_c
        .globl  _u_a_mod
        .globl  _u_b_mod
        .globl  _u_c_mod
        .globl  _u_sd
        .globl  _u_sq

        .globl  _x
        .globl  _x_1
        .globl  _K

        .globl  _pwm_period_reg
        .globl  _pwmon

;-----------------------------Base Addresses---------------------------
MODIFIED   .set    0    ; modified address lines on the board
************************************
*Register addresses of the PWM UNIT.
        .if MODIFIED=1
PWMTCR  .set    1000H
PWMTR   .set    1001h
PWMPR   .set    1008h
PWMCR0  .set    1009h
PWMCR1  .set    1002h
PWMCR2  .set    1003h
PWMAR   .set    100ah
PWMIR   .set    100bh
PWMVR   .set    1004h
PWMDTR  .set    1005h
PWMIOR  .set    100ch
        .else
PWMTCR  .set    1000H
PWMTR   .set    1001h
PWMPR   .set    1002h
PWMCR0  .set    1003h
```

```
PWMCR1  .set    1004h
PWMCR2  .set    1005h
PWMAR   .set    1006h
PWMIR   .set    1007h
PWMVR   .set    1008h
PWMDTR  .set    1009h
PWMIOR  .set    100ah
        .endif
************************************
*Register addresses of the GPIO UNIT.
        .if MODIFIED=1
GPIOTCR  .set    0000H
GPIOTR   .set    0001h
GPIOPR   .set    0008h
GPIOCCR  .set    0009h
GPIOCR0  .set    0002h
GPIOCR1  .set    0003h
GPIOCR2  .set    000ah
GPIOCR3  .set    000bh
GPIOCOR  .set    0004h
GPIOIO   .set    0005h
GPIOCRIO .set    000ch
GPIOCRIO1 .set    000dh
GPIOIT    .set    0006h
GPIOVEC    .set    0007h
GPIOCMP    .set    000eh
        .else
GPIOTCR  .set    0000H
GPIOTR   .set    0001h
GPIOPR   .set    0002h
GPIOCCR  .set    0003h
GPIOCR0  .set    0004h
GPIOCR1  .set    0005h
GPIOCR2  .set    0006h
GPIOCR3  .set    0007h
GPIOCOR  .set    0008h
GPIOIO   .set    0009h
GPIOCRIO .set    000ah
GPIOCRIO1 .set    000bh
GPIOIT    .set    000ch
GPIOVEC    .set    000dh
GPIOCMP    .set    000eh
GPIOINC .set    000fh
```

```
      .endif
************************************
*Addresses of ADC and MUX
ADBASE  .set    2000h
MUXBASE .set    2001h



;-----------------------------Reset Vectors-----------------------------
        .sect  "vectors"        ;power up reset vector
        b       _c_int0         ;go to start of program
        .space    16*6
        b       _c_int4         ;timer interrupt



        .data
        .align

variables:
counter .word 0     ;variable for switching between
;-----------------------------Cur. Control-----------------------------
cur_control:        ;/* here are all variables for the current control*/
temp    .word   0
temp1   .word   0
temp2   .word   0
_pwm_period_reg .word 0

;/*From ACCTRL.H */
;-- system matrices of current controllers
;-- i_a not necessary --> high precision mode
;i_b : scalable constant vector (4) of fractional
;  := ( 1.35898572914201E-01,
;      -1.35898572914201E-01,
;       0.00000000000000E+00,
;       0.00000000000000E+00);
;_i_b    .int    4453, -4453, 0, 0   ;Not scaled
;MODIFIED FOR NEW TIMING
_i_b    .int    4453*2,-4453*2, 0, 0   ;Not scaled

;i_c : scalable constant vector (1) of fractional
;  := ( 1.00000000000000E+00);
_i_c    .int    4096                ;Scaled by 2^-3
```

```
;i_d : scalable constant vector (4) of fractional
;  := ( 2.36686390532544E+00,
;      -2.36686390532544E+00,
;       7.14981873603059E-01,
;       3.43195760762822E+00);
_i_d    .int    9695, -9695, 2929, 14057 ;Scaled by 2^-3


;-- state variables
;d_xk      : vector (1) of fractional;   -- d-current
_d_xk   .int    0
;d_xk1     : vector (1) of fractional;
_d_xk1  .int    0
;d_xk1_hp  : rawaccumulator;
_d_xk1_hp .long 0

;q_xk      : vector (1) of fractional;   -- q-current
_q_xk   .int    0
;q_xk1     : vector (1) of fractional;
_q_xk1  .int    0
;q_xk1_hp  : rawaccumulator;
_q_xk1_hp .long 0


;/* From ACTRANS.H */
;-- matrix to transform phase currents to current vector in stator frame
;phase_to_stator1 : scalable constant vector (2) of fractional
;  := (1.0, 0.0);
; This is not used
;_phase_to_stator1   .int    32767, 0    ;not scaled

;phase_to_stator2 : scalable constant vector (2) of fractional
;  := ( 0.577350269, 1.154700538);
_phase_to_stator2   .int    9459, 18919 ;scaled by 2^-1

;-- matrix to transform from stator frame to rotor flux frame
;-- the coefficients are runtime dependent and are
;-- predefined worst case for the scalar product scaling
;stator_rotor1 : scalable constant vector (2) of fractional
;  := ( 0.707106781, 0.707106781);
_stator_rotor1  .int    23170, 23170        ;not scaled
```

```
;stator_rotor2 : scalable constant vector (2) of fractional
;  := (-0.707106781, 0.707106781);
_stator_rotor2  .int    -23170, 23170       ;not scaled

;-- matrix to transform voltage vector in stator frame to phase voltages
;stator_to_phase1  : scalable constant vector (2) of fractional
; := ( 1, 0);
; This is not used
;_stator_to_phase1   .int    32767, 0    ;not scaled

;stator_to_phase2  : scalable constant vector (2) of fractional
;  := ( -0.5, 0.866025403);
_stator_to_phase2   .int    -16384, 28378    ;not scaled

;stator_to_phase3  : scalable constant vector (2) of fractional
;  := ( -0.5, -0.866025403);
_stator_to_phase3   .int    -16384, -28378   ;not scaled

;-- intermediate variables
;result        : vector (2) of fractional; -- intermediate result vector
_result .int    0, 0
;rotor_voltage : vector (2) of fractional; -- voltage in rotor flux frame
_rotor_voltage  .int    0,0

;/*From ACIMOD.H */
;-- system matrices of flux model
;m_a1 : scalable constant vector (2) of fractional
;  := ( 9.94247360164699E-01,
;       0.00000000000000E+00);
_m_a1   .int    32579, 0        ;not scaled
;m_a2 : scalable constant vector (2) of fractional
;  := ( 0.00000000000000E+00,
;       1.00000000000000E+00);
; This is not used
;_m_a2   .int    0, 32767        ;not scaled
;m_b_1 : scalable constant vector (5) of fractional
;  := ( 6.90506233209837E-03,
;       0.00000000000000E+00,
;       0.00000000000000E+00,
;       0.00000000000000E+00,
;       0.00000000000000E+00);
_m_b_1  .int    226, 0, 0, 0, 0     ;not scaled
```

```
;m_b_2 : scalable constant vector (5) of fractional
;  := ( 0.00000000000000E+00,
;       4.99999999920422E-02,
;       0.00000000000000E+00,
;       0.00000000000000E+00,
;       0.00000000000000E+00);
_m_b_2  .int    0, 1638, 0, 0, 0    ;not scaled
;m_c_1 : scalable constant vector (2) of fractional
;  := ( 9.99999999999999E-01,
;       0.00000000000000E+00);
;_m_c_1  .int    32767, 0        ;not scaled
;MODIFIED FOR NEW TIMING
_m_c_1  .int    32767, 0        ;Scaled by 2^-1
;m_c_2 : scalable constant vector (2) of fractional
;  := ( 3.61100000000000E-01,
;       0.00000000000000E+00);
;_m_c_2  .int    11833, 0        ;not scaled
;MODIFIED FOR NEW TIMING
_m_c_2  .int    23665, 0        ;not scaled
;m_c_3 : scalable constant vector (2) of fractional
;  := ( 0.00000000000000E+00,
;       9.99999999999999E-01);
;_m_c_3  .int    0, 16384    ;Scaled by 2^-2
;MODIFIED FOR NEW TIMING
_m_c_3  .int    0, 32767    ;Scaled by 2^-2
;m_d_3 : scalable constant vector (5) of fractional
;  := ( 0.00000000000000E+00,
;       0.00000000000000E+00,
;       0.00000000000000E+00,
;       2.00000000000000E+00,
;       0.00000000000000E+00);
; This is not used
;_m_d_3  .int    0, 0, 0, 32767, 0   ;scaled by 2^-1
;m_d_4 : scalable constant vector (5) of fractional
;  := ( 0.00000000000000E+00,
;       1.00000000000000E+00,
;       1.00000000015916E+00,
;       0.00000000000000E+00,
;       0.00000000000000E+00);
_m_d_4  .int    0, 16384, 16384, 0,0    ;scaled by 2^-1
;m_d_5 : scalable constant vector (5) of fractional
;  := ( 0.00000000000000E+00,
;       0.00000000000000E+00,
```

```
;       1.00000000015916E+00,
;       0.00000000000000E+00,
;       0.00000000000000E+00);
; This is not used
;_m_d_5  .int    0, 0, 32767, 0, 0   ;not scaled
;m_d_6 : scalable constant vector (5) of fractional
;  := ( 0.00000000000000E+00,
;       0.00000000000000E+00,
;       0.00000000000000E+00,
;       0.00000000000000E+00,
;       1.59154943096444E-02);
_m_d_6  .int    0, 0, 0, 0, 522     ;not scaled

;-- state variables
;m_xk   : vector (2) of fractional; -- flux model
_m_xk   .int    0,0
;m_xk1  : vector (2) of fractional;
_m_xk1  .int    0,0

;/* From VEL_CTRL.DSP */
;-- external inputs
;phase_current   : vector (2) of fractional;
;input is phase_current;
;  i_phase_a renames phase_current (1);
;  i_phase_b renames phase_current (2);
_phase_current
_i_phase_a  .int    0
_i_phase_b  .int    0



;-- current controller inputs
;d_controller_input  : vector (4) of fractional;
;  i_sd_ref          renames d_controller_input (1);
;  i_sd         renames d_controller_input (2);
;  d_decouple        renames d_controller_input (3);
_d_controller_input
_i_sd_ref   .int    0
_i_sd       .int    0
_d_decouple .int    0
            .int    0

;q_controller_input  : vector (4) of fractional;
```

```
;  i_sq_ref      renames q_controller_input (1);
;  i_sq             renames q_controller_input (2);
;  q_decouple       renames q_controller_input (3);
;  psi_decouple     renames q_controller_input (4);
_q_controller_input
_i_sq_ref   .int    0
_i_sq       .int    0
_q_decouple .int    0
_psi_decouple   .int    0

;-- controller outputs
;u_sd     : fractional;       -- output from d current controller
_u_sd   .int    0
;u_sq     : fractional;       -- output from q current controller
_u_sq   .int    0

;-- phase voltages
;u_a : fractional;
_u_a    .int    0
;u_b : fractional;
_u_b    .int    0
;u_c : fractional;
_u_c    .int    0

;-- phase voltages to compute duty cycles
;u_a_mod : fractional;
_u_a_mod    .int    0
;u_b_mod : fractional;
_u_b_mod    .int    0
;u_c_mod : fractional;
_u_c_mod    .int    0

;-----------------------------Vel. Control-----------------------------
          .align
;/*From ACCTRL.H */
vel_control:    ;/*here are the variables for the velocity controller*/
;-- system matrices of velocity controller
;-- v_a not necessary --> high precision mode
;v_b : scalable constant vector (2) of fractional
;  := ( 1.05998807393070E+00,
;       -1.05998807393070E+00);
;_v_b    .int    17367, -17367     ;Scaled by 2^-1
;MODIFIED FOR NEW TIMING
```

```
_v_b    .int    17367*8/25, -17367*8/25     ;Scaled by 2^-1


;v_c : scalable constant vector (1) of fractional
;  := ( 1.00000000000000E+00);
_v_c    .int    1024        ; Scaled by 2^-5

;v_d : scalable constant vector (2) of fractional
;  := ( 1.88495559240000E+01,
;      -1.88495559240000E+01);
_v_d    .int    19302*2/5, -19302*2/5       ;Scaled by 2^-5


;-- system matrices of first order lag for velocity reference
;a_v_fol : scalable constant vector (1) of fractional
;  := ( -4.5412768e-02); -- computed matrix - 1.0 !!!
_a_v_fol .int   -1488           ;Not scaled

;b_v_fol : scalable constant vector (1) of fractional
;  := ( 4.49627708289495E-02);
_b_v_fol .int   1473            ;Not scaled

;c_v_fol : scalable constant vector (1) of fractional
;  := ( 9.86896949527394E-01);
;_c_v_fol .int   32339           ;Not scaled
;MODIFIED FOR NEW TIMING
_c_v_fol .int   32339           ;Not scaled

;d_v_fol : scalable constant vector (1) of fractional
;  := ( 2.28822623921473E-02);
_d_v_fol .int   750     ;Not scaled

; Filter for velocity value
;fir_coeff : scalable constant vector (5) of fractional
;  := (0.2, 0.2, 0.2, 0.2, 0.2);
_fir_coeff      .int    6553, 6553, 6553, 6553, 6553

;encoder_counter   : vector (2) of fractional;
_encoder_counter        .int    0,0
;fir_omega_m       : vector (5) of fractional;
_fir_omega_m    .int    0, 0, 0, 0, 0
```

```
;v_xk      : vector (1) of fractional;   -- velocity
_v_xk   .int    0
;v_xk1     : vector (1) of fractional;
_v_xk1  .int    0
;v_xk1_hp  : rawaccumulator;
_v_xk1_hp .long 0
;qirfs     : fractional;
_qirfs  .int    0

;v_fol_xk  : vector (1) of fractional;   -- velocity reference first order lag
_v_fol_xk .int  0
;v_fol_xk1 : vector (1) of fractional;
_v_fol_xk1 .int 0
;v_fol_xk1_hp : rawaccumulator;
_v_fol_xk1_hp .long 0


;-- input vectors
;v_fol_u   : vector (1) of fractional;   -- for velocity first order lag
_v_fol_u .int   0


;/* From VEL_CTRL.DSP */
;omega_m_ref       : fractional;         -- reference of rotor velocity
;input is omega_m_ref;
_omega_m_ref    .int    0

;-- velocity controller inputs
;v_controller_input : vector (2) of fractional;
;  omega_m_ref_delayed renames v_controller_input (1);
;  omega_m             renames v_controller_input (2);
_v_controller_input
_omega_m_ref_delayed    .int    0
       .if cls_lp=1
_omega_m_est    .int    0
       .else
_omega_m        .int    0
       .endif

       .if cls_lp=0    ;defining the other variable outside
_omega_m_est    .int    0
       .else
```

```
_omega_m        .int    0
        .endif

;-- miscellaneous
;i_sq_max  :fractional;
_i_sq_max   .int    0


        .globl  _u_al, _u_be, _i_al, _i_be
        .globl  diff_FI, h, _x, _x_1, diff_h,P, P_1, _K, RSLT, Q, R
        .align
;-----------------------------Kalman Filter-----------------------------
; New data page for Kalman filter variables
kal_fil

eps_int .long   0
;input vector
U
_u_al   .word 0
_u_be   .word 0
;output vector
Y
_i_al   .word 0
_i_be   .word 0


diff_FI .word   25680, 0, 498, 0, 0
        .word   0, 25680, 0, 498, 0
        .word   3675, 0, 32358, 0, 0
        .word   0, 3675, 0, 32358, 0
        .word   0, 0, 0, 0, 32767

_x      .word   0, 0, 0, 0, 0

h       ;h=[x(1),x(2)]
_x_1    .word   0, 0, 0, 0, 0

diff_h  .word   32767, 0, 0, 0, 0
        .word   0, 32767, 0, 0, 0

P       .word   0, 0, 0, 0, 0
        .word   0, 0, 0, 0, 0
        .word   0, 0, 0, 0, 0
```

```
        .word   0, 0, 0, 0, 0
        .word   0, 0, 0, 0, 0

P_1     .word   0, 0, 0, 0, 0
        .word   0, 0, 0, 0, 0
        .word   0, 0, 0, 0, 0
        .word   0, 0, 0, 0, 0
        .word   0, 0, 0, 0, 0

Q       .word   0, 0, 0, 0, 80h

R       .word   7000h, 7000h

RSLT    .space 25*16

_K      .word   0, 0
        .word   0, 0
        .word   0, 0
        .word   0, 0
        .word   0, 0

        .text

;-----------------------------Macros-----------------------------------
        .copy "fdiv.inc"        ;fractional division macro


shconst1    .set    4000h
shconst2    .set    6000h
shconst3    .set    7000h
shconst4    .set    7800h
shconst5    .set    7C00h
shconst6    .set    7E00h
shconst7    .set    7F00h

;SATurate Accumulator macro
;performs saturation (1<=Num<=7) before shifting by Num
sata    .macro  Num
        setc    ovm             ;saturating
        exar
        lacc    #shconst:Num:, 15
        sfl
        exar            ; ACCB = constant for saturating
```

```
        addb
        sbb
        sbb
        addb
        clrc    ovm
        .endm


;Round And Saturate Accumulator macro.
;performs rounding and saturation (1<=Num<=7) before shifting by Num
rasa    .macro  Num
        add     #1, 15-Num      ;rounding
        setc    ovm             ;saturating
        exar
        lacc    #shconst:Num:, 15
        sfl
        exar            ; ACCB = constant for saturating
        addb
        sbb
        sbb
        addb
        clrc    ovm
        .endm


        .copy "matG.inc" ;macros for matrix manipulation

;-------------------------Routines for the Kalman filter ----------------
;*********
;Inputs: i_al, i_be, u_al, u_be
;Outputs: omega_m_est, sin(eps_fs), cos(eps_fs)
        .globl  _kalman_filter
_kalman_filter:
        ldp     #kal_fil
        sar     AR0,*+
        sar     AR2,*+
        sar     AR3,*+
        sar     AR4,*+
;       % calc. inp and out vec of kalman filter
;       U = [u(1); u(2)];
;       Y = [u(3); u(4)];
```

```
;       %at this point we have x=x[k-1], U=u[k-1], Y=y[k]????????
;
;       %prediction
;       diff_FI=[1-Kr_K/Kl_K*T, 0, Lh_K*Rr_K/Lr_K/Lr_K/Kl_K*T,
Lh_K/Lr_K*omega_m_scl*x(5)/Kl_K*T, Lh_K/Lr_K*psi_scl*x(4)/Kl_K*T;
;               0, 1-Kr_K/Kl_K*T, -Lh_K/Lr_K*omega_m_scl*x(5)/Kl_K*T,
Lh_K*Rr_K/Lr_K/Lr_K/Kl_K*T,-Lh_K/Lr_K*psi_scl*x(3)/Kl_K*T;
;               Lh_K/Tr_K*T, 0, 1-T/Tr_K, -omega_m_scl*x(5)*T, -psi_scl*x(4)*T;
;               0, Lh_K/Tr_K*T, omega_m_scl*x(5)*T, 1-T/Tr_K, psi_scl*x(3)*T;
;               0, 0, 0, 0, 1]./Q_scl;

        lar     AR2,#_x+4
        lar     AR3,#diff_FI+3
        mar     *,AR2
        lt      *-,AR3          ;Treg=x(5)
        mpy     #6256
        pac
        sach    *,1
        neg
        adrk    4
        sach    *,1
        mpy     #-5147
        pac
        adrk    6
        sach    *,1
        neg
        adrk    4
        sach    *,1,AR2
        lt      *-,AR3          ;Treg=x(4)
        mpy     #-5147
        pac
        sbrk    3
        sach    *,1
        mpy     #6256
        pac
        sbrk    10
        sach    *,1,AR2
        lt      *,AR3           ;Treg=x(3)
        mpy     #-6256
        pac
        adrk    5
        sach    *,1
        mpy     #5147
```

```
      pac
      adrk    10
      sach    *,1
;
;     x_1=[diff_FI(1,1)*x(1)+diff_FI(1,3)*x(3)+diff_FI(1,4)*x(4);
;         diff_FI(2,2)*x(2)+diff_FI(2,3)*x(3)+diff_FI(2,4)*x(4);
;         diff_FI(3,1)*x(1)+diff_FI(3,3)*x(3)+diff_FI(3,4)*x(4);
;         diff_FI(4,2)*x(2)+diff_FI(4,3)*x(3)+diff_FI(4,4)*x(4);
;         diff_FI(5,5)*x(5)]...
;        +T*[u_s_scl*u(1)/Kl_K; u_s_scl*u(2)/Kl_K; 0; 0; 0]./x_scl;

      spm     1
      setc    ovm
      lar     AR4,#_x_1
      lar     AR2,#_x
      lar     AR3,#diff_FI
      mar     *,AR2
      lt      *+
      mar     *+,AR3
      mpy     *+
      mar     *+,AR2
      ltp     *+,AR3
      mpy     *+,AR2
      lta     *+,AR3
      mpy     *+,AR2
      lar     AR2,#U
      lta     *,AR4
      mpy     #10749
      apac
      sach    *+,AR2          ;storing in x_1

      lar     AR2,#_x+1
      lar     AR3,#diff_FI+6
      lt      *+,AR3
      mpy     *+,AR2
      ltp     *+,AR3
      mpy     *+,AR2
      lta     *+,AR3
      mpy     *+,AR2
      lar     AR2,#U+1
      lta     *,AR4
      mpy     #10749
      apac
```

```
      sach    *+,AR2          ;storing in x_1

      lar     AR2,#_x
      lar     AR3,#diff_FI+10
      mar     *,AR2
      lt      *+
      mar     *+,AR3
      mpy     *+
      mar     *+,AR2
      ltp     *+,AR3
      mpy     *+,AR2
      lta     *+,AR3
      mpy     *+,AR4
      apac
      sach    *+,AR2          ;storing in x_1

      lar     AR2,#_x+1
      lar     AR3,#diff_FI+16
      lt      *+,AR3
      mpy     *+,AR2
      ltp     *+,AR3
      mpy     *+,AR2
      lta     *+,AR3
      mpy     *+,AR4
      apac
      sach    *+,AR2          ;storing in x_1

      lar     AR2,#_x+4
      lar     AR3,#diff_FI+24
      lt      *,AR3
      mpy     *,AR4
      pac
      sach    *,AR2
      spm     0
      clrc    ovm

;
;     P_1=diff_FI*P*diff_FI' + GAM*Q*GAM'; %P[k|k-1] is ready
      mmtfraG P, diff_FI, RSLT,   5,5,5
      mmfraG  diff_FI, RSLT, P_1, 5,5,5
      madiG   P_1,Q,5

;
```

```
;       % calculation of h, diff_h
;       h=[x(1); x(2)];
;       diff_h=[1 0 0 0 0
;               0 1 0 0 0];
        ;Nothing to do

;       % system of Kalman filter
;       K=P_1*diff_h'* inv(diff_h*P_1*diff_h' +R); %K[k]
        mmtfraG P_1, diff_h, RSLT, 5,5,2
        mmfraG  diff_h, RSLT, _K, 2,5,2 ;_K is yet unused, use as temp
        madiG   _K, R, 2
        madjG   _K, P                   ;P is now unused, use it as temp.
        mmfraG  RSLT, P, _K, 5, 2, 2
        detG    P, RSLT     ; supposing, that det(RSLT)<1
        mdsclG  _K, RSLT, _K, 5, 2, RSLT+1

;       sys = x_1+K*(Y-h);      %x[k] is ready
        msubG   Y, h, P, 2,1            ;use P as temp
        mmfraG  _K, P, RSLT, 5,2,1
        maddG   _x_1, RSLT, _x, 5,1
;       P=P_1-K*diff_h*P_1;     %P[k] is ready
        mmfraG _K, diff_h, RSLT, 5, 2, 5
        mmfraG  RSLT, P_1, P , 5, 5, 5
        msubG   P_1,P,P,5,5

;We need an additional integrator to calculate eps_RS_est
        ldp     #kal_fil
        lacc    _x+4
        ldp     #vel_control
        sacl    _omega_m_est

;Calculation of sin(eps_fs) and cos(eps_fs) from psi_r_al, und psi_r_be
        vnormG  _x+2,_stator_rotor1, RSLT
        ldp     #cur_control
        lacc    _stator_rotor1
        or      _stator_rotor1+1
        bcnd    ok_k, NEQ
        lacc    #32767
        sacl    _stator_rotor1
ok_k
        lacc    _stator_rotor1
        sacl    _stator_rotor2+1
        lacc    _stator_rotor1+1
```

```
        neg
        sacl    _stator_rotor2

k_epi
        mar     *,AR1
        mar     *-
        lar     AR4,*-
        lar     AR3,*-
        lar     AR2,*-
        lar     AR0,*
        ret


;-------------------------Routines for the velocity control------------

;*********
;Inputs: v_controller_input
;    i_sq_max
;Output: i_sq_ref
        .globl  _v_control
_v_control:
        ldp     #vel_control
;    update (v_xk1, v_xk);
        lacc    _v_xk1
        sacl    _v_xk

;    accumulate scalpro (controller_out_scp)
;      qirfs := v_c * v_xk1 + v_d * _v_controller_input;
;    end accumulate;
        lt      _v_c
        mpy     _v_xk1
        ltp     _v_d
        mpy     _v_controller_input
        lta     _v_d+1
        mpy     _v_controller_input+1
        apac
        rasa    6
        sach    _qirfs, 6

;    if  ABS (qirfs) <= i_sq_max then
;       accumulate prescalpro (controller_state_scp)
;    v_xk1_hp := v_xk1_hp + v_b * v_controller_input;
;       end accumulate;
```

```
;    end if;
       lacc    _qirfs
       sub     _i_sq_max
       bcnd    overrun, GT
       lacc    _qirfs
       add     _i_sq_max
       bcnd    underrun, LT

       lacc    _v_xk1_hp, 16
       clrc sxm
       add _v_xk1_hp+1
       setc sxm
       lt      _v_b
       mpy     _v_controller_input
       lta     _v_b+1
       mpy     _v_controller_input+1
       apac
       sata    2
       sach    _v_xk1_hp       ;not correcting after Q15 multipl.
       sacl    _v_xk1_hp+1 ; here.
;  accumulate prescalpro (controller_state_scp)
;    v_xk1 (1) := v_xk1_hp;
;  end accumulate ;
       sach    _v_xk1, 2

       b       ok_v


overrun
       lacc    _i_sq_max
       sacl    _qirfs
       b       comp

underrun
       lacc    _i_sq_max
       neg
       sacl    _qirfs

comp
       lacc    _qirfs, 10
       lt      _v_d
       mpy     _v_controller_input
       lts     _v_d+1
```

```
        mpy     _v_controller_input+1
        spac
        sach    _v_xk1, 6
        sach    _v_xk1_hp,4
        sacl    _v_xk1_hp+1,4

ok_v
        ldp     #_pwmon
        lacc    _pwmon
        ldp     #vel_control
        bcnd    on0,GT
        lacc    #0
        sacl    _v_xk1_hp
        sacl    _v_xk1_hp+1
        sach    _v_xk1
        sacl    _qirfs

on0

        lacc  _qirfs
        ldp     #cur_control
        sacl    _i_sq_ref

        ret


;*********
;Input:     _omega_m_ref
;Output:        _omega_ref_delayed
        .globl  _v_first_order_lag
_v_first_order_lag:
        ldp     #vel_control
;  v_fol_u (1) := v_ref_in;
        lacc    _omega_m_ref
        sacl    _v_fol_u

;  update (v_fol_xk1, v_fol_xk);
        lacc    _v_fol_xk1
        sacl    _v_fol_xk

;  accumulate scalpro (v_fol_out_scp)
;   omega_m_ref_delayed := c_v_fol * v_fol_xk1 + d_v_fol * v_fol_u;
;  end accumulate;
```

```
        lt  _c_v_fol
        mpy _v_fol_xk1
        ltp _d_v_fol
        mpy _v_fol_u
        apac
        rasa    1
        sach    _omega_m_ref_delayed,1


;   -- delay replacement to avoid quantization effects

;  accumulate prescalpro (v_fol_state_scp)
;    v_fol_xk1_hp := v_fol_xk1_hp + a_v_fol * v_fol_xk + b_v_fol * v_fol_u;
;  end accumulate;
        lacc    _v_fol_xk1_hp, 16
        clrc sxm
        add _v_fol_xk1_hp+1
        setc sxm
        lt  _a_v_fol
        mpy     _v_fol_xk
        lta _b_v_fol
        mpy _v_fol_u
        apac
        sata    1
        sach    _v_fol_xk1_hp   ;not correcting after Q15 multipl.
        sacl    _v_fol_xk1_hp+1 ;here.


;  accumulate prescalpro (v_fol_state_scp)
;    v_fol_xk1 (1) := v_fol_xk1_hp;
;  end accumulate
        sach    _v_fol_xk1, 1
        ldp     #_pwmon
        lacc    _pwmon
        ldp     #vel_control
        bcnd    on1,GT
        lacl    #0
        sacl    _v_fol_xk1_hp
        sacl    _v_fol_xk1_hp+1
        sacl    _v_fol_xk1
        sacl    _omega_m_ref_delayed

on1
```

```
        ret


;------------------------Routines for the current control-------------
;**********
        .globl  _q_control
_q_control:
        ldp     #cur_control
;  update (q_xk1, q_xk);
        lacc    _q_xk1
        sacl    _q_xk

;  accumulate scalpro (controller_out_scp)
;    u_sq := i_c * q_xk1 + i_d * q_controller_input;
;  end accumulate;
        lt  _i_c
        mpy _q_xk1
        ltp _i_d
        mpy _q_controller_input
        lta _i_d+1
        mpy _q_controller_input+1
        lta _i_d+2
        mpy _q_controller_input+2
        lta _i_d+3
        mpy _q_controller_input+3
        apac
        rasa    4   ;Saturating before shifting
        sach    _u_sq, 4

;  accumulate prescalpro (controller_state_scp)
;    q_xk1_hp := q_xk1_hp + i_b * q_controller_input;
;  end accumulate;
        lacc    _q_xk1_hp, 16
        clrc sxm
        add     _q_xk1_hp+1
        setc sxm
        lt      _i_b
        mpy     _q_controller_input
        lta     _i_b+1
        mpy     _q_controller_input+1
        ; i_b[2] and i_b[3] == 0, so we do not multiply them here.
        apac
        sata    1
```

```
        sach    _q_xk1_hp   ; not correcting after Q15 multipl.
        sacl    _q_xk1_hp+1 ; here.

;  accumulate prescalpro (controller_state_scp)
;    q_xk1 (1) := q_xk1_hp;
;  end accumulate
        sach    _q_xk1,1
        ldp     #_pwmon
        lacc    _pwmon
        ldp     #cur_control
        bcnd    on2,GT
        lacl    #0
        sacl    _q_xk1_hp
        sacl    _q_xk1_hp+1
        sacl    _q_xk1
        sacl    _u_sq

on2

        ret


;*********
;Input: _d_controller_input
;Output: _u_sd
        .globl  _d_control
_d_control:
        ldp     #cur_control
;  update (d_xk1, d_xk);
        lacc    _d_xk1
        sacl    _d_xk

;  accumulate scalpro (controller_out_scp)
;    u_sd := i_c * d_xk1 + i_d * d_controller_input1;
;  end accumulate;
        lt  _i_c
        mpy     _d_xk1
        ltp _i_d
        mpy _d_controller_input
        lta _i_d+1
        mpy _d_controller_input+1
        lta _i_d+2
        mpy _d_controller_input+2
```

```
        lta _i_d+3
        mpy _d_controller_input+3
        apac
        rasa    4
        sach    _u_sd,4


;       accumulate prescalpro (controller_state_scp)
;    d_xk1_hp := d_xk1_hp + i_b * d_controller_input;
;       end accumulate;
        lacc    _d_xk1_hp, 16
        clrc sxm
        add _d_xk1_hp+1
        setc sxm
        lt  _i_b
        mpy     _d_controller_input
        lta _i_b+1
        mpy     _d_controller_input+1
        ; i_b[2] and i_b[3] == 0, so we do not multiply them here.
        apac
        sata    1
        sach    _d_xk1_hp   ;not correcting after Q15 multipl.
        sacl    _d_xk1_hp+1     ;here.


;  accumulate prescalpro (controller_state_scp)
;    d_xk1 (1) := d_xk1_hp;
;  end accumulate
        sach    _d_xk1, 1
        ldp     #_pwmon
        lacc    _pwmon
        ldp     #cur_control
        bcnd    on3,GT
        lacl    #0
        sacl    _d_xk1_hp
        sacl    _d_xk1_hp+1
        sacl    _d_xk1
        sacl    _u_sd

on3

        ret
```

```
;*********
;Inputs: _eps_fs
;    _phase_current
;Outputs:_i_sd
;    _i_sq
        .globl  _phase_to_rotor
_phase_to_rotor:
        ldp     #cur_control

;   -- set input vector
;  -- transform phase to stator frame
;  accumulate scalpro (transformation_scp)
;    result (1) := phase_to_stator1 * phase_current;
;  end accumulate;
        lacc _phase_current
        ; we do not multiply by phase_to_stator[0], because it is 1
        ; we do not use phase_to_stator[1], because it is 0
        sacl _result
        ldp     #kal_fil
        sacl    _i_al
        ldp     #cur_control


;  accumulate scalpro (transformation_scp)
;    result (2) := phase_to_stator2 * phase_current;
;  end accumulate;
        lt  _phase_to_stator2
        mpy _phase_current
        ltp _phase_to_stator2+1
        mpy _phase_current+1
        apac
        rasa    2
        sach    _result+1, 2
        ldp     #kal_fil
        sach    _i_be,2
        ldp     #cur_control

;   -- transform stator frame to rotor flux frame
;  accumulate scalpro (transformation_scp)
;    i_sd := stator_rotor1 * result;
```

```
;  end accumulate;
        lt  _stator_rotor1
        mpy _result
        ltp _stator_rotor1+1
        mpy _result+1
        apac
        rasa    1
        sach    _i_sd, 1

;  accumulate scalpro (transformation_scp)
;    i_sq := stator_rotor2 * result;
;  end accumulate
        lt  _stator_rotor2
        mpy _result
        ltp _stator_rotor2+1
        mpy _result+1
        apac
        rasa    1
        sach    _i_sq, 1

        ret



;*********
;Inputs:    u_sd
;       u_sq
;Outputs:   u_a
;       u_b
;       u_c
        .globl  _rotor_to_phase
_rotor_to_phase:
        ldp     #cur_control
;   -- set input vector
;  rotor_voltage (1) := u_sd;
;  rotor_voltage (2) := u_sq;
        lacc    _u_sd
        sacl    _rotor_voltage
        lacc    _u_sq
        sacl    _rotor_voltage+1

;   -- get inverse of matrix stator_rotor
;   -- by transponation (swapping elements (2,1) and (1,2))
```

```
;  dummy := stator_rotor2 (1);
;  stator_rotor2 (1) := stator_rotor1 (2);
;  stator_rotor1 (2) := dummy;
        lacc    _stator_rotor2
        sacb
        lacc    _stator_rotor1+1
        sacl    _stator_rotor2
        lacb
        sacl    _stator_rotor1+1

;   -- transform rotor flux frame to stator frame
;  accumulate scalpro (transformation_scp)
;    result (1) := stator_rotor1 * rotor_voltage;
;  end accumulate;
        lt  _stator_rotor1
        mpy _rotor_voltage
        ltp _stator_rotor1+1
        mpy _rotor_voltage+1
        apac
        rasa    1
        sach    _result, 1
        ldp     #kal_fil
        sach    _u_al, 1
        ldp     #cur_control

;  accumulate scalpro (transformation_scp)
;    result (2)  := stator_rotor2 * rotor_voltage;
;  end accumulate;
        lt  _stator_rotor2
        mpy _rotor_voltage
        ltp _stator_rotor2+1
        mpy _rotor_voltage+1
        apac
        rasa    1
        sach    _result+1, 1
        ldp     #kal_fil
        sach    _u_be, 1
        ldp     #cur_control

;   -- transform stator frame to phase
;  accumulate scalpro (transformation_scp)
;    u_a := stator_to_phase1 * result;
;  end accumulate;
```

```
        lacc _result
        ; _stator_to_phase1 = { 1, 0 } !!
        sacl    _u_a

;  accumulate scalpro (transformation_scp)
;    u_b := stator_to_phase2 * result;
;  end accumulate;
        lt  _stator_to_phase2
        mpy _result
        ltp _stator_to_phase2+1
        mpy _result+1
        apac
        rasa    1
        sach    _u_b, 1

;  accumulate scalpro (transformation_scp)
;    u_c := stator_to_phase3 * result;
;  end accumulate
        lt  _stator_to_phase3
        mpy _result
        ltp _stator_to_phase3+1
        mpy _result+1
        apac
        rasa    1
        sach    _u_c, 1

        ret



;------------------------------Peripherial routines----------------------
        .globl  _to_pwm
_to_pwm:
        ldp     #cur_control
        lt      _pwm_period_reg

        ;we have to push parameters in REVERSE order!!

        lacc    #0              ; dummy parameter (bvec)
        sacl    *+

        mpy     _u_c_mod
        pac
```

```
        add     _pwm_period_reg, 15 ; += period/2
        sach    *+              ;stack = (period * (1 + u_a_mod)) /2

        mpy     _u_b_mod
        pac
        add     _pwm_period_reg, 15 ; += period/2
        sach    *+              ;stack = (period * (1 + u_a_mod)) /2

        mpy     _u_a_mod
        pac
        add     _pwm_period_reg, 15 ; += period/2
        sach    *+              ;stack = (period * (1 + u_a_mod)) /2

        call    _send_to_pwm  ;library function for pwm handling
        sbrk    #4              ;restoring stack
        RET


        .globl  _gpio_incr
_gpio_incr
        in      *,GPIOINC
        lacc    *
        ret


        .globl  _input_omega_m
_input_omega_m:
;   -- read encoder counter
;  input (encoder_counter (1));
        call    _gpio_incr

        ldp     #vel_control
                sacl    _encoder_counter

;   -- compute current velocity
;  accumulate scalpro (counter_scp) and update encoder_counter
;    fir_omega_m (1) := encoder_to_velocity * encoder_counter;
;  end accumulate;
        lacc    _encoder_counter
        sub     _encoder_counter+1
        dmov    _encoder_counter
        sacl    _encoder_counter
;   the velocity is multiplied by 32
```

```
;       sacl    _fir_omega_m, 5
;MODIFIED FOR NEW TIMING
;   the velocity is multiplied by 32*1.25
        lt      _encoder_counter
        mpy     #20480  ;1.25/2
        pac
        sach    _fir_omega_m, 7

;   -- filter current velocity
;  accumulate scalpro (fir_scp) and update fir_omega_m
;    velocity := fir_coeff * fir_omega_m;
;  end accumulate
        zap
        lt      _fir_omega_m+4
        mpy     _fir_coeff+4
        ltd     _fir_omega_m+3
        mpy     _fir_coeff+3
        ltd     _fir_omega_m+2
        mpy     _fir_coeff+2
        ltd     _fir_omega_m+1
        mpy     _fir_coeff+1
        ltd     _fir_omega_m
        mpy     _fir_coeff
        apac
        rasa    1
        sach    _omega_m, 1
        ret


;------------------------------Setup routines--------------------------


;!!! THIS MIGHT BE NEEDED LATER
;       LACC    #0101H
;       SACL    temp
;       OUT     temp, GPIOCRIO1    ; Sending a 1 to the DIG0 of the GPIO
                        ; This pin is connected to the PROT of the motor
                        ; driver card, and it enables all PWM channels
                        ; in the motor



_timer_init:
```

```
        .globl  _timer_init
        lacc    #10000      ;timer interrupt in every 500 us
        samm    PRD
        lacc    #100000b        ;divide down = 1, load tim from prd
        samm    TCR
        ret


_intr_init:
        .globl  _intr_init
        ldp     #cur_control
        clrc    cnf
        lacc    #830h
        samm    PMST
        lacc    #5
        sacl    counter
        call    _timer_init

        lacc    #8              ; enable only timer interrupt
        samm    IMR
        eint                    ; enable interrupts
        ret
          .end
```

```
; Kalman Filter for the TMS320C5x
; vel_ctr.c
;     Author : Balazs Simor
;     Date :     June, 1996
#include <stdlib.h>

#include <uart.h>
#include <conio.h>
#include <board.h>
#include <ad.h>
#include <incr.h>
#include <pwm.h>
#include <gpio.h>

int sine(int angle);

#define I_SD_REF_K 6554 /*=frac(0.20) scaled reference for d_current component => 2.0 A*/
#define I_SQ_MAX_K 13107 /*=frac(0.40) scaled maximum for q_current component => 4.0 A*/

#define PWM_PERIOD 600 /* period of PWM generating*/


long int UART_freq=4915200; /* 4.9152 MHz */
char *header="Field oriented velocity control with Kalman filter\r";
int spd=0; /* speed reference */
int pwmon=0; /* turns pwm on and off*/
long pwmsc = 0;

/*---------------------External variables-------------------------*/

#define extasm(varname) \
extern int varname;\
int *P##varname = &varname;

extasm(omega_m_ref)
extasm(omega_m_ref_delayed)
extasm(i_sq_max)
extasm(i_sq_ref)
extasm(i_sd_ref)
extasm(phase_current)
extasm(i_sd)
extasm(i_sq)
```

```
extasm(u_sd)
extasm(u_sq)
extasm(omega_m)
extasm(omega_m_est)
extasm(d_decouple)
extasm(q_decouple)
extasm(psi_decouple)
extasm(u_a)
extasm(u_b)
extasm(u_c)
extasm(u_al)
extasm(u_be)
extasm(i_al)
extasm(i_be)
extasm(u_a_mod)
extasm(u_b_mod)
extasm(u_c_mod)
extasm(pwm_period_reg)
extasm(x)
extasm(x_1)
extasm(K)


/*----------------------Interrupt handling-------------------------*/

int xxx = 0;

void current_control(void)
{
/*This is the main current control routine. Called from the interrupt rtn.*/
/*;    -- get phase currents*/
       Pphase_current[0] = (get_ad_val(0)+34-512)<<6;
       Pphase_current[1] = (get_ad_val(1)+36-512)<<6;


/*    transform phase currents to rotorflux frame*/
   phase_to_rotor (/*eps_fs, phase_current, i_sd, i_sq*/);

/*    calculate decoupling value d_component*/
/*    *Pd_decouple = (- (long)(*Pi_sq) * (long)(*Pomega_fs))>>15;*/

/*    PI-controller for d-current*/
   d_control (/*d_controller_input, u_sd*/);
```

```
/*    calculate decoupling value q_component*/
/*    *Pq_decouple =  ((long)(*Pi_sd) * (long)(*Pomega_fs))>>15;
    *Ppsi_decouple = ((long)(*Pomega_rs) * (long)(*Ppsi_rd))>>15;*/

/*    PI-controller for q-current*/
    q_control (/*q_controller_input, u_sq*/);

/*    transform voltage vector from rotorflux frame to phase voltages*/

/*    *Pu_sd =  2000;
    *Pu_sq = 0;*/
    rotor_to_phase (/*u_sd, u_sq, u_a, u_b, u_c*/);

/*    xxx += 10;
    *Pu_a = sine(xxx)/5;
    *Pu_b = sine(xxx-21845)/5;
    *Pu_c = sine(xxx-43691)/5;*/
/*    *Pu_a = 30000;
    *Pu_b = -15000;
    *Pu_c = -15000;*/


 /* phases a and b are swapped !!! */
    *Pu_a_mod = *Pu_b;
    *Pu_b_mod = *Pu_a;
    *Pu_c_mod = *Pu_c;

    to_pwm();
    kalman_filter(); /*IN: u_al, u_be, i_al, i_be
                       OUT: omega_m_est*/
}

void speed_control(void)
{
/*This routine does the velocity control. It is called from the interrupt
  routine.*/
/*;  input (omega_m_ref);    -- get velocity reference from communication address
       ; This will be given by the user, and will be in a variable.*/

    input_omega_m (/*omega_m*/); /*get position counter and compute rotor velocity*/
/*      this will be measured by the incremental encoder interface in the
```

```
       GPIO */

/*    first order lag to delay velocity reference*/
   v_first_order_lag (/*omega_m_ref, omega_m_ref_delayed*/);

/*    PI - controller for rotor velocity*/
   v_control (/*i_sq_max, v_controller_input, i_sq_ref*/);
}

void c_int4(void) /*main interrupt handler routine */
{
static int co=0;  /* counter for calling speed controller */

   current_control();  /*every 500us*/
   if (0 == (co=(++co)%2))
     speed_control(); /*every 1ms */
}

void init_control(void)
{
 *Pomega_m_ref = 0;          /* clear omega_m_ref*/
 *Pi_sq_max = I_SQ_MAX_K;       /* q current maximum*/
 *Pi_sd_ref = I_SD_REF_K;       /* d current reference is constant*/
}

void init_hardware(void)
{
       ws_init();             /*setting up wait-states for the board */
       *Ppwm_period_reg = PWM_PERIOD;
       pwm_init(M_SYMM, EN_ALL, CHM_SET, CHM_SET, CHM_SET, SW_HARD,
             LOW, LOW, LOW, LOW, 0, PWM_PERIOD, DIR_POS, 0);
/*     gpio_init(15L, CLK_DIV1, 0, 0, CLK_DIV1,
CAP_DSBL,CAP_DSBL,CAP_DSBL,CAP_DSBL,
             0);*/
       ad_init();
       incr_init();
       uart_init(((UART_freq*10)/16/9600+5)/10);
       intr_init();
}

void setspeed()
{
       clrscr();
```

```
        sendstr(header);
        sendstr("\r");
        spd = inputval("Enter speed value", spd, -3000, 3000);
        *Pomega_m_ref = (int)((long)spd*32767/3000);
}


void speedtest(void)
{
char str[20];
int val;
int i;
char c;

    clrscr();
    sendstr(header);
    sendstr("\rSpeed of the motor: 0000000");
    for (c=0; c==0; c = (kbhit() ? getch() : 0) )
        {
        val = (int)((long)(*Pomega_m) * 3000/32767);
        ltoa ((long)val, str);
        sendstr("\b\b\b\b\b\b\b");
        for (i=strlen(str); i<7; i++)
            sendstr(" ");
        sendstr(str);
        } /* for */

}

void i_sqtest(void)
{
/*char str[20];
int val;
int i;
char c;

    clrscr();
    sendstr(header);
    sendstr("\rValue of variable i_sq: 0000000");
    for (c=0; c==0; c = (kbhit() ? getch() : 0) )
        {
        ltoa ((long)(*Pi_sq), str);
        sendstr("\b\b\b\b\b\b\b");
```

```
        for (i=strlen(str); i<7; i++)
            sendstr(" ");
        sendstr(str);
        }*/
}

void i_sdtest(void)
{
/*char str[20];
int val;
int i;
char c;

    clrscr();
    sendstr(header);
    sendstr("\rValue of variable i_sd: 0000000");
    for (c=0; c==0; c = (kbhit() ? getch() : 0) )
        {
        ltoa ((long)(*Pi_sd), str);
        sendstr("\b\b\b\b\b\b\b");
        for (i=strlen(str); i<7; i++)
            sendstr(" ");
        sendstr(str);
        } */
}

void incrtest(void)
{
/*char str[20];
unsigned val;
int i;
char c;

    clrscr();
    sendstr(header);
    sendstr("\rInput from the incremental encoder interface: 0000000");
    for (c=0; c==0; c = (kbhit() ? getch() : 0) )
        {
        val = get_incr();
        ltoa ((unsigned long)val, str);
        sendstr("\b\b\b\b\b\b\b");
        for (i=strlen(str); i<7; i++)
            sendstr(" ");
```

```
        sendstr(str);
        } */

}

void adtest(int ch)
{
char str[20];
int val;
int i;
char c;

    clrscr();
    sendstr(header);
    sendstr("\rInput from the AD converter: 0000000");
    for (c=0; c==0; c = (kbhit() ? getch() : 0) )
        {
        val = Pphase_current[ch];
        ltoa ((long)val, str);
        sendstr("\b\b\b\b\b\b\b");
        for (i=strlen(str); i<7; i++)
            sendstr(" ");
        sendstr(str);
        } /* for */

}

void vartest(void)
{
/*char str[20];
int val;
int i;
char c;

    clrscr();
    sendstr(header);
    sendstr("\rVariable: 0000000");
    for (c=0; c==0; c = (kbhit() ? getch() : 0) )
        {
        val = *Pomega_fr;
        val = Px[0];
        ltoa ((long)val, str);
        sendstr("\b\b\b\b\b\b\b");
```

```
        for (i=strlen(str); i<7; i++)
            sendstr(" ");
        sendstr(str);
        } */
}


void al_be_test(void)
{
char str[20];
int val;
int i;
char c;

    clrscr();
    sendstr(header);
    sendstr("\r i_al, i_be, u_al, u_be:\r");
    for (c=0; c==0; c = (kbhit() ? getch() : 0) )
        {
        ltoa ((long)(*Pi_al), str);
        for (i=strlen(str); i<7; i++)
            sendstr(" ");
        sendstr(str);
        sendstr("; ");
        ltoa ((long)(*Pi_be), str);
        for (i=strlen(str); i<7; i++)
            sendstr(" ");
        sendstr(str);
        sendstr("; ");
        ltoa ((long)(*Pu_al), str);
        for (i=strlen(str); i<7; i++)
            sendstr(" ");
        sendstr(str);
        sendstr("; ");
        ltoa ((long)(*Pu_be), str);
        for (i=strlen(str); i<7; i++)
            sendstr(" ");
        sendstr(str);
        sendstr("\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b");
        } /* for */
}

void msrmnt(void)
```

```
{
char str[20];
int val;
int i;
char c;
int co;

   co = 400;
   *Pomega_m_ref=0;
   for (c=0; c==0; c = (kbhit() ? getch() : 0) )
      {
      if (!(--co))
         {
         co=60;
         *Pomega_m_ref=(*Pomega_m_ref==10923 ? -10923 : 10923);
         }
      val = *Pomega_m;
      ltoa ((long)val, str);
      sendstr(str);
      sendstr(",");
      val = *Pomega_m_ref_delayed;
      ltoa ((long)val, str);
      sendstr(str);
      sendstr(",");
      }
}

void estspdtest(void)
{
char str[20];
int val;
int i;
char c;

   clrscr();
   sendstr(header);
   sendstr("\rEstimated speed of the motor: 0000000");
   for (c=0; c==0; c = (kbhit() ? getch() : 0) )
      {
      val = (int)((long)(*Pomega_m_est) * 3000/32767);
      ltoa ((long)val, str);
      sendstr("\b\b\b\b\b\b\b");
      for (i=strlen(str); i<7; i++)
```

```
            sendstr(" ");
        sendstr(str);
        } /* for */

}

void runmenu(void)
{
char c;
int val, i;
char b[10]="";
        for (;;)
        {
        clrscr();
        sendstr(header);
        sendstr("\rMain Menu\r");
        sendstr("\t1) Set speed reference (currently: ");
        ltoa ((long)spd, b);
        sendstr(b);
        sendstr(")\r\t2) See encoder input\r\t3) See current \"a\"\r"
                "\t4) See current \"b\"\r\t5) See measured speed\r"
                "\t6) Turn PWM generating ");
        sendstr(pwmon ? "OFF" : "ON");
        sendstr("\r\t7) See i_sq\r\t8) Measurement Mode"
                "\r\t9) See estimated speed\r\tA) Vartest"
                "\r\tB) See i_sd"
                "\r\tC) See i_al, i_be, u_al, u_be"
                "\rPlease make your choice\r");
        for (c = 0; (c<'1' || c>'9') && (c<'A' || c>'C');c = toupper(uartgetc()));
        switch (c)
           {
           case '1': setspeed(); break;
           case '2': incrtest(); break;
           case '3': adtest(0); break;
           case '4': adtest(1); break;
           case '5': speedtest(); break;
           case '6': pwmon = 1-pwmon; break;
           case '7': i_sqtest(); break;
           case '8': msrmnt(); break;
           case '9': estspdtest(); break;
           case 'A': vartest(); break;
           case 'B': i_sdtest(); break;
           case 'C': al_be_test(); break;
```

```
        } /* switch */
    }
}

void main(void)
{
    init_control();
    init_hardware();
    runmenu();
}
```

```
; Kalman Filter for the TMS320C5x
; sin.asm
;     Author : Balazs Simor
;     Date :      June, 1996
.globl  _sine

        .data
sintab: .include   sine.tab
tabend:
        .word   0

tabsize .set    tabend - sintab

              .even
temp    .word   0
temp1   .word   0
tempAR3 .word   0

        .text

**********************
* int sine(int angle);
**********************
_sine
        ldp #temp
        mar *-
        lacc *+
        sacl    temp
        lt  temp
        mpy #tabsize
        pac
        sacl    temp1   ; fractional part of place in table in +Q16 format.
        bsar    16      ;position in table is in ACC
        bcnd    poz, GEQ
        add #tabsize
poz:    add #sintab     ;address of lower value in ACC
        sacl    temp
        sar AR3, tempAR3
        lar AR3, temp
        mar     *, AR3
        mar     *+      ;address of higher value in AR3
        lacc    temp1, 14       ;converting +Q16 to Q15
```

```
and     #0FFFFh, 14
sach    temp1, 1
lt      *-      ; higher value
mpy     temp1
lacc    #32767
sub     temp1           ; acc = 1-temp1
sacl    temp1
ltp     *, AR1  ;lower value
mpy     temp1
apac
bsar    15      ; The interpolated output is in Acc
lar AR3, tempAR3  ;restoring AR3
ret
.end
```