# Flux and Speed Estimation for Induction Machines

AN331-29

# Table of Contents

## Summary

Over the last couple of years the market for variable speed drives has escalated drastically. The manufactures have seen the potential in not only controlling the speed or torque range, but also minimizing the consumption of power. This means new types of control-algorithms / schemes are suddenly needed to incorporate these solutions.

During the last years the standard microprocessor has been the most prices competitive for the low-end motor drives. Since the introduction of the DSP micro-controller the manufactures have been able to implement higher level control-solutions as a respond to customer's demands. These advanced control-solutions will not only substitute the standard drive-systems, but also give the customer the opportunity to choose a better and less power requiring system for the same price.

This applications note describes an easy way to calculate the direct stator flux and from this extract the control angle and the rotor speed. This note is intended to be used as one of the building blocks for a complete drive unit. Linking this applications note together with notes like the PWM, ADC and PI applications notes lead the user to create a full vector controlled drive system for an induction machine.

Experimental testing of the flux and speed estimation blocks has been carried out on a full drive-system. Results and drive information is as described in the application note.


# 1 Direct Stator Flux Calculation

The field-oriented control of an induction machine is normally achieved on base of a measured shaft angle. The measurement is used to simplify the calculation of the flux rotating in the machine. With the use of a Motorcontrol DSP, ultra-fast calculation can substitute (simply by adding a theoretical based sensorless algorithm) the measurement of the position in the feedback and a competitive flux-controller can be implemented.

## *1.1 Flux-estimation on base of the introduced Back-EMF.*

The Stator-Flux in the machine, is calculated from the voltages and currents described in [Ref1]

$$\lambda_{alpha} = \int (V_{alpha,corrected} - R_s I_{alpha})dt. \qquad \lambda_{beta} = \int (V_{beta,corrected} - R_s I_{beta})dt. \qquad [1]$$

$$V_{*,corrected} = \frac{V_{DC,meash}}{V_{DC,\max}} * V_{*} \qquad [2]$$

Where,

$R_s$ is the stator resistor and the $V_{"*", corrected}$ is the voltages in the stationary αβ-reference frame corrected to the measured voltage-level of the DC-bus.

The magnitude of the Flux can on base of the motor-symmetry[1] be calculated as:

$$|\lambda_s| = \sqrt{\lambda_{alpha}{}^2 + \lambda_{beta}{}^2} \qquad [3]$$

[1] A three phase symmetrical system will in the reference frame have two quadrate flux components in the stationary αβ-reference frame

And from the magnitude and the rotating fluxes, the transformation angle for the system can be expressed as:

$$COS\theta = \frac{\lambda_{alpha}}{|\lambda_s|} \quad and \quad SIN\theta = \frac{\lambda_{beta}}{|\lambda_s|} \qquad [4]$$

The angle, θ, which can be used in subsequent field oriented or vector control schemes, is the angle of the stator flux vector.

$$T_e = \frac{3}{2} P \,\mathrm{Re}\left[j\lambda_m (i_d - j i_q)\right] = \frac{3}{2} P \lambda_m i_q \qquad [5]$$

This equation indicates that only the $i_q$ component orthogonal to the general flux contributes to a torque production. If the angle in the space phasor reference isn't fixed to the flux the torque production will be less than maximum.

### *1.1.1 Integration (Calculating the Back-EMF)*

With a pure integrator problems like drift and offset are normal. To minimize these aspects a new kind of integration is here introduced. This new integration scheme is a combination of a low-pass filter and a pure integrator, see [Ref5].

$$y = \frac{1}{s} x \qquad [6]$$

Adding a low-pass filter with a cut-off frequency of $\omega_c$ to the integrator can be expressed as [7]. Here the first term of the right hand side represent the low-pass filter and the second term can be described as the feedback compensation for the output of the filter.

$$y = \frac{1}{s+\omega_c} x + \frac{\omega_c}{s+\omega_c} y \qquad [7]$$

Adding these terms together and including a limitation of the feedback magnitude can be represented as

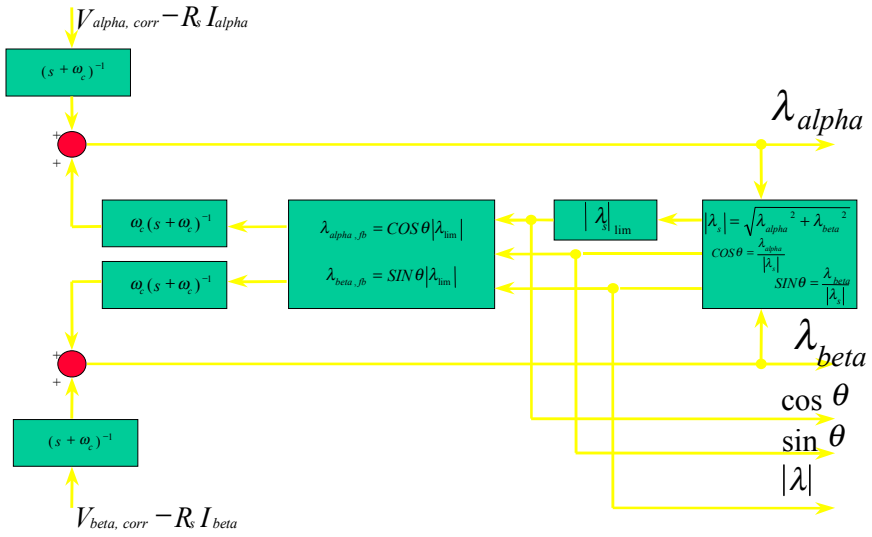


**Figure 1 - Integration with feedback compensation**

Figure 1. Here it can be seen that the output of the integration block is the flux and angle references related to the distributed flux in the motor. The idea with this integration scheme is to minimize the biasing and problems with initialization values to overcome errors in the flux-estimation. As can be seen of Figure 2 the problems with a pure integrator during drift can be illustrated as follows:

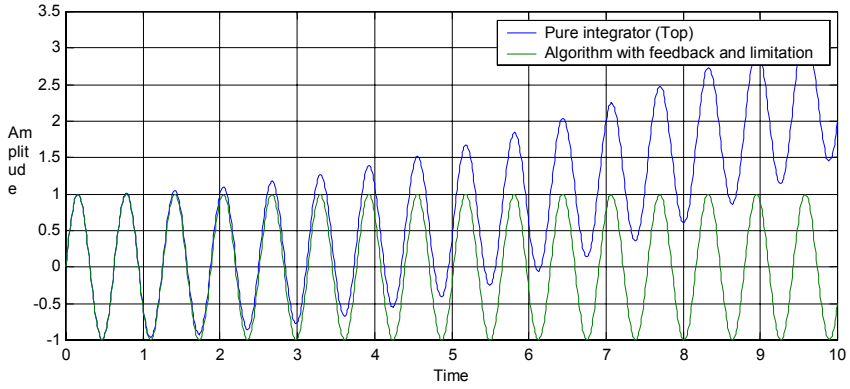


**Figure 2 - drift problems of the integrator**

Here the pure integrator is applied with a dc-offset which will saturate the result the result of the integration. With the use of the algorithm the limitation and the offset-drift are controlled as can be seen on the green waveform.

## *1.2 Speed Calculation*

The speed calculation of the induction machine is as the fluxes bound to the mathematical expression of the motor. For the induction machine the rotor speed can be extrapolated from the synchronous- and the slip-speeds.

$$\omega_r = \omega_{synchronous} - \omega_{slip} = \omega_e / p - \omega_{slip} \qquad [8]$$

From section 1 all the information on the back-EMF and the fluxes in the αβ-frame are calculated. The electrical speed can be from these information be calculated as a cross multiplication of the back-EMFs and fluxes over the magnitude of the resulting flux [9].

$$\omega_{electrical} = \frac{EMF_{beta}\ \lambda_{alpha} - EMF_{apha} \cdot \lambda_{beta}}{\left|\lambda_s\right|^2} \qquad [9]$$

Where,

$$\begin{aligned} EMF_{alpha} &= V_{alpha,corrected} - R_s I_{alpha} \ \ and \\ EMF_{beta} &= V_{beta,corrected} - R_s I_{beta} \end{aligned} \qquad [10]$$

Calculating the slip speed is more closely related to the parameters of the machine than the fluxes and back-EMFs as for the prior flux-estimation. Here the leakage-factor (σ) and the rotor time-constant becomes more critical factors.

$$\omega_{slip} = \frac{L_s \cdot i_{qs}}{T_r \cdot (\lambda_{ds} - \sigma L_s i_{ds})} \quad [11]$$

Here $T_r = L_r/R_r$

The total leakage factor can be expressed as:

$$\sigma = \left(1 - \frac{L_m^{\ 2}}{L_s L_r}\right) \quad [12]$$

where,

$L_m$ is the mutual inductance and $L_s$, $L_r$ are stator and rotor inductances.

## 1.3 Parameters and Input

The problem with calculation of any values in a fixed point DSP is the conversion and scaling of numbers from "real" values (floating-point) to scaled values in the DSP (Fixed-point format). The parameters used for calculation in this note are based on motor-parameter measured on the motor using standard static methods. No adjustment due to parameter change in satuation or temperature is implemented in this applications example.

### 1.3.1 Flux Estimator

As can be seen in section 1.1 the only motor parameter that interacts, with the Flux-estimation is the stator winding resistance ($R_s$). Input values for the calculation are Currents and Voltages in the stationary reference frame.

The value of $R_s$ needs to be calculated and scaled accordingly to the other system parameters. Taking scaling-factors for voltages and current into account allows the fixed-point calculations to operate within maximum scaled range.

Fixed-point values for $R_s$ (any motor) are determined as follows:

$$R_{sfix} = \frac{R_s}{V_{scale} / I_{scale}} \quad [13]$$

where,

$V_{scale}$ The maximum voltage used in the system

$I_{scale}$ The maximum current used in the system

By defining all the values used in the calculation like this, the equations are reduced to a dimensionless per-unit system and the following calculations do not need to handle units.

### 1.3.2 Speed Estimator

The speed-estimator is determined in the same way as the Flux-estimator by motor-specific parameters. As can be seen form the equations in section 1.2, the speed equations also adapt the inductances, leakage-factor and time-constant for the induction machine besides from the calculated fluxes and back-EMFs.

In the same way as in the Flux-estimator values, a per-unit system for inductances can be determined as:

$$L_{sfix} = \frac{L_s}{V_{scale} / I_{scale}} \quad [14]$$

The fluxes and back-EMFs are results calculated in the Flux-estimator and are directly used as input for the speed-estimator. Final values in this estimator is the leakage-factor and additional values used in the calculation. These are:

*Ls/Tr:* Used in the slip calculation

*Leakage*Ls:* Used in the slip calculation

*Flux reference:* Used as level definition of the machine-flux

*Flux limit:* Value used to set the maximum wanted flux in the machine.

*Flux limit sqrt:* Flux limit^2 (Used as checkup in the estimator)

All above mentioned and calculated values can be found in the **"main.h"** file. Here the values are calculated in fixed-point with relation to the actual motor-parameters.

## 1.4 The Complete System

The complete system build up for this applications note is explained on Figure 3. Here the combination of Park and Clarke transformations (AN331-11) and Voltage by Frequency operation (AN331-24) are used to enable the two estimators. Furthermore are the PWM and ADC block used to control and read values from the motor.

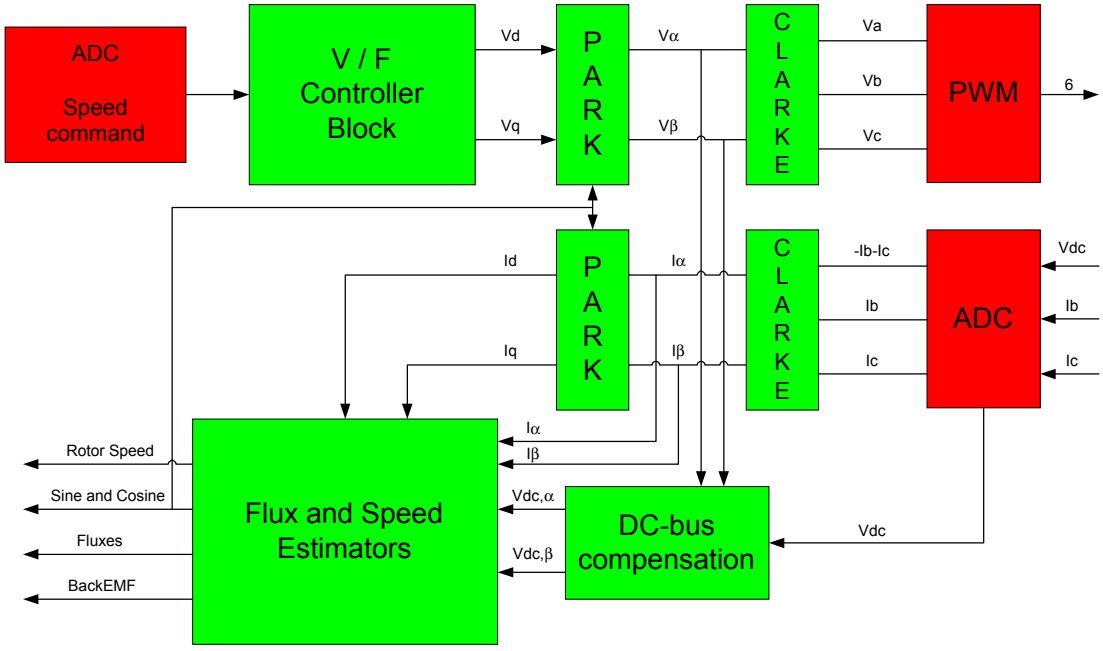


**Figure 3 - Complete model of the Flux and Speed Estimator**

These blocks together form the simple correlation between measured voltages and currents. The two estimator blocks compute the controller-angle (sine and cosine) along with the rotor speed.

## 1.5 Additional Hardware Requirements.

As explained in section 1.2 these estimators are based on the measurement of current and voltages. Therefore it is necessary to add some additional hardware. In this applications note the phase currents are

sensed by the use of two LEM sensors[2] and the DC-voltage are sensed by the PowerIRtrain(power inverter module)[3]. Other sensing techniques can be used to measure the currents and the voltage depending on the application. Here the experimental results are just defined based on these particular sensor configurations.

### *1.6 Experimental plots / Checkup*

When working with a system like an estimator it is very important to ensure that all input-factors and values to the "estimator" are in range and working correctly. The next plots illustrate Voltages, Current

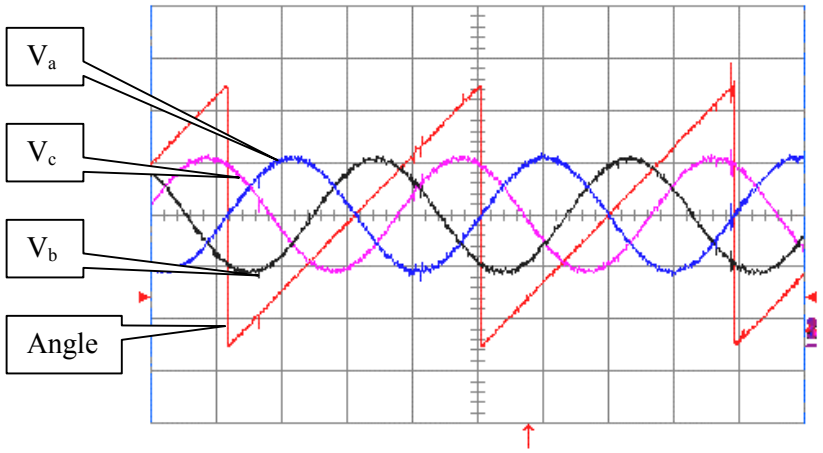


**Figure 4 - Phase voltages Vabc,ref and angle**

and Fluxes calculated with the modules discussed above. As can be seen from Figure 4 - Figure 6 the values used in the calculation are symmetrical and scaled accordingly to the system.

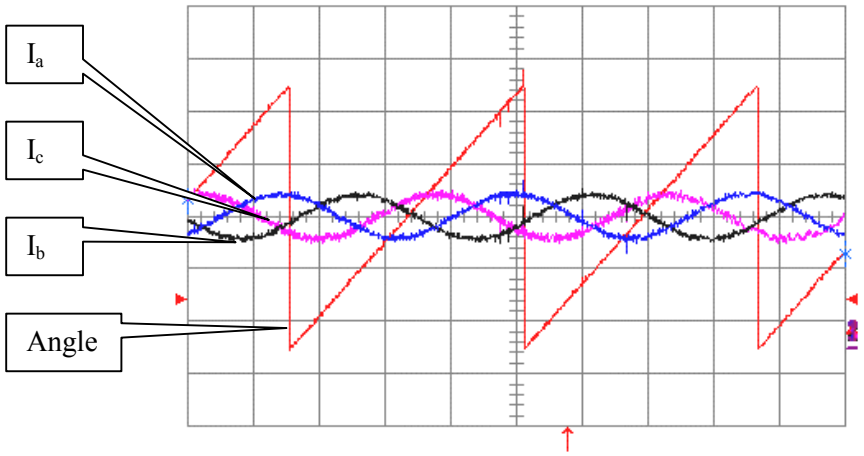


**Figure 5 - Phase Currents Iabc and angle**

[2] For further information see http://www.lem.com/www.nsf

[3] For further information see http://www.irf.com

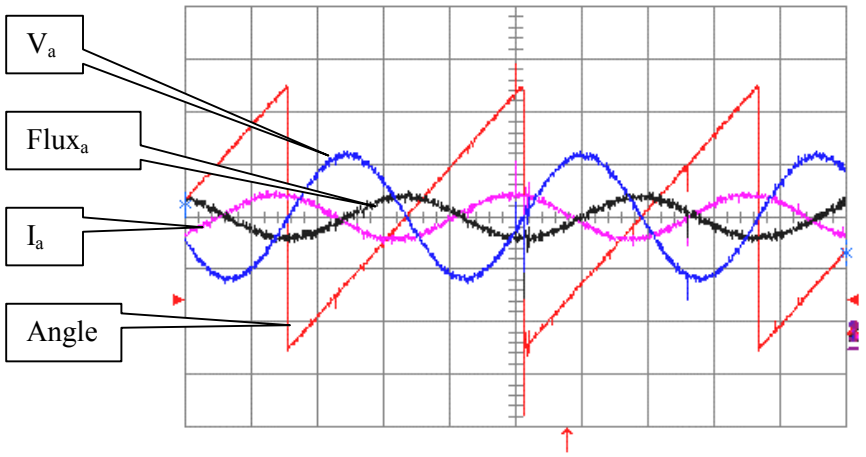


**Figure 6 - Voltage, Flux , Current and angle**

For these tests, the three-phase induction motor is driven open-loop using standard V/F technique (AN331-24). From Figure 4 it can be seen that the three phase voltages form a balanced set with respect to the angle. Additionally, it can be seen from Figure 5, that, for these operating conditions, the motor phases currents also form a balanced three phase system, as expected. Finally, Figure 6 illustrates the phase voltage, the current and estimated flux leakage for one phase of the machine. Clearly, under these operation conditions, the flux leakage is approximately 90° phase shifted from the voltage. This is expected by virtue of the inherent approximation of [15]

$$\psi_a \approx \int (V_a) dt. \qquad [15]$$

# 2 The Estimator Application Routines

## *2.1 Using the Flux and Speed Application Routines*

These application routines provide various functions that configure and enable the ADMC331 part to estimates the stator flux and rotor-speed of an induction machine as described in the previous sections. These estimators can be used in a complete Field Oriented Controller to optimise the torque and speed performance of the IM. With the use of ADIs Standard Motor Control Library (see Library Documentation File) it is possible to link these modules together with dedicated blocks of software to define the complete controller

The routines are developed as easy-to-use blocks, which have to be linked with the used library functions to build the complete application software. The routines for these application routines consist of some files where eight of them are the most relevant. These files are:

| ***File name*** | ***Usage*** |
|---|---|
| Main.dsp (dsp,h) | Set-up of the structure for the PWM generation. Define set-points in frequencies and voltages<br>Defines the external interface to the estimators |
| Cur_Volt (dsp, h) | Calculate offset and scale currents and voltages |
| Flux_est (dsp, h) | Calculate the Flux and back-EMF based on the measured Voltages and Currents |
| Speedest (dsp,h) | Calculate the rotor speed from the Flux, back-EMF and Currents. |

**Table 1: Files used with the two estimator routines**

As with the structure from the ADIs Standard Motor Control Library, macros are defined. For these applications six macros are used for configuration and convenience in the code. The following table defines the set of macros that are defined with this application.

| ***Operation*** | ***Usage*** |
|---|---|
| Calculate the current offsets | Calc_I_Offsets; |
| Read the phase currents with offset | Read_current_with_offset; |
| Read the DC-link voltages - scaled | Read_Vdc; |
| Configuration of the Flux estimator | Flux_Estimation_Init; |
| Configuration of the Speed estimator | Speed_Estimation_Init |
| Estimating the Fluxes and Back-EMFs | Flux_Estimation; |
| Estimating the Speeds of the motor | Speed_Estimation; |

**Table 2: Implemented routines for the two estimator blocks**

As already mentioned in the theory, these routines require some configuration constants, which are declared in a dedicated section of the main include-file "main.h". If a routine requires internal configuration constants, they are declared in the associated include-file "cur_volt.h", "flux_est.h" or

"speedest.h". The following section will explain each of the routines in detail linked with the relevant segments of code that are found in any of the files described in Table 1.

## 2.2 Configuring the Flux Estimator for usage

This routine is called though the "Flux_Estimation_Init" macro and initializes the all the startup values and references to the flux-estimator. Furthermore the 32-Bit filter algorithm for the feedback filter, see section 1.1.1, are reset and initialized.

```
Flux_Estimation_Init_:

    AR = 0;
    dm(Flux_alphabeta)      = AR;   dm(Flux_alphabeta+1)    = AR;
    dm(Flux_Mod)            = AR;
    dm(MODflag)             = AR;
    dm(BackEMF_alphabeta)   = AR;   dm(BackEMF_alphabeta+1) = AR;
    dm(Valphabeta_ref)      = AR;   dm(Valphabeta_ref+1)    = AR;
    dm(SinCos) = AR;

    AR = 0X7FFF;
    dm(SinCos+1)            = AR;

    AR = Flux_Ref;          dm(Flux_Reference)      = AR;
    AR = Flux_Lim;          dm(Flux_Limit)          = AR;
    AR = Flux_LimSqt;       dm(Flux_Limit_sq)       = AR;

    Filter_1st_32_Init(Flux_alpha_Filter_1st_32_Delay);{ reset 32-bit delay line}
    Filter_1st_32_Init(Flux_beta_Filter_1st_32_Delay); { reset 32-bit delay line}
    Filter_1st_32_Init(EMF_alpha_Filter_1st_32_Delay); { reset 32-bit delay line}
    Filter_1st_32_Init(EMF_beta_Filter_1st_32_Delay);  { reset 32-bit delay line}

rts;
```

## 2.3 Configuring the Speed Estimator for usage

This routine is called by the macro "Speed_Estimation_Init" and initializes the all the Speed-estimates along with the fluxes to zero before startup.

```
Speed_Estimation_Init_:
    ar = 0;
    dm(Welectrical) = ar;
    dm(Wslip) = ar;
    dm(Wrotor) = ar;
    dm(Flux_dq) = ar;
rts;
```

## 2.4 Reading and scaling the voltages and currents: Cur_Volt routines

These routines are, as the rest of the programs, structured so that selected pieces of the code can be called though macros in the "Main.dsp"-program. To enable higher flexibility of structure this topology has been chosen. One has to be aware that this code is implemented as a good starting point for other algorithms which, can be linked directly to these routines.

*The first macro "Calc_I_Offsets" calculates the offset of the measured current. This is needed to ensure correct symmetry of the three phases. In the case of offset on the hardware, these are measured and corrected before entering the estimator algorithms. This routine is combined with the "Main.dsp"-file. It reads 16 values of the zero-offset without entering the estimator algorithm and divides the sum of these values by 16 to get an average for each of the currents offset.*

```
{*************************************************************************
* Calc_I_Offsets - measure the two phase current offsets.  This code is only    *
* executed for the first 16 PWM cycles.                                         *
*************************************************************************}
```

```
Calc_I_Offsets_:

    ADC_Read(ADC1);                              { Read value on ADC1        }
    AY0 = dm(Iabc_offset+1);                     { load old Ib_offset        }
    SR = ASHIFT AR BY -4 (LO);                 { divide latest reading by 16   }
    AR = SR0 + AY0;                            { keep running total            }
    dm(Iabc_offset+1) = AR;

    ADC_Read(ADC2);                              { Read value on ADC2        }
    AY0 = dm(Iabc_offset+2);                     { load old Ic_offset        }
    SR = ASHIFT ar BY -4 (LO);                 { divide latest reading by 16   }
    AR = SR0 + AY0;                            { keep running total            }
    dm(Iabc_offset+2) = AR;

Count_Down:
    AR = dm(Count);                            { decrement the count value     }
    AR = AR - 1;
    dm(Count) = AR;

rts;
```

*Second macro " Read_current_with_offset" reads the two sensed currents - Ib and Ic and based on the symmetry the third current is calculated. When the offset has been subtracted a scaling-factor (VI_Scaling) is multiplied with the result to ensure operation within the limited chosen values in "Main.h". Due to the power of a scaling factor bigger than 1, a shift of 1 is made to accomodate the correct scaling.*

```
{*****************************************************************************
* Calculate the three phase currents based on the two measured. Ia = -(Ib+Ic)   *
{*****************************************************************************}
Read_current_with_offset_:

    MY0 = VI_Scaling;              { scaling factor for right Current reading  }
                                   { ratio between Current and Voltagesensing  }
    ADC_Read(ADC1);                { Read value on ADC1                        }
    AY1 = DM(Iabc_offset+1);
    AR = AR - AY1;
    MR = AR*MY0 (SS);
    SR = ASHIFT MR1 BY 1 (HI);
    DM(Iabc+1) = SR1;                            { Store in Ib               }
    AX0 = SR1;

    ADC_Read(ADC2);                              { Read value on ADC2        }
    AY1 = DM(Iabc_offset+2);
    AR = AR - AY1;
    MR = AR*MY0 (SS);
    SR = ASHIFT MR1 BY 1 (HI);
    DM(Iabc+2) = SR1;                            { Store in Ic               }
    AY1 = SR1;

    AR =  AX0 + AY1;
    AR = - AR;
    DM(Iabc) = AR;                               { Store in Ia               }
rts;
```

*The last macro "Read_Vdc" ensure the reading of the DC-voltages is correct and scaled accordingly to that specified in "Main.h". Underneath is the routine on every conversion updates the Vdc_scaling factor. This correction of the DC-voltage measurement ensures correct scaled voltage values independent of variations in the DC-link voltages due to load.*

```
{*****************************************************************************
*   Measure the bus-voltage for correction of the x/y-frame voltages.          *
*****************************************************************************}
Read_Vdc_:
ADC_Read(ADC3);                          { Read value on ADC3            }
    MY0 = dm(Vdc_max_inv);
    MR = AR*MY0 (SS);                    { calculate Vdc * 1/Vdc(max)    }
    dm(Vdc_scaling) = MR1;

rts;
```

## 2.5 Calculating the fluxes and back-EMFs of the Motor: Flux_est routines

These routines are implemented to enable the calculations of the fluxes and thereby the control-angle of an induction-machine. The algorithm is based on the mathematical equations from section 1.1 and are combined with the usage of a low-pass filter (AN331-33).

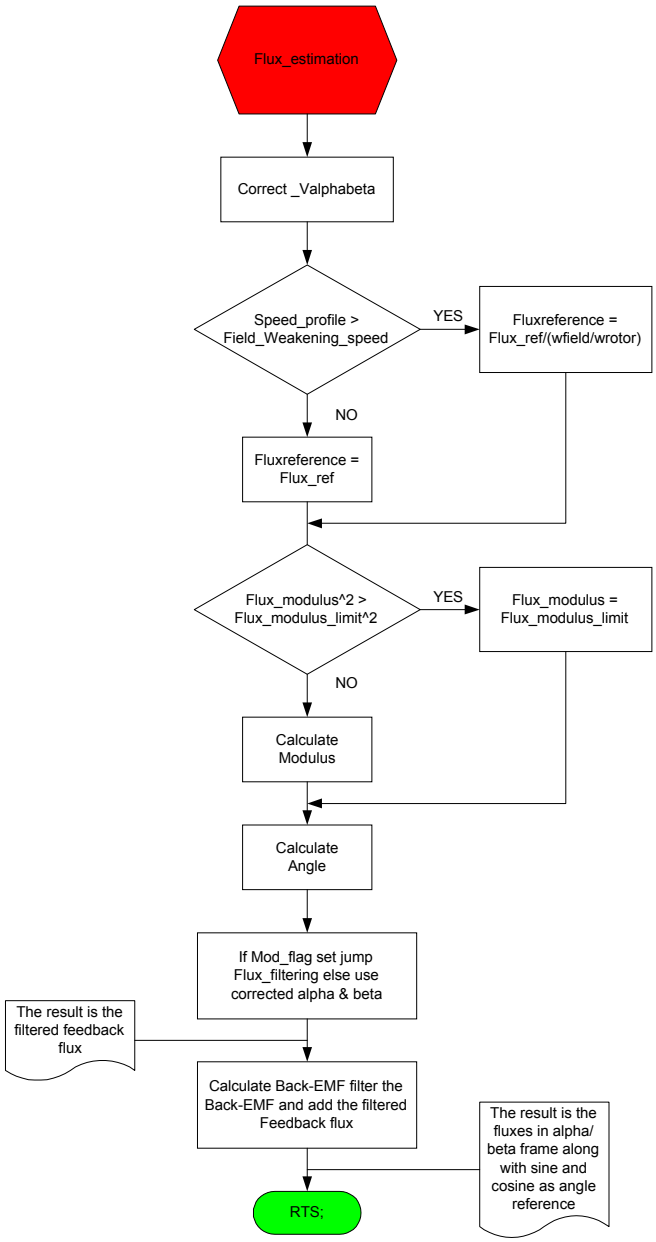


**Figure 7 - Flow-diagram for the Flux Estimator**

*The code is almost self documenting one just has to remember all the scaling factors from the definition in "Main.h". Furthermore the low-pass filter used to minimize drift and offset as discussed in section 1.1.1 is defined for 5 Hz cut-off frequency. See "Main.h".*

```
{*****************************************************************************
* Flux estimation from Flux_alphabeta                                        *
*****************************************************************************}
Flux_Estimation_:

    Call Correct_Valphabeta_;   { correct voltages with DC-link drop          }

    AY0 = Flux_Ref;
    AY1 = dm(Speed_profile);
    AX0 = FIELD_WEAKENING_SPEED;{ Calculated in Main.h                        }
    AF = AX0 - AY1;
    IF LT CALL FieldWeakening_;

    dm(Flux_Reference) = AY0;   { store flux reference, either base or field  }
                                { weakened                                    }
                                { Flux_Limit_sq = Flux_alpha^2 + Flux_beta^2 }
    SR1 = dm(Flux_alphabeta);   { alpha                                       }
    MR1 = dm(Flux_alphabeta+1); { beta                                        }
    MY0 = MR1;
    MR = MR1*MY0 (SS);          { beta^2                                      }
    MY0 = SR1;
    MR = MR + SR1*MY0 (SS);     { MR = alpha^2 + beta^2                       }
    IF MV SAT MR;

    AY0 = DM(Flux_Limit_sq);    { if the calculated stator flux magnitude is  }
                                { greater                                     }
    AF = MR1 - AY0;             { than the limit in Flux_Limit_sq, then use   }
                                { the limit                                   }
    IF LT JUMP Calc_Modulus;    { value.                                      }
    AR = DM(Flux_Limit);
    DM(Flux_Mod) = AR;
    dm(Flux_Mod_sq) = ay0;
    AR = 1;
    DM(MODflag) = AR;
    JUMP Calculate_Angle;


{*****************************************************************************
* Calculate the Modulus of the flux                                          *
*****************************************************************************}
Calc_Modulus:

    dm(Flux_Mod_sq) = MR1;            { store calculated flux modulus squared }

    AR = 0;
    DM(MODflag) = AR;
    DM(save_I+6) = I6;
    DM(save_M+6) = M6;
    DM(save_L+6) = L6;
    M6 = 1; L6 = 0;
    SR = ASHIFT MR1 BY -1 (HI);

    Square_Root(SR1,0x0);               { SR1 = SQRT[alpha^2+beta^2]          }

    DM(Flux_Mod) = SR1;                 { save MOD(alpha,beta)                }
    I6 = dm(save_I+6);
    M6 = dm(save_M+6);
    L6 = dm(save_L+6);


{*****************************************************************************
* Calculate the Angle from Flux_alphabeta and SIN,COS                        *
*****************************************************************************}
```

```
Calculate_Angle:                        { Calc Angle Sine and Cosine            }

    AY1=dm(Flux_alphabeta+1);
    AX0=dm(Flux_Mod);
    Signed_Division(AY1,0x0,AX0);
    DM(SinCos) = AR;

    AY1=dm(Flux_alphabeta);
    AX0=dm(Flux_Mod);
    Signed_Division(AY1,0x0,AX0);
    DM(SinCos+1) = AR;

    SR1 = dm(Flux_alphabeta);
    MR1 = dm(Flux_alphabeta+1);
    AR = dm(MODflag);
    AR = PASS AR;
    IF EQ JUMP Flux_Filtering;

{*****************************************************************************
* If MODflag is set,then calculate Flux_xs,ys from the constant Flux_Reference  *
* and sine / cosine                                                             *
*****************************************************************************}
Last_Transf:

    MX0 = dm(SinCos+1);
    MY0 = dm(Flux_Reference);
    MR  = MX0*MY0 (SS);
    SR1 = MR1;                  { alpha }
    MX0 = dm(SinCos);
    MR  = MX0*MY0 (SS);         { beta }
{*****************************************************************************
* The filtered feedback                                                         *
*****************************************************************************}
Flux_Filtering:

    AR = MR1;                                   { reload beta                   }
    Filter_1st_32(Flux_beta_Filter_1st_32_Delay, Flux_Filter_1st_32_Coef);
    AR = SR1;                                   { store SR1                     }
    SR = ASHIFT MR1 BY -1 (HI);
    MR1 = SR1;
    SR1 = AR;                                   { restore SR1                   }
    dm(Flux_alphabeta_FB+1) = MR1;
    AR = SR1;                                   { reload alpha                  }
    Filter_1st_32(Flux_alpha_Filter_1st_32_Delay, Flux_Filter_1st_32_Coef);
    SR = ASHIFT MR1 BY -1 (HI);
    MR1 = SR1;
    dm(Flux_alphabeta_FB) = MR1;

{*****************************************************************************
* Add the filtered flux to the flux                                             *
*****************************************************************************}

EMF_Filtering:
    Call Calculate_BEMF_;
    AR = DM(BackEMF_alphabeta);
    SR = ASHIFT AR BY -1 (HI);
    AR = SR1;
    Filter_1st_32(EMF_alpha_Filter_1st_32_Delay, EMF_Filter_1st_32_Coef);
    AY1 = DM(Flux_alphabeta_FB);
    AR = MR1 + AY1;
    DM(Flux_alphabeta) = ar;

    AR = DM(BackEMF_alphabeta+1);
    SR = ASHIFT AR BY -1 (HI);
    AR = SR1;
    Filter_1st_32(EMF_beta_Filter_1st_32_Delay, EMF_Filter_1st_32_Coef);
    AY1 = DM(Flux_alphabeta_FB+1);
    AR = MR1 + AY1;
    DM(Flux_alphabeta+1) = AR;

rts;
```

*This routine is correcting the Valpha/beta depending on changes on the DC-Link Voltage.*

```
{*****************************************************************************
* Correct Valphabeta with the measured Vdc                                   *
*****************************************************************************}

Correct_Valphabeta_:

    Read_Vdc;

    MX0 = dm(Vdc_scaling);      {Vdc_alphabeta=Valphabeta*Vdc(measured)/Vdc(max)}

    MY0 = DM(Valphabeta_ref);
    MR = MX0*MY0 (SS);
    SR = ASHIFT MR1 BY 1 (HI);  { due to scaling from main.h                    }
    dm(Vdc_alphabeta) = SR1;

    MY0 = DM(Valphabeta_ref+1); {Vdc_alphabeta=Valphabeta*Vdc(measured)/Vdc(max)}
    MR = MX0*MY0 (SS);
    SR = ASHIFT MR1 BY 1 (HI);  { due to scaling from main.h                    }
    dm(Vdc_alphabeta+1) = SR1;

rts;
```

*Calculation of the Back-EMF. Here the values for Rs, Vdc_alphabeta and Ialphabeta are used to calculate the first term in the Flux estimator.*
*Last section explains the field-weakening region where the flux reference is lowered to scale down the magnetization of the motor.*

```
{*****************************************************************************
* Calculate the Backe EMF on base of the currents and voltagedrop            *
*****************************************************************************}

Calculate_BEMF_:

    MY0 = dm(Rs);                   { calculate back EMFs                      }
    AX0 = dm(Vdc_alphabeta);
    AR  = dm(Ialphabeta);
    MR  = AR*MY0 (SS);
    AY0 = MR1;
    AR = AX0 - AY0;
    dm(BackEMF_alphabeta) = AR;     {EMFalpha = Vdc_alpha - Ialpha * Rs         }

    AX0 = dm(Vdc_alphabeta+1);
    AR = DM(Ialphabeta+1);
    MR = AR*MY0 (SS);
    AY0 = MR1;
    AR = AX0 - AY0;
    dm(BackEMF_alphabeta+1) = AR;   {EMFbeta = Vdc_beta - Ibetas * Rs           }
rts;

{*****************************************************************************
*                                                                            *
*   Input:  Wrotor                                                           *
*   Output: MOD_FluxRef_s                                                    *
*   In the fieldweakening region the base speed is divided with the speed    *
*   setpoint and the factor is multiplied  with Flux_Ref.                    *
*****************************************************************************}

FieldWeakening_:

    AX0 = dm(Wrotor);
    Signed_Division(FIELD_WEAKENING_SPEED,0,AX0);

    MY0 = Flux_Ref;
    MR  = AR*MY0 (SS);          {Division * FluxRef }
    AY0 = MR1;

rts;
```

## 2.6 Calculating the speeds of the motor: Speedest routines

These routines are as the flux-routines implemented to enable the calculations of the speeds in the machine. The algorithm is based on the mathematical speed equations from section 1.2 and are compared to the flux (angle) - estimator much more depending on parameters. Here as can be seen in section 1.2 the inductances and resistors of the motor are needed. The problem with these speed calculations is mainly the temperature rise that changes the rotor-time constant dramatically. If this happen (as it does with high load these equations becomes inaccurate and extraordinary parameter calculations has to be added. This are not discussed in this applications note.

*Here the calculations are fixed to the d/q-frame. Fluxes are firstly transformed and with the use of the parameters of the machine the slip-speed is calculated. From the Fluxes and Back-EMFs along with the knowledge of the flux-amplitude the electrical speed can also be calculated. This speed divided by the ratio of polepair is equal to the synchronous speed of the machine. Finally the Rotor speed can be extrapolated be subtracting the slip speed from the electrical speed.*

```
Speed_Estimation_:

{*****************************************************************************}
{   Slip estimation                                                           }
{   Forward Park transform on the flux  -  Use the angle and the two x/y-frame  }
{   fluxs.                                                                    }
{*****************************************************************************}

    refframe_Set_DAG_registers_for_transformations;

    refframe_Reverse_Park_SinCos(Flux_alphabeta,Flux_dq,SinCos);

    sr1=dm(Flux_dq);                          { reload Flux_dq                }
    mx0 = Leakage_Ls;                         { calculate denominator         }
    my0 = dm(Idq);
    mr = mx0 * my0 (SS);
    if mv sat mr;
    ax1 = mr1;
    ay1 = sr1;
    ar = ay1 - ax1;     { Denominator in ar = Flux_ds - leakage * Lsfix * Ids   }

    mx0 = Ls_over_Tr;                         { calculate numerator           }
    my0 = dm(Idq+1);
    mr = mx0 * my0 (SS);

    Signed_Division(MR1,MR0,AR);

    dm(Wslip) = ar;

{*****************************************************************************
*   Electrical speed estimation.                                             *
*****************************************************************************}

Welectrical_estimation:

    mx0 = dm(BackEMF_alphabeta+1);
    my0 = dm(Flux_alphabeta);
    mr = mx0 * my0 (SS);

    mx0 = dm(BackEMF_alphabeta);
    my0 = dm(Flux_alphabeta+1);
    mr = mr - mx0 * my0 (SS);
    if mv sat mr;

    ar = dm(Flux_Mod_sq);
    sr = ashift ar by 1 (lo);              { divided by 2, number of pole pairs}
    Signed_Division(MR1,MR0,SR0);          { result=Welectrical(per pole-pair) }
                                           { =syncronous speed                 }

    dm(Welectrical) = ar;
```

```
{*****************************************************************************
*   Rotor speed calculation.                                                 *
*****************************************************************************}

Wrotor_estimation:
    ax1 = dm(Welectrical);
    ay1 = dm(Wslip);
    ar = ax1 - ay1;
    dm(Wrotor) = ar;
rts;
```

# 3 The main program: Main.dsp

The file "main.dsp" contains the initialisation and PWM Sync and Trip interrupt service routines. To activate, build the executable file using the attached **build.bat** either within your DOS prompt or clicking on it from Windows Explorer. This will create the object files and the **main.exe** example file. This file may be run on the Motion Control Debugger.

In the following, a brief description of this is given.

*Start of code – declaring start location in program memory*

```
.MODULE/RAM/SEG=USER_PM1/ABS=0x30      Main_Program;
```

*Next, the general systems constants and PWM configuration constants (main.h – see the next section) are included. Also included are the Library functions for the PWM, ADC, DAC, Transformations to D/Q - Alpha / Beta, and of course the applications specific routines - v_f_ctrl, ramps and routines related to the speed and flux-estimators.*

```
#include <main.h>;                          {specific constants for this program}

#include <pwm331.h>;
#include <adc331.h>;
#include <autocal.h>;
#include <dac331.h>;
#include <mathfun.h>;
#include <refframe.h>;

#include <v_f_ctrl.h>                       { applications specific constants   }
#include <ramps.h>;                         { applications specific constants   }
#include <cur_volt.h>;                      { applications specific constants   }
#include <flux_est.h>;                      { applications specific constants   }
#include <speedest.h>;                      { applications specific constants   }
```

*First the PWM block initialisation. Note how the interrupt vectors for the PWMSync and PWMTrip service routines are passed as arguments. Secondly, setting the corresponding bit in the IMASK register enables the IRQ2 interrupt. Then initialisation of the ADC, AutoCalibration and DAC block is completed. The next step is to initialise the profiles used in the speed definition along with the flux and speed initialisation of the estimators. Lastly before entering a loop which just waits for interrupts the counter and the offsets of the currents are initialised .*

```
Startup:

    PWM_Init(PWMSYNC_ISR, PWMTRIP_ISR);

    IFC = 0x80;                                 { Clear any pending IRQ2 inter. }
    ay0 = 0x200;                                { unmask irq2 interrupts.       }
    ar = IMASK;
    ar = ar or ay0;
    IMASK = ar;                                 { IRQ2 ints fully enabled here  }

    ADC_Init;
    AUTOCAL_Init;
    DAC_Init;
```

```
    Ramps_Init_Speed_profile;
    Flux_Estimation_Init;
    Speed_Estimation_Init;

{*****************************************************************************
* Initialize the counter and Offset for the offset calibration               *
*****************************************************************************}
    ax0 = 16;
    dm(Count) = ax0;
    ax0 = 0x0000;
    dm(Iabc_Offset+1) = ax0;
    dm(Iabc_Offset+2) = ax0;

{*****************************************************************************}

MAIN:                           {Wait for interrupt to occur}
        nop;
        nop;
        jump MAIN;

RTS;
```

*The first thing that is done in the PWMSYNC_IRS is the Autocalibration. Then the DAC is paused to ensure no pointer conflict. To ensure that the currents offsets are correct the offsets of the hardware are in the next 16 cycles measured and averaged.*

*Now the currents can be read and corrected with the correct offset **"Read_Current_with_Offset".** Now the phase currents are corrected and the transformation to the alpha/beta frame can be enabled. The input to the flux-estimator is now available and the complete flux and angle estimation is done. When the angle (sine and cosine) are calculated the park transformation into the d/q frame can be enabled and finally the speed estimation can be called.*

*To test the complete system a standard V/F system are defined. This system is controlled by a Speed_command read though the converter on ADCAUX1 and used as set-point for the V/F control. With the call of Set_Minimum_Speed the minimum selected speed (see "main.h") is selected. Using **ramps_Calculate_Speed_Profile(0x4,0x4fff)** the acceleration time is set to 10 seconds and from here the Speed_profile value is used to calculate the V/F Angle and the V/F Voltages. Finally the complete PWM sequence is calculated with the use of the macro **PWM_update_demanded_Voltage.***

*For evaluation of the estimated speeds along with the control angle (sine and cosine) these values are plotted to the first four DACs.*

```
{*****************************************************************************
* PWM Interrupt Service Routine                                              *
*****************************************************************************}

PWMSYNC_ISR:

    AutoCal_Calibrate;                      { First PWM cycle              }
    DAC_Pause;

{*****************************************************************************
* Calculate the offset of the Channels for Current calculation               *
*****************************************************************************}
Count_16:                       { PWMSYNC cycles 2-17 measure the current off   }
    AR = DM(Count);
    AR = pass AR;
    if eq jump Offset_saved;
    Calc_I_Offsets;             { Macro that calculates current offset          }
    jump END_PWM;

{*****************************************************************************
* When offset is saved - do the reading of all currents                      *
*****************************************************************************}

Offset_saved:
    Read_Current_with_Offset;   { Macro that reads phase current with offset    }
```

```
{*****************************************************************************
* Forward Clark-transformation use the 3 phase currents to calculate Ix and Iy  *
*****************************************************************************}
    refframe_Set_DAG_registers_for_transformations;
    refframe_Forward_Clarke(Iabc,Ialphabeta);

{*****************************************************************************
* Call the FLUX estimation block to calculate angle and fluxes                  *
*****************************************************************************}
    Flux_Estimation;

{*****************************************************************************
* Reverse Park-transformation use the 2 Currents_alphabeta to calculate Idq     *
*****************************************************************************}
    refframe_Set_DAG_registers_for_transformations;
    refframe_Reverse_Park_SinCos(Ialphabeta,Idq,SinCos);

{*****************************************************************************
* Call the SPEED estimation block to calculate Electrical and mechanical speed  *
*****************************************************************************}
    Speed_Estimation;

{*****************************************************************************
* Read speed-command from POT                                                   *
*****************************************************************************}
    ADC_Set_AUXch(1);           { Select Auxiliary channel 1 as analog input    }
    ADC_Read(ADCAUX);           { Read value on ADCAUX1                         }
    dm(Speed_command) = ar;     { Store in Speed_command                        }

    AR = abs AR;                { Check if speed is in the minimum speed range  }
    AY0 = Minimum_speed;
    AR = AR - AY0;
    if ge jump Over_Min_Speed;  { if speed_Command< Minspeed jump Over_Min_Speed}

    MY0 = Minimum_speed;
    dm(Speed_command) = MY0;

Over_Min_Speed:
{*****************************************************************************
* Calculate values from ramps and set the angle calculation                     *
*****************************************************************************}
    ramps_Calculate_Speed_Profile(0x4,0x4fff);  { 10 sec.  see ramps.h          }
    V_F_ctrl_CALCULATE_ANGLE_VOLT;

{*****************************************************************************
* Do the complete SVM scheeme for the PWM block                                 *
*****************************************************************************}
    V_F_ctrl_PWM_CALCULATION;                   { This is without SVM           }

{*****************************************************************************
* Do the complete PWM scheeme for the PWM block                                 *
*****************************************************************************}
    ax0 = DM(Vabc); ax1 = DM(Vabc+1); ay0 = DM(Vabc+2);
    PWM_update_demanded_Voltage(ax0,ax1,ay0);

{*****************************************************************************
* Resume the DAC - Use the DAC as debugger option for the code                  *
*****************************************************************************}
    DAC_resume;

PLOT:
    MY0 = DM(Wrotor);               Dac_Put(1, MY0);
    MY0 = DM(Wslip);                Dac_Put(2, MY0);
    MY0 = DM(SinCos);               Dac_Put(3, MY0);
    MY0 = DM(SinCos+1);             Dac_Put(4, MY0);

DAC_Update;

END_PWM:

RTI;
```

*The PWM-Trip routine is in this example used to check on the trip pin on the PowerIR-train[4]. This TRIP-pin is hardwired to the PWM_TRIP_PIN on the ADMC331 device. When the PowerIRtrain's pin goes low, in the case of **over-current** or **temperature** the PWMTRIP_ISR check on the status of the pin. The actions are as given*

1. *Check the PWMTRIP in the SYSSTAT – if high jump to restart*
2. *If not wait 80 μs – and then check SYSSTAT again*
3. *If it now has gone high call restart PWM .....*

```
PWMTRIP_ISR:

{*****************************************************************************
* PWM Trip Interrupt Service Routine                                         *
*****************************************************************************}
Trip_Ena:

    CNTR = H#3FF ;
    DO Wait0 UNTIL CE;         { wait 80us}
Wait0:  NOP;

    Test_Bit_DM(SYSSTAT,0); { check the PWMTRIP input. Still low  ?            }
    If_Clr_Jump(Trip_Ena);

    Test_Bit_DM(SYSSTAT,0); { check the PWMTRIP input again. Gone high restart  }
    If_Set_Jump(RESTART_PWM);

    DIS SEC_REG;

    RTI;

{*****************************************************************************
* After a shutdown - restart the PWM.                                        *
*****************************************************************************}

RESTART_PWM:

    IFC = 0X80;                                        { clear IRQ2 interupt   }
    PWM_Init(PWMSYNC_ISR, PWMTRIP_ISR);
    AR = 0;                          { clear SPEED_PROFILE to ensure SAFE start  }
    DM(SPEED_PROFILE)=AR;


RTI;

.ENDMOD;
```

## 3.1 The main include file: main.h

This file contains the definitions of ADMC331 constants, general-purpose macros and the configuration parameters of the system and library routines. It should be included in every application. For more information refer to the "The Library Documentation File" document.

*This file is mostly self-explaining. The relevant sections to this example are shown here. The frequency of the used crystal (12.96MHz in case of the ADMC331 Evaluation Kit) is expressed in kHz. Then ADMC331 specific constants, ROM-Utilities and general-purpose macros are included. Refer to the ADMC331 documentation for details on the ROM-Utilities.*

```
{*****************************************************************************
* General System Parameters and Constants                                    *
*****************************************************************************}
```

[4] For more information look in datasheet on IRPTXXX family at www.irf.com

```
.CONST  Cry_clock      = 12960;     { Crystal clock frequency [kHz]             }

#include <admc331.h>;
#include <romutil.h>;     { included because of compatibiliy with ROMUTIL users }
#include <macro.h>;
```

*As described in the "The Library Documentation File", every library routine has a section in main.h for its configuration parameters. The following defines the parameters for the **cur_volt** block and the **flux and speed estimator** blocks used in this example. All the parameters are values measured on the chosen motor. The calculation of the Constants ".CONST" are done by hand –but can be handled by any other program. These values need to be corrected for any other motor. In this example a Bodine motor (type 34R6BFPP) are used for experimental results.*

```
{*************************************************************************}
{ Library: Current_Voltage Block                                          }
{ file   : Cur_Volt.dsp                                                   }
{ Application Note:                                                       }

{ VOLTGAE DIFINITION:                                                     }
{ Vscale        = 330;  { measured DC bus voltage, DC                     }
{ Vline         = 230;  { motor rating, line to line voltage, Vrms        }
{ VphaseRMS     = Vline / sqrt(3);                                        }
{ Vphase        = VphaseRMS * sqrt(2);                                    }
{ Vmax          = min(0.5, (Vphase*sqrt(2))/Vscale);                      }
{ Vmax_sqr      = Vmax^2;                                                 }

{ CURRENT DIFINITION:                                                     }
{ Imax          = 1.2;              { motor rated current, Amps RMS       }
{ Iscale        = Imax * sqrt(2);   { motor rated current, Amps peak      }

{ MEASUREMENT DIFINITION:                                                 }
{ VadcMax = 3.5;        ADMC331 parameter (See converter note)            }
{ VadcMin = 0.3;        ADMC331 parameter (See converter note)            }
{ VadcMid = (VadcMax - VadcMin)/2 + VadcMin;                              }
{ Sense_Ratio = 0.825 (Ratio for the current sensing / hall-effect sensors [V/A]}
{ Offset      = 1.8 ( signal offsetted around [ V ])                      }
{ Current_Scaling_Input = Iscaled * Sense_Ratio [V]                       }

{ Maximum_VI_input = ((Current_Scaling_Input+Offset)/VadcMax)  ~ 3.2/3.5   }
{ VI_Scaling = ((1/Maximum_VI_input);                                     }
{ Note divide by 2 to keep less than 1                                    }

{ VdcIn         = 2.97; Measured 330 V                                    }
{ VdcMax        = (VdcIn/VadcMax)                                         }
{ Vdc_Inverse   = 1/(VdcMax)/2;                                           }
{ Note divide by 2 to keep less than 1                                    }

{*************************************************************************
* The constants are calculated on base of above equations                *
*************************************************************************}

.CONST  VI_Scaling  = 0x45FF;                  { Calculated from above    }
.CONST  Vdc_Inverse = 0x4B6A;                  { Calculated from above    }

{ MOTOR PARAMETER DIFINITION:                                             }
{ Rs = 14.6         ( Ohms, per phase stator resistance [ Ohm ]       )   }
{ Rr = 12.77;       ( Ohms, per phase rotor winding resistance [ Ohm ])   }
{ L2 = 51.8e-3;
{ Lr = 348.2e-3;    ([H], per phase rotor self inductance: L2+Lm    )     }
{ L1 = 22.2e-3;     ([H], stator leakage inductance                 )     }
{ Ls = 318.5e-3;    ([H], per phase stator self inductance: L1+Lm   )     }
{ Lm = 296.3e-3;    ([H], per phase magnetizing inductance          )     }
{ Tr=  Lr/Rr;        - rotor time constant                                }
{ Leakage=1-Lm^2/(Ls*Lr) - total leakage factor                           }

{ Rsfix = Rs/(Vscale/Iscale);      14.6/194.5 =7.5E-2                     }
{ Lsfix=Ls/(Vscale/Iscale);                                               }
{ Ls_over_Tr= (Lsfix/Tr);                                                 }
{ Leakage_Ls= (Leakage * Lsfix);                                          }

.CONST  Rsfix       = 0x99B;                   { Calculated from above    }
```

```
.CONST  Lsfix       = 0x34;                     { Calculated from above         }

.CONST  Ls_over_Tr  = 0x7FA;                    { Calculated from above         }
.CONST  Leakage_Ls  = 0xA;                      { Calculated from above         }

{*****************************************************************************}
{ Library: Filter block for the flux-estimation                               }
{ file   : Filter.dsp                                                         }
{ Application Note:     IIR filters                                           }
{ Definition:                                                                 }
{ LowPassFilter:    y(n)=b.x(n) + b.x(n-1) - a.y(n-1)                         }
{ Wc=cutoff freq:   1/(s+Wc); Wc/(1+Wc)                                       }
{*****************************************************************************}
{ Parameter difinition:                                                       }
{ NumPoles = 4;                                                               }
{ MaxRotorSpeed = 2400;                                                       }
{ MaxFrequency = (NumPoles/2)*MaxRotorSpeed/60;                               }
{ omega_base=2*pi*BaseFrequency/Wscale                                        }
{*****************************************************************************}
{ W0=2*pi*5;        In this case 5Hz                                          }
{ T=1/PWM_freq (Sample Time)                                                  }
{ a0=(W0*T)/(2+W0*T)        Coeffiecient a0 Flux_filter                       }
{ a1=(W0*T)/(2+W0*T)        Coeffiecient a1                                   }
{ b0=(2-W0*T/(2+W0*T))      Coeffiecient b0                                   }
{ Wscale=2*pi*MaxFrequency = 502.65                                           }
{ a0=(Wscale*T)/(2+Wscale*T)        Coeffiecient a0 EMF_filter                }
{ a1=(Wscale*T)/(2+Wscale*T)        Coeffiecient a1                           }

{*****************************************************************************}
{ Defined for flux_est.dsp                                                     }

.CONST A0_Flux   =  0x003200;{ added to zeros in lower bit .... see filter note }
.CONST A1_Flux   =  0x003200;{ added to zeros in lower bit .... see filter note }
.CONST B0_Flux   =  0x7F9800;{ added to zeros in lower bit .... see filter note }

.CONST A0_EMF    =  0x033500;{ added to zeros in lower bit .... see filter note }
.CONST A1_EMF    =  0x033500;{ added to zeros in lower bit .... see filter note }
.CONST B0_EMF    =  0x7F9800;{ added to zeros in lower bit .... see filter note }

{*****************************************************************************}
{ Library: Flux Estimation BLOCK                                              }
{ file   : Flux_est.dsp                                                       }
{ Application Note: Flux and Speed estimation on an induction machine         }

{ Description of parameters:                                                  }
{ FluxRef   = (Vmax / omega_base);                                            }
{ FluxLim   = 1.2 * FluxRef;                                                  }
{ FluxLimSqt    = FluxLim * FluxLim;                                          }
{ Field_weakening_speed = 2*pi*BaseFrequency/Wscale                           }

{*****************************************************************************}

.CONST Flux_Ref         =  0x5555;              { Calculated from above         }
.CONST Flux_Lim         =  0x6666;              { Calculated from above         }
.CONST Flux_LimSqt      =  0x51E5;              { Calculated from above         }

.CONST FIELD_WEAKENING_SPEED =0x6000;

{*****************************************************************************}
```

# 4 Experimental results of the two estimators.

The two plots below illustrate the calculated fluxes, back-EMF, speed and control angles. These results are taken at the maximum chosen speed 80 Hz but can be calculated on any induction machine.

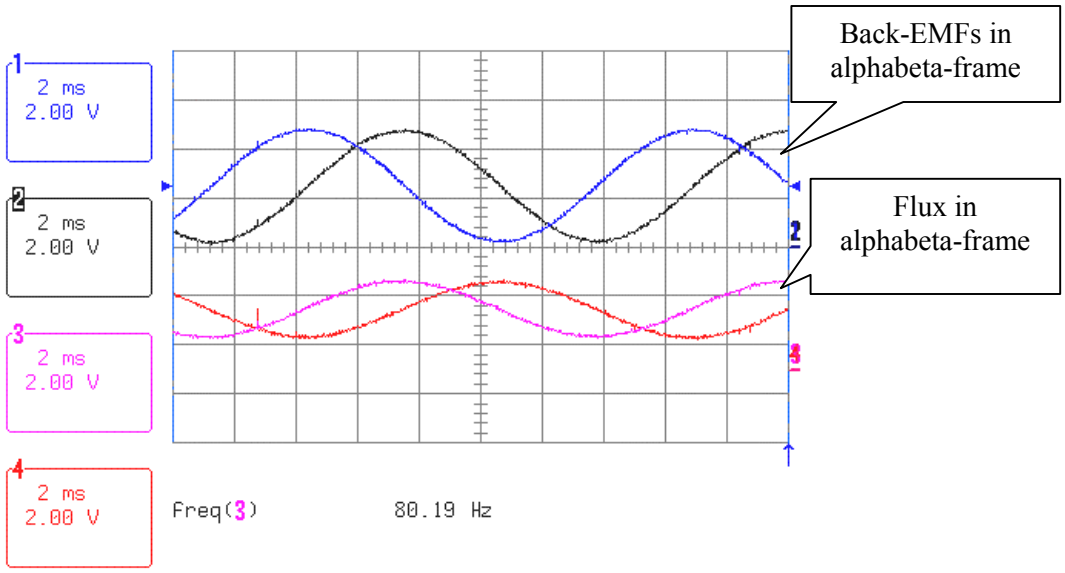


**Figure 8 - Back-EMFs and Fluxes**

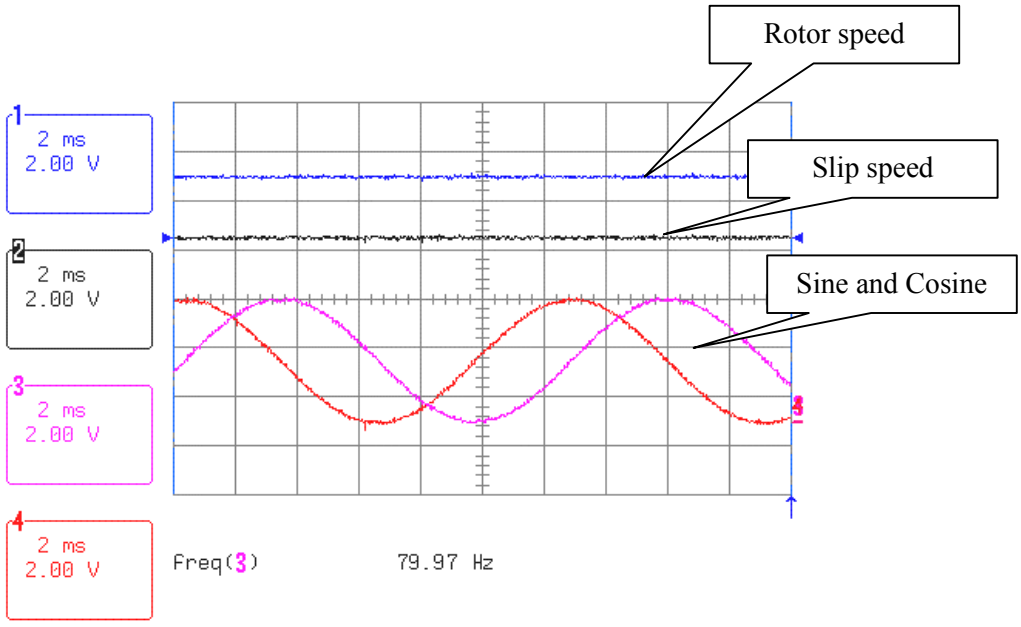


**Figure 9 - Speeds and control angles**

# 5 Reference

[Ref1] "A Tutorial in AC Induction and Permanent Magnet Synchronous Motors" - Analog Devices Inc. – Fred Flett.

[Ref2] " Power Electronic Control of AC Motors" - JMD Murphy & FG Turnbull – Pergamon Press '88.

[Ref3] "Electric Motors and Drives" – Austin Huges – Newnes '93.

[Ref4] "Implementation of Direct Stator Flux Orientation Control on a Versatile DSP Based System. Xingyi Xu and D.W. Novotny. Department of Electrical and Computer Engineering, University of Wisconsin-Madison. IEEE '90.

[Ref5] "New Integration Algorithms for Estimation Motor Flux Over a Wide Speed Range". J. Hu and B. Wu. Department of Electrical and Computer Engineering Ryerson Polytechnic University, Toronto, Ontario, Canada. IEEE '97.