Resolver-to-digital conversion with the ADMC401

AN401-22

© Analog Devices Inc., January 2000

Table of Contents

Summary

The application note describes how to obtain position information from a resolver using the simultaneous sampling capability of the Analogue-to-Digital converter of the ADMC401. The underlying theory will be treated as well as a recommended hardware scheme. At last, a software example will be described that may be downloaded from the accompanying zipped files. This example calculates the position information from the sampled resolver signals.

1 Introduction

The resolver is a transformer with one rotatable primary winding mounted on the rotor of the drive and two stationary secondary windings (Figure 1). The transformation ratios are modulated by the sine and cosine of the angle θ of the rotor.

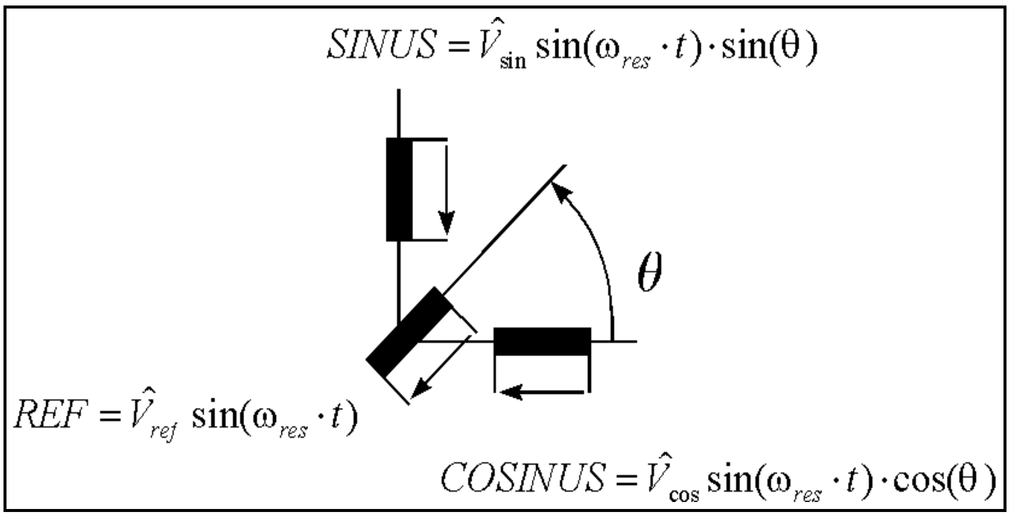

Figure 1: Resolver as rotatable transformer

The following equations are applied for the further discussion of the resolver evaluation principle:

$$REF = \hat{V}_{ref} \sin(\omega_{res} \cdot t) \quad (1)$$

$$SINUS = \hat{V}_{\sin} \sin(\omega_{res} \cdot t) \cdot \sin(\theta) \quad (2)$$

$$COSINUS = \hat{V}_{\cos} \sin(\omega_{res} \cdot t) \cdot \cos(\theta) \quad (3)$$

Here, θ is the rotor angle and $f_{res}=\omega_{res}/2\pi$ is the frequency of the resolver excitation signal.

These signals are shown in Figure 2.

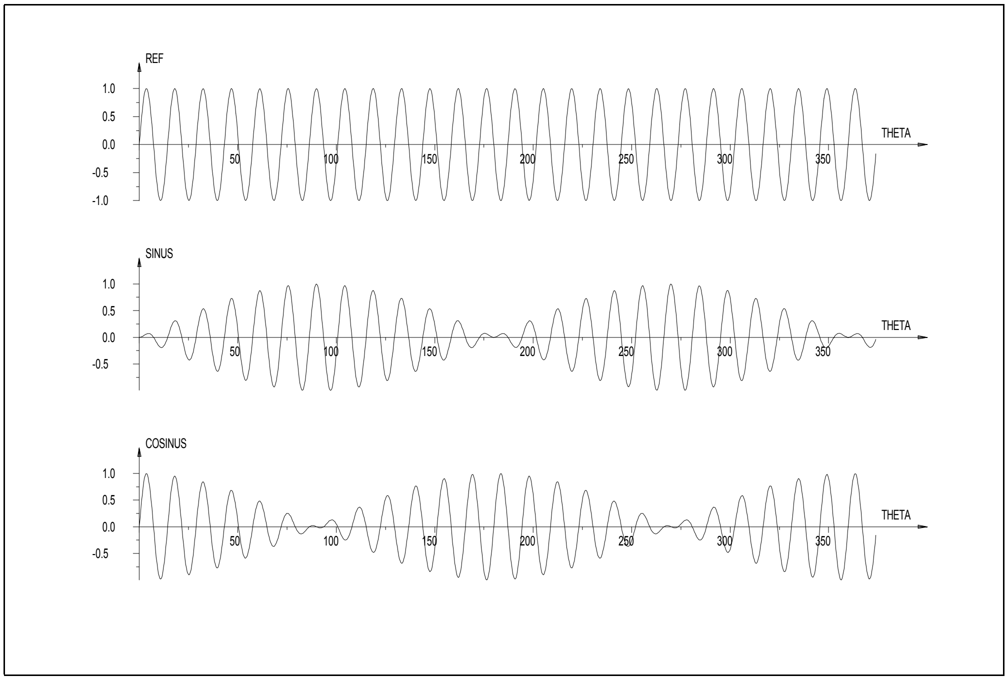

Figure 2 Typical resolver signals

2 Operation principle with sampling A/D-converter

With the A/D-converter of the ADMC 401 time discrete sets of samples from the SINUS and COSINUS signal traces are taken. In order to minimise the signal distortion the samples have to be taken repetitively with f_{res} at the maximum amplitude of each period. These conditions are shown in Figure 3.

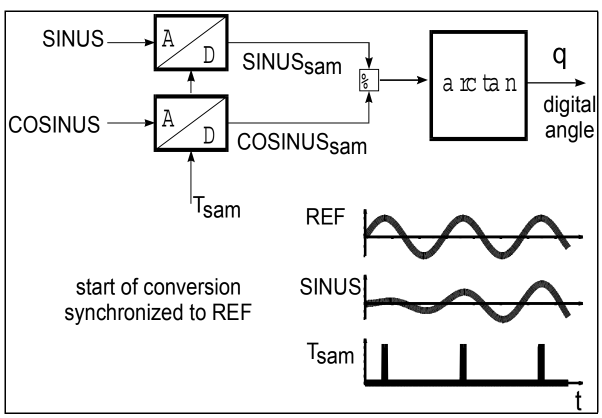

Figure 3 Signal detection with sampling A/D-converter

Thus the digital signal samples become:

$$SINUS_{sam} = \hat{V}_{\sin} \sin(\theta) \tag{4}$$

$$COSINUS_{sam} = \hat{V}_{\cos} \cos(\theta) \tag{5}$$

From this signal samples the rotor angle θ can be easily calculated according to

$$\frac{SINUS_{sam}}{COSINUS_{sam}} = \frac{\hat{V}_{\sin}\sin(\theta)}{\hat{V}_{\cos}\cos(\theta)} = \frac{\hat{V}_{\sin}}{\hat{V}_{\cos}}\tan(\theta) \qquad (6)$$

For an ideal resolver the amplitude of V_{sin} and V_{cos} will be equal. Thus the rotor angle is calculated from:

$$\theta = \arctan\left(\frac{SINUS_{sam}}{COSINUS_{sam}}\right) \qquad (7)$$

as depicted in Figure 3.

3 Resolution of the resolver-to-digital conversion and signal filtering

The quantisation error of the A/D-conversion with the ADMC 401 is 2^{-12}. This error is determined by an error of one LSB at 12 bit resolution. Errors caused by offsets or linearity are neglected for the following analysis. Because of the non-linearity of the signal processing according to Figure 3 the quantisation error is depending on the rotor position. For a simplified calculation the total error per revolution amounts to $2\cdot 2^{-12}$. This estimation takes into account the quantisation error of each sampled and digitised signals $COSINUS_{SAM}$ and $SINUS_{SAM}$ and further the error amplification caused by the division and the ARCTAN-function. It is assumed, that the ARCTAN function is evaluated only for the range from 0..45°. Without this assumption the quantisation error would increase substantially. The quantisation error is now related to the total angle range of 2π per revolution. This results in a relative position error of

$$ERROR_{res} = \frac{2\cdot 2^{-12}}{2\cdot\pi} \approx 2^{-13.65} \qquad (8)$$

This is an equivalent resolution of 13,65 bit for the position detection or an estimated resolution of 12854/revolution.

Thus normalised to 360 degree per revolution the position error caused by the A/D-quantisation and by the numeric signal processing principle amounts to:

$$ERROR_{\deg} = \pm\frac{1}{2}\cdot 2^{-13.65}\cdot 360\ \deg = \pm 0{,}014\ \deg \qquad (9)$$

The following simulation example demonstrates the distribution of the quantisation error for a rotor running at a constant speed of 500rpm. The upper curve shows the rotor position ramp. Further the diagram depicts the signals $COSINUS_{SAM}$ and $SINUS_{SAM}$, digitised with 12 bit and at last the position error caused normalised to degree. The above-calculated limits of ± 0,014 degree are indicated by the dotted lines. The result shows that the given formulas are sufficient to estimate the theoretical resolution limit.

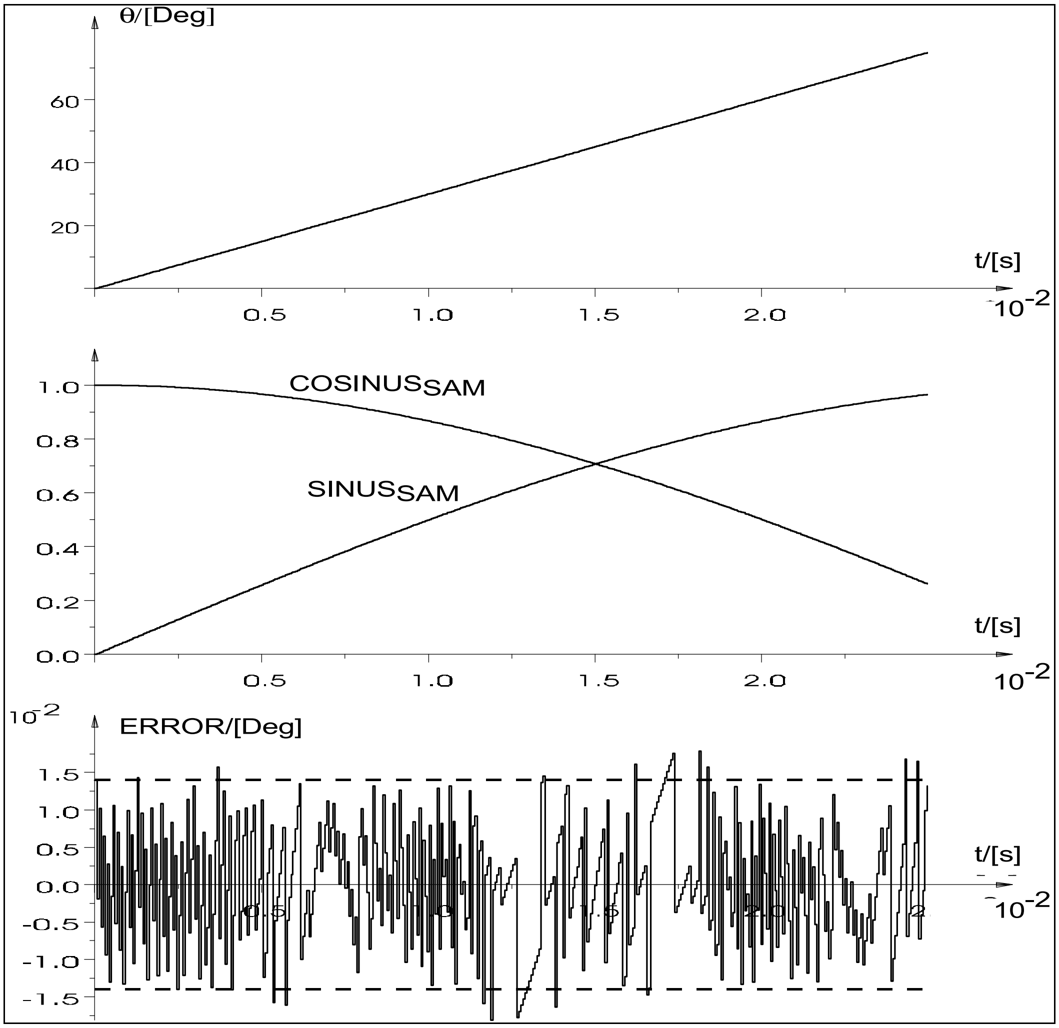

Figure 4: Position error caused by the signal quantisation and numeric effects

Servo systems typically apply a PWM-switching frequency of 16 kHz and a PWM time interval of $T_{pwm} = 62{,}5\ \mu s$. This period is also applicable for the calculation of the torque control of the drive including current commutation and current control. For this purpose the sampling period T_{sam} of the resolver output signals and the resolver excitation frequency f_{res} has to be set in an appropriate manner.

In servo systems the rotor speed is calculated by the application of a time discrete differentiation of the rotor angle similar to

$$SPEED = \frac{\theta(\nu) - \theta(\nu - 1)}{T_{con}} \tag{10}$$

Where $\nu \equiv \nu \cdot T_{con}$ indicates the sequence of the control periods with the interval time T_{con}.

The resolution of the speed signal is calculated according to

$$\Delta SPEED = \frac{1}{2^{13,65}\, T_{con}} \tag{11}$$

Some results are given in the following table:

ΔSPEED	T_{con}
75 rpm	62,5 µs
38 rpm	125 µs
19 rpm	250 µs
10 rpm	500 µs

Table 1: Discrete speed resolution

Thus the time interval T_{con} for the speed control loop must be selected with care. It is recommended to set the speed control time interval to a multiple of T_{pwm}. E.g. $T_{con} = 4..8 \cdot T_{pwm}$ depending on the application requirements. Furthermore a combination with a low pass filter type is useful for a further improvement of the speed signal quality.

4 Implementation of a 2nd order tracking filter

Compared to the ARCTAN calculation the implementation of a 2nd order tracking filter according to Figure 5 shows an improved performance, because the low pass filter is an integral part of the evaluation scheme and parasitic numeric effects are reduced.

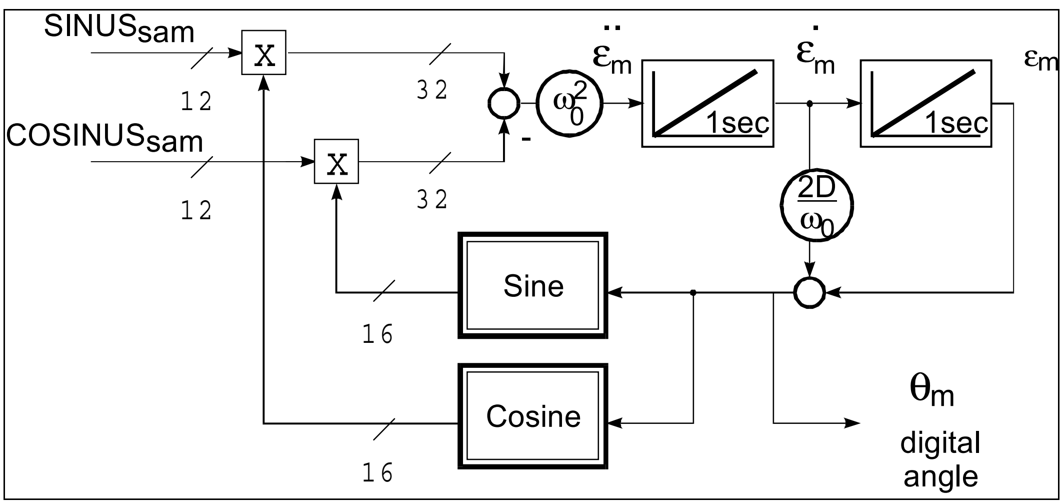

Figure 5 Block diagram of a tracking filter for the resolver evaluation

As shown in Figure 5, the tracking filter applies a demodulation scheme. For small deviations ($\theta \approx \theta_m$) the filter is characterised by the following transfer function:

$$F_m(s) = \frac{\theta_m}{\theta} = \frac{2D\frac{s}{\omega_0}+1}{\frac{s^2}{\omega_0^2}+2D\frac{s}{\omega_0}+1} \qquad (12)$$

The ratings for ω_0 and the damping factor D are selected according to the dynamic requirements of the speed and the position control loop of the servo drive. With a small rate ω_0 the smoothing of the position and of the speed signal is improved. The parameters may be adapted dynamically to improve the drive behaviour under various operating conditions.

The time discrete program function for the tracking filter shall be calculated with the same control frequency that is used for the current control and the commutation. But as recommended the speed signal and the speed control loop is calculated with an expanded time period T_{con} (Ref. Table 1).

5 Tracking error

The applied transfer function has zero tracking error at constant speed operation of the drive. At acceleration with a constant rate of $\ddot{\theta}_{const}$ the following tracking error occurs:

$$\Delta\theta = \theta - \theta_m = \frac{\ddot{\theta}_{const}}{\omega_0^2} \qquad (13)$$

The following simulation result depicts the behaviour for acceleration from standstill to a speed of 3000rpm within 10ms. For practical applications and a typical rating of ω_0 the tracking error can be neglected. For this example f_0 = 500 Hz and D = 0.7 were used.

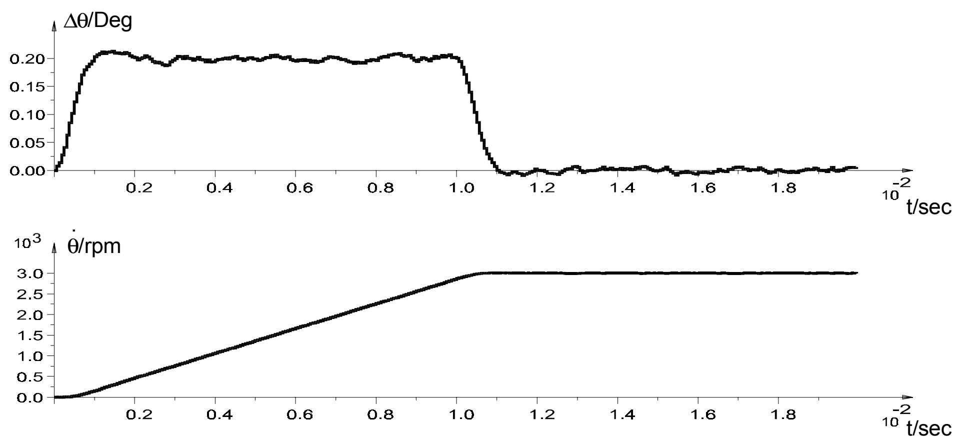

Figure 6 Tracking error during drive acceleration

6 External circuits

The ADMC 401 contains a fast, high accuracy, multiple-input analogue-to-digital conversion system with simultaneous sampling capabilities. The ADC system permits up to eight dedicated analogue inputs to be converted within 2µs.

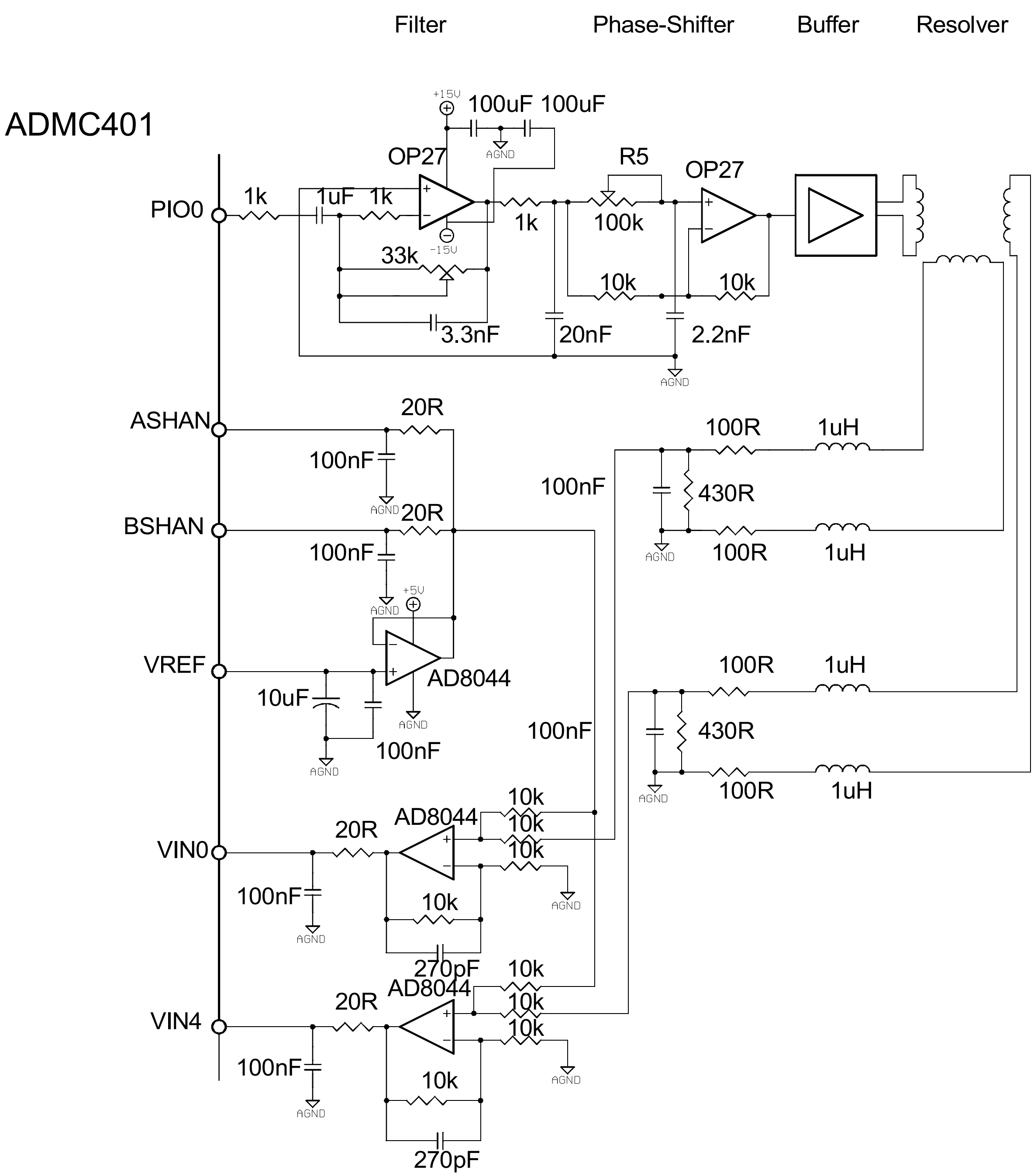

Figure 7: Resolver connection diagram

The resolver excitation hardware consists of one low pass filter for smoothing the square wave digital output and a phase shift filter to adjust signal delays caused by the resolver and the signal filter. The conversion of the SINUS and COSINUS resolver output is done synchronous to the resolver excitation with a constant phase shift. The ADC-input circuit consists of a filter for the suppression of noise caused by the PWM switching and a buffer to stabilise the VREF output of the ADMC 401.

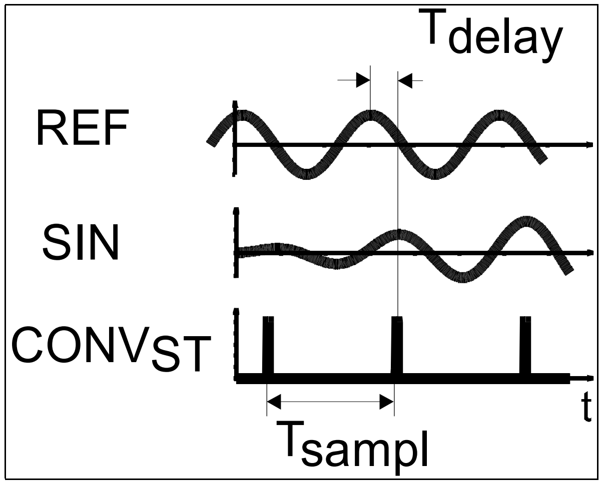

Figure 8 Resolver timing diagram

The SINUS and COSINUS signals from the resolver are sampled in the maximum of each period (Figure 7). For this purpose the phase shift (T_{delay}) has to be adjusted via R5 (Figure 7).

7 Sources of errors

Due to the noisy environment in drive applications the sampling of the resolver outputs is very sensitive. A shielding and twisted pair wiring is necessary to improve the signal to noise ratio. Especially the sampling method of the A/D-converter is a source of error because the noise of the power electronic switching superposes the resolver signal.

The signal to noise ratio may be improved by employing an integral measurement and sampling method, e.g. the application of an analogue integrator with synchronous clear or an AD-oversampling conversion method.

To achieve a good stability of the digital converted input signals an appropriate circuit-board layout and signal shielding is required.

Offsets resulting from operational amplifiers and reference voltage drift are minimised by the application of an automatic offset compensation procedure started at each power up of the system.

8 Automatic ADC offset adjust procedure

```
ADC_OFFSET:                        { start of module            }

  CALL READ_ADC1;                  {read value channel 0        }
  DM (SINU_AMPL) = AR;             {write back result           }

  CALL READ_ADC5;                  {read value channel 4        }
  DM (COSU_AMPL) = AR;             {write back result           }

                                   {                            }
  AX0 = DM(OFFSET_CNT);            { load counter value         }
  AR  = AX0 + 1;                   { increment counter          }
  DM(OFFSET_CNT) = AR;             { write back                 }
  AY0 = 0x7ffe;                    { counter = 32767 ?          }
  AR  = AX0 - AY0;                 {                            }
  IF EQ JUMP RESET_COUNTER;        { yes-> jump to label        }
  MR = 0;                          { no-> add the measured current }
  MR0 = DM(OFFSET1_LO);            { to the 32bit variable      }
  MR1 = DM(OFFSET1_HI);            {                            }
```

```
  MX0 = 0x1;                         { load ONE                      }
  MY0 = DM(SINU_AMPL);               { load sampled sine signal      }
  MR  = MR + MX0 * MY0 (SS);         { MAC  the offsets              }
  DM(OFFSET1_LO) = MR0;              { write to 32bit variable       }
  DM(OFFSET1_HI) = MR1;              {                               }
                                     {                               }
  MR = 0;                            { reset MR register             }
  MR0 = DM(OFFSET3_LO);              { do the same for ADC channel 5 }
  MR1 = DM(OFFSET3_HI);              {                               }
  MX0 = 0x1;                         {                               }
  MY0 = DM(COSU_AMPL);               { load sampled cosine signal    }
  MR  = MR + MX0 * MY0 (SS);         {                               }
  DM(OFFSET3_LO) = MR0;              {                               }
  DM(OFFSET3_HI) = MR1;              {                               }
  JUMP END_OFFSET;                   {                               }
                                     {                               }
RESET_COUNTER:                       {                               }
                                     {                               }
  AX0 = DM(OFFSET1_HI);              { store the value for offset    }
  DM(SINU_OFFSET) = AX0;             { offset sampled sine           }
  AX0 = DM(OFFSET3_HI);              { store the value for offset    }
  DM(COSU_OFFSET) = AX0;             { offset sampled cosine         }
                                     {                               }
  AX0 = 0;                           { reset counter                 }
  DM(OFFSET_CNT) = AX0;              { and the integrator variables  }
  DM(OFFSET1_HI) = AX0;              {                               }
  DM(OFFSET1_LO) = AX0;              {                               }
  DM(OFFSET3_HI) = AX0;              {                               }
  DM(OFFSET3_LO) = AX0;              {                               }
                                     {                               }
  mem_clear_bit (CONTROL,12);        { disable offset calibration    }
END_OFFSET:                          {                               }
                                     {                               }
  RTS;                               {                               }
```

9 Example tracking filter source code

```
{********************************************************************}
.CONST DOMEGA0          = 23170;  { Constant for 2D/w0                 }
.CONST OMEGA2           = 512;   { Constant for w0 * w0              }
{********************************************************************}

RESOLVER:


  Set_DAG_registers_for_trigonometric; {init L5, M5 for trigonometric functions}

  Sin(THETA);                     { calculate sine of THETA            }
  DM(SIN_THETA) = AR;             { write back result                  }
  MX0 = AR;
  AR = DM(COSU_AMPL);             { load amplitude RESOLVER COS val    }
  SR =  ASHIFT AR BY 1(hi);       { multiply by 2 to amplify signal    }
                                  { This shift is not needed, if the resolver}
                                  { inputs fit the 0..4V range of the ADMC401}
                                  { analog front end}
  MY1 = SR1;
  MR  = MX0*MY1 (SS);             { THETACOS32=sin(THETA)*COSU_AMPL    }
  DM(THETACOS32L) = MR0;          { store value THETACOS32 low         }
  DM(THETACOS32H) = MR1;          { store value THETACOS32 high        }

  Set_DAG_registers_for_trigonometric; {init L5, M5 for trigonometric functions}
  Cos(THETA);                     { calculate cosine of THETA          }
  DM(COS_THETA) = AR;             { write back result                  }
  MX0 = AR;
  AR = DM(SINU_AMPL);             { load amplitude RESOLVER SIN val    }
  SR =  ASHIFT AR BY 1(hi);       { multiply by 2 to amplify signal    }
                                  { This shift is not needed, if the resolver}
                                  { inputs fit the 0..4V range of the ADMC401}
                                  { analog front end}
  MY1 = SR1;
```

```
    MR  = MX0*MY1 (SS);              { THETASIN32=sin(THEAT)*SINU_AMPL    }
    DM(THETASIN32L) = MR0;           { store value THETASIN32 low         }
    DM(THETASIN32H) = MR1;           { store value THETASIN32 high        }

    AX0 = DM(THETASIN32L);           {get THETASIN32L                     }
    AX1 = DM(THETASIN32H);           {get THETASIN32H                     }
    AY0 = DM(THETACOS32L);           {get THETACOS32L                     }
    AY1 = DM(THETACOS32H);           {get THETACOS32H                     }
    AR = AX0-AY0;                    {SUB LSWs                            }
    AR=AX1-AY1+C-1;                  {SUB MSWs 32 Bit                     }
    MX1 = AR;                        {MX1 = THETACOS-THETASIN(High)       }

    MY0 = OMEGA2;                    {get OMEGA2 const 16 Bit             }
    MR = MX1*MY0(SU);                {Compute MSW                         }
    IF MV SAT MR;
    DM(EPSM__) = MR1;                {Store to EPSM__ (input first integrator)}
                                     {calculate first integrator 32 bit   }
    AY0 = DM(EPSM_INTL_);            {get EPSM_INT_ low word              }
    AY1 = DM(EPSM_INTH_);            {get EPSM_INT_  high word            }
    AX0 = MR0;                       {integrator input low word           }
    AX1 = DM(EPSM__);                {get 16 Bit EPSM__                   }
    AR = AX0+AY0;                    {ADD LSWs                            }
    SR0 = AR, AR=AX1+AY1+C;          {ADD MSWs 32 Bit                     }
    SR1 = AR;                        {SR=EPS_INT_+EPS__                   }
    DM(EPSM_INTL_) = SR0;            {store EPSM_INT_ low word            }
    DM(EPSM_INTH_) = SR1;            {store EPSM_INT_ high word           }
    DM(EPSM_) = SR1;                 {Store integrator result to EPSM_    }
                                     {calculate second integrator 32 bit  }
    AY0 = DM(EPSM_INTL);             {get EPSM_INT low word               }
    AY1 = DM(EPSM_INTH);             {get EPSM_INT   high word            }
    AX0 = SR0;                       {integrator input low word           }
    AX1 = DM(EPSM_);                 {get 16 Bit EPSM__                   }
    AR = AX0+AY0;                    {ADD LSWs                            }
    SR0 = AR, AR=AX1+AY1+C;          {ADD MSWs 32 Bit                     }
    SR1 = AR;                        {SR=EPS_INT+EPS_                     }
    DM(EPSM_INTL) = SR0;             {store EPSM_INT low word             }
    DM(EPSM_INTH) = SR1;             {store EPSM_INT high word            }
    DM(EPSM) = SR1;                  {Store integrator result to EPSM     }

    MR1 = SR1;                       { get EPSM (32 bit)for add multiply  }
    MR0 = SR0;                       { get EPSM (32 bit)for add multiply  }
    MX0 = DOMEGA0;                   { get const DOMEGA0                  }
    MY1 = DM(EPSM_);                 { get EPSM_                          }
    AR = MY1;                        { multiply by 16 to amplify signal   }
    SR = ASHIFT AR by 4(hi);
    MY1 = SR1;
    MR  = MR + MX0*MY1 (US);         { THETA=EPSM+EPSM_*DOMEGA0           }
    DM(THETA) = MR1;                 { store value THETA                  }


    RTS;
```

10 Calculation of digital values for D and ω0

The calculation of the digital values for D and ω_0 is based on Figure 9 for small deviations.

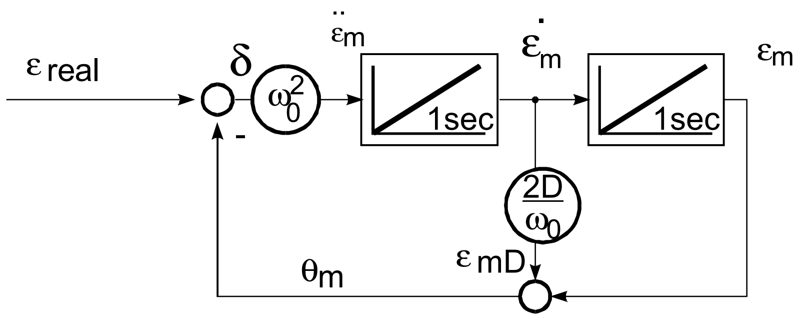

Figure 9: Tracking filter for small deviations

The digital value EPSM for the rotor angle ε is calculated as follows:

$$EPSM = \varepsilon_m \frac{2^{15}}{\pi} \tag{14}$$

This results in an overflow of the position at a full revolution when using a 16-bit variable for the rotor position. A digital value for the rotor frequency is calculated as follows: Assuming an integrator time constant of 1s means that at rotor speed of 1Hz a full rotor revolution is completed after 1s.

$$1Hz = 2\pi \frac{rad}{s} \Rightarrow \varepsilon_m(1s) = 2\pi \cdot rad \tag{15}$$

In the software, the integrator is realised as an accumulator that, at every T_{sample}, adds a discrete value to the integrator value. It is useful to normalise the position, so that every full revolution an integrator overflow occurs. If, e.g., the rotor position is stored as a 16-bit variable, this yields $2\pi = 2^{16}$.

Assuming a sample frequency of 8kHz and the same parameters as above mentioned, a digital input value of $2^{16}/T_{Sample}$= 8.192 results in an overflow at 1sec. The value of 8.192 represents 1Hz.

The following example shows how the digital values for ω_0^2 (OMEGA2) and $2D/\omega_0$ (DOMEGA0) are calculated. The user has to select the values for the constants OMEGA2 and DOMEGA0 dependent on the desired frequency ω_0 and damping factor D (see code sample chapter 9). The sample frequency is determined by the constant PWMfreq in the module constant.h. The interrupts are generated with the frequency PWMfreq. It has to be considered, that the tracking filter is calculated only every second interrupt.

The second integrator determines the position as follows:

$$\varepsilon_m = \varepsilon_m + \Delta t \dot{\varepsilon}_m = \varepsilon_m + \Delta\varepsilon_m, \dot{\varepsilon}_m = \omega_m \tag{16}$$

The position increment, which is added at each calculation of the resolver routine, is:

$$\Delta\varepsilon_m = \Delta t \omega_m \tag{17}$$

The digital representation is:

$$\Delta EPSM = EPSM\_ \cdot K\_ , with \quad K\_ = 1 \tag{18}$$

This results in

$$EPSM\_ = \omega_m \frac{2^{15}}{\pi} T_{Samp} \tag{19}$$

The tracking error is calculated as follows:

$$\varepsilon_{mD} = \frac{2D}{\omega_0} \omega_m \tag{20}$$

with digital values:

$$EPS\_MD = 2 \cdot DOMEGA0 \cdot EPSM\_ \cdot 16 \tag{21}$$

The digital value for EPSM_ is shifted by 4 before calculating the tracking error. This results in a factor 16. At each multiplication the ADMC performs an implicit left shift by 1 (multiply by 2). Only the high word of the MAC-operation is taken for the angle THETA. Thus the digital representation DOMEGA0 for $2D/\omega_0$ is:

$$DOMEGA0 = \frac{D}{\omega_0} \frac{1}{T_{Samp}} \frac{1}{16} \cdot 2^{16} \tag{22}$$

or, equivalent to this equation:

$$D = \frac{DOMEGA0}{2^{16}} \cdot 16 \cdot T_{Samp} \omega_0 \tag{23}$$

The rotor acceleration is calculated according to the following equation:

$$\ddot{\varepsilon}_m = \delta \omega_0^2 \tag{24}$$

δ is an angle and normalised in the same manner as ε_m:

$$DELTA = \delta \frac{2^{15}}{\pi} \tag{25}$$

Each time interval a speed increment $\Delta\omega_m$ is added to the rotor speed:

$$\Delta\omega_m = \ddot{\varepsilon}_m \Delta t \tag{26}$$

The digital speed increment is calculated as follows:

$$\Delta EPSM\_ = EPSM\_\_ \cdot K\_\_ \quad , with \quad K\_\_ = 1 \tag{27}$$

$$\Delta EPSM\_ = \Delta\omega_m \Delta t \cdot \frac{2^{15}}{\pi} \tag{28}$$

$$EPSM\_\_ = \ddot{\varepsilon}_m \Delta t \cdot \Delta t \cdot \frac{2^{15}}{\pi} \tag{29}$$

$$EPSM\_\_ = \ddot{\varepsilon}_m T_{Samp}^2 \frac{2^{15}}{\pi} \tag{30}$$

The rotor acceleration is calculated by multiplying the position difference and the digital value for ω_0^2. Again, the multiplication result is implicit shifted by 1.

$$EPSM\_\_ = 2 \cdot DELTA \cdot OMEGA2 \tag{31}$$

Comparing the last two equations leads to an expression for OMEGA2:

$$OMEGA2 = \frac{1}{2}\omega_0^2 \cdot T_{Samp}^2 2^{16} \tag{32}$$

Or, equivalently, to this equation:

$$\omega_0 = \frac{\sqrt{2 \cdot OMEGA2}}{T_{Samp} 2^8} \tag{33}$$

With OMEGA2 = 512 and DOMEGA0 = 23170 (see code example chapter 9) this results in D = 0.71 and ω_0 = 1000 Hz.

11 Signal characteristics

To determine the error of the resolver position evaluation, the angle THETA of the resolver filter was compared to the position measured by an additional encoder. The position difference is shown in

Figure 10. The motor was speeded to 3000rpm and then reverted to -3000rpm. This speed reversion is shown. The maximum difference between the position calculated by the resolver filter and the encoder during the transient amounts to 2.5°.

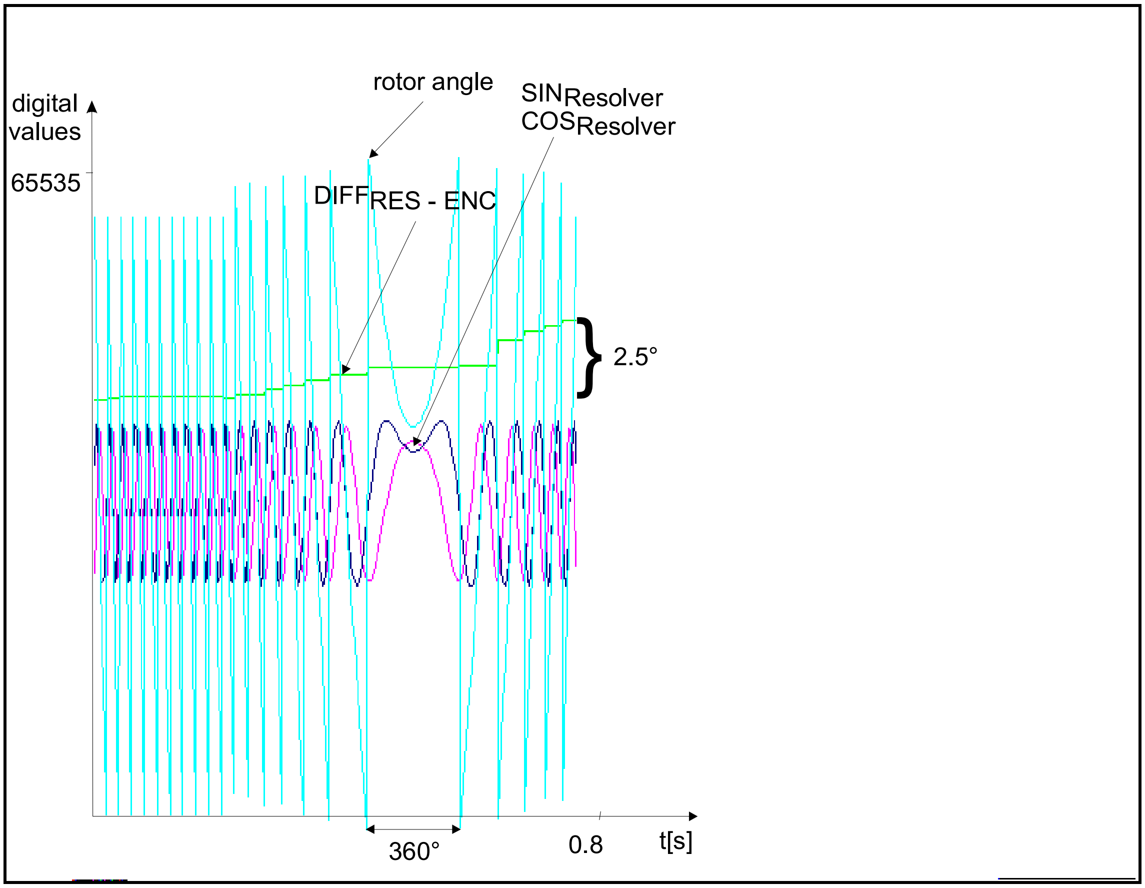

Figure 10: Measurement of the resolver position error