Advanced Control Library

User Reference Manual

56800E, 56800Ex
Digital Signal Controller

56800Ex_ACLIB
Rev. 0
02/2014

freescale.com

freescale™

Chapter 1 License Agreement

Chapter 2 INTRODUCTION

2.1 Overview

This reference manual describes the Advanced Control Library (ACLIB) for the Freescale 56F800E(X) family of Digital Signal Controllers. This library contains optimized functions.

2.2 Supported Compilers

Advanced Control Library (ACLIB) is written in assembly language with C-callable interface. The library was built and tested using the CodeWarrior™ Development Studio version 10.3.

The library is delivered in library module 56800Ex_ACLIB.lib and is intended for use in small data memory model projects. The interfaces to the algorithms included in this library have been combined into a single public interface include file, gdflib.h. This is done to simplify the number of files required for inclusion by application programs. Refer to the specific algorithm sections of this document for details on the software Application Programming Interface (API), defined and functionality provided for the individual algorithms.

2.3 Installation

If user wants to fully use this library, the CodeWarrior Development Studio should be installed prior to the Advanced Control Library. In case Advanced Control Library is installed while CodeWarrior Development Studio is not present, users can only browse the installed software package, but will not be able to build, download and run code. The installation itself consists of copying the required files to the destination hard drive, checking the presence of CodeWarrior and creating the shortcut under the Start->Programs menu.

The Advanced Control Library release is installed in its own folder named 56800Ex_ACLIB.

Perform the following steps to start the installation process:

1. Execute 56800Ex_FSLESL_rXX.exe.
2. Follow the FSLESL software installation instructions on your screen.

2.4 Library Integration

The library integration is described in AN4586 which can be downloaded from www.freescale.com.

2.5 API Definition

The description of each function described in this Advanced Control Library

user reference manual consists of following subsections:

Synopsis

This subsection gives the header files that should be included within a source file that references the function or macro. It also shows an appropriate declaration for the function that can be substituted by a macro. This declaration is not included in the program; only the header files should be included.

Prototype

This subsection shows the original function prototype declaration with all its arguments.

Arguments

This optional subsection describes input arguments to a function or macro.

Description

This subsection provides the description of functions or macros. It explains the algorithms used by functions or macros.

Return

This optional subsection describes the return value (if any) of function or macro.

Range Issues

This optional subsection specifies the ranges of input variables.

Special Issues

This optional subsection specifies special assumptions that are mandatory for correct function calculation; for example saturation, rounding, and so on.

Implementation

This optional subsection specifies, whether a call of the function generates a library function call or a macro expansion. It also consists one or more examples for the use of function. The examples are often fragments of code (not completed programs) for illustration purposes.

See Also

This optional subsection provides a list of related functions or macros.

Performance

This section specifies the actual requirements of the function or macro in terms of required code memory, data memory, and number of clock cycles to execute.

2.6 Data Types

The 16-bit DSC core supports four types of two's-complement data formats:

- Signed integer
- Unsigned integer
- Signed fractional
- Unsigned fractional

Signed and unsigned integer data types are useful for general purpose computation; they are familiar with the microprocessor and microcontroller programmers. Fractional data types allow powerful numeric and digital-signal-processing algorithms to be implemented.

2.6.1 Signed Integer (SI)

This format is used for processing data as integers. In this format, the N-bit operand is represented using the N.0 format (N integer bits). The range of signed integer numbers is as follows:

$$-2^{[N-1]} \le SI \le [2^{[N-1]} - 1] \qquad \textbf{\textit{Eqn. 2-1}}$$

This data format is available for bytes, words, and longs. The most negative signed word that can be represented is –32,768 ($8000) and the most negative signed long word is –2,147,483,648 ($80000000).

The most positive signed word is 32,767 ($7FFF) and the most positive signed long word is 2,147,483,647 ($7FFFFFFF).

2.6.2 Unsigned Integer (UI)

The unsigned integer numbers are positive only and they have nearly twice the magnitude of a signed number of the same size. The range of unsigned integer numbers is as follows:

$$0 \le UI \le [2^{[N-1]} - 1] \qquad \textbf{\textit{Eqn. 2-2}}$$

The binary word is interpreted as having a binary point immediately to the right of the integer's least significant bit. This data format is available for bytes, words, and long words. The most positive 16-bit unsigned integer is 65,535 ($FFFF), and the most positive 32-bit unsigned integer is 4,294,967,295 ($FFFFFFFF). The smallest unsigned integer number is zero ($0000), regardless of size.

2.6.3 Signed Fractional (SF)

In this format, the N-bit operand is represented using 1.[N–1] format (one sign bit, N–1 fractional bits). The range of signed fractional numbers is as follows:

$$-1.0 \le SF \le 1.0 - 2^{-[N-1]} \qquad \textbf{Eqn. 2-3}$$

This data format is available for words and long words. For both word and long-word signed fractions, the most negative number that can be represented is –1.0; its internal representation is \$8000 (word) or \$80000000 (long word). The most positive word is \$7FFF ($1.0 - 2^{-15}$) and its most positive long word is \$7FFFFFFF ($1.0 - 2^{-31}$).

2.6.4 Unsigned Fractional (UF)

The unsigned fractional numbers can be positive only and they have nearly twice the magnitude of a signed number with the same number of bits. The range of signed integer numbers is as follows:

$$0.0 \le UF \le 2.0 - 2^{-[N-1]} \qquad \textbf{Eqn. 2-4}$$

The binary word is interpreted as having a binary point after the MSB. This data format is available for words and longs. The most positive 16-bit unsigned number is \$FFFF, or $\{1.0 + (1.0 - 2^{-[N-1]})\} = 1.99997$. The smallest unsigned fractional number is zero (\$0000).

2.7 User Common Types

Table 2-1. User-Defined Typedefs in *56800E_types.h*

Mnemonics	Size — bits	Description
Word8	8	To represent 8-bit signed variable/value.
UWord8	8	To represent 16-bit unsigned variable/value.
Word16	16	To represent 16-bit signed variable/value.
UWord16	16	To represent 16-bit unsigned variable/value.
Word32	32	To represent 32-bit signed variable/value.
UWord32	32	To represent 16-bit unsigned variable/value.
Int8	8	To represent 8-bit signed variable/value.
UInt8	8	To represent 16-bit unsigned variable/value.
Int16	16	To represent 16-bit signed variable/value.
UInt16	16	To represent 16-bit unsigned variable/value.
Int32	32	To represent 32-bit signed variable/value.

Table 2-1. User-Defined Typedefs in *56800E_types.h* (continued)

UInt32	32	To represent 16-bit unsigned variable/value.
Frac16	16	To represent 16-bit signed variable/value.
Frac32	32	To represent 32-bit signed variable/value.
NULL	constant	Represents NULL pointer.
bool	16	Boolean variable.
false	constant	Represents false value.
true	constant	Represents true value.
FRAC16()	macro	Transforms float value from <–1, 1) range into fractional representation <–32768, 32767>.
FRAC32()	macro	Transforms float value from <–1, 1) range into fractional representation <–2147483648, 2147483648>.

2.8 V2 and V3 Core Support

This library document is written to support both 56800E (V2) and 56800Ex (V3) cores. The V3 core offers new set of math instructions which can simplify and accelerate the algorithm runtime. Therefore, certain algorithms can have two prototypes.

It is recommended to use V3 algorithms, if the library is used in the 56800Ex core, because of the following reasons:

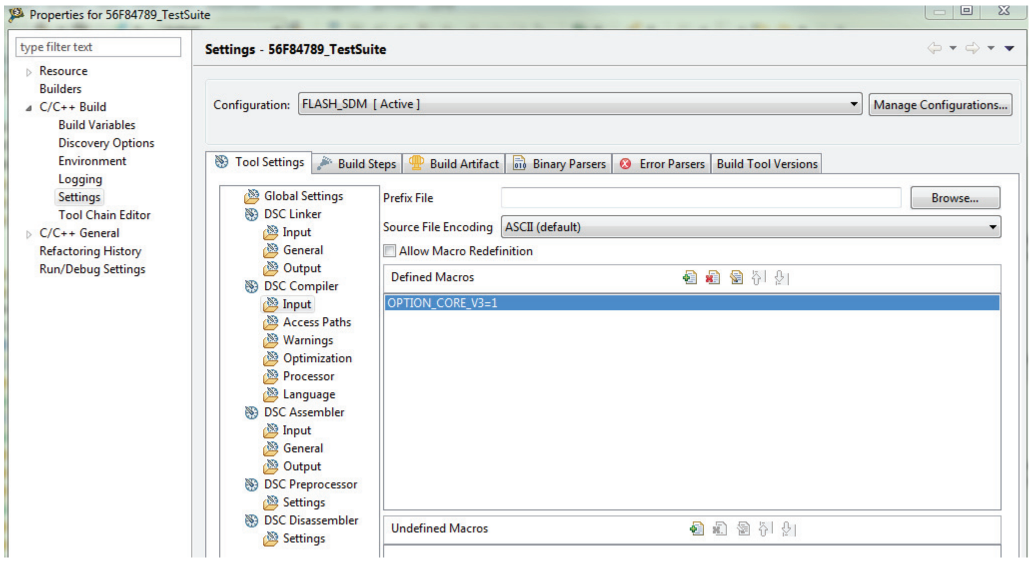
- the code is shorter
- the execution is faster
- the precision of 32-bit calculation is higher

To select the correct algorithm implementation the user has to set up a macro: OPTION_CORE_V3. If this macro is not defined, it is automatically set up as 0. If its value is 0, then V2 algorithms are used. If its value is 1, then V3 algorithms are used. The best way is to define this macro is in the project properties (see Figure 2-1). Use the following steps to define this macro:

1. In the left hand tree, expand the C/C++ Build node.
2. Click on the Settings node.
3. Under the Tool Settings tab, click on the DSC Compiler/Input node.
4. In the Defined Macros dialog box click on the first icon (+) and type the following macro:

   OPTION_CORE_V3=1.

5. Click OK.
6. Click OK on the Properties dialog box.


Properties for 56F84789_TestSuite

Settings - 56F84789_TestSuite

Configuration: FLASH_SDM [Active]

Prefix File

Source File Encoding ASCII (default)

Allow Macro Redefinition

Defined Macros

OPTION_CORE_V3=1

Undefined Macros

Figure 2-1. V2/V3 core option

2.9 Special Issues

All functions in the Advanced Control Library are implemented without storing any volatile registers (refer to the compiler manual) used by the respective routine. Only non-volatile registers (C10, D10, R5) are saved by pushing the registers on the stack. Therefore, if the particular registers initialized before the library function call are to be used after the function call, it is necessary to save them manually.

Chapter 3 FUNCTION API

3.1 API Summary

Table 3-1. API functions summary

Name	Arguments	Output	Description
ACLIB_PMSMBemfObsrvAB	*MCLIB_2_COOR_SYST_ALPHA_BETA_T *pudtIalbet MCLIB_2_COOR_SYST_ALPHA_BETA_T *pudtUalbet Frac16 f16Speed ACLIB_BEMF_OBSRV_AB_T * const pudtCtrl*	void	This function calculates the algorithms of finding permanent-magnet axis.
ACLIB_PMSMBemfObsrv12AB	*MCLIB_2_COOR_SYST_ALPHA_BETA_T *pudtIalbet MCLIB_2_COOR_SYST_ALPHA_BETA_T *pudtUalbet Frac16 f16Speed ACLIB_BEMF_OBSRV_AB_T * const pudtCtrl*	void	This function calculates the algorithms of finding permanent-magnet axis. This version uses the quicker 12-bit precision sine calculation therefore it is quicker but with reduced precision in comparison to **ACLIB_PMSMBemfObsrvAB**.
ACLIB_AngleTrackObsrv	*MCLIB_ANGLE_T *pudtSinCos ACLIB_ANGLE_TRACK_OBSRV_T * const pudtCtrl*	void	This function calculates the algorithm of velocity and position-tracking observer.
ACLIB_AngleTrackObsrv12	*MCLIB_ANGLE_T *pudtSinCos ACLIB_ANGLE_TRACK_OBSRV_T * const pudtCtrl*	void	This function calculates the algorithm of velocity and position-tracking observer. This version uses the quicker 12-bit precision sine calculation therefore it is quicker but with reduced precision in comparison to **ACLIB_AngleTrackObsrv**.
ACLIB_TrackObsrv	Frac16 f16ThetaErr ACLIB_TRACK_OBSRV_T * const pudtCtrl	void	This function calculates the tracking observer for determination angular speed and position of input error functional signal.
ACLIB_PMSMBemfObsrvDQ	MCLIB_2_COOR_SYST_D_Q_T *pudtIdq MCLIB_2_COOR_SYST_D_Q_T *pudtUdq,Frac16 f16Speed ACLIB_BEMF_OBSRV_DQ_T * const pudtCtrl	void	The function calculates the algorithm of back electromotive force observer in rotating reference frame.
ACLIB_IntegratorInitVal	Frac16 f16InitVal ACLIB_INTEGRATOR_T *pudtIntg	void	The function initializes the initial value of the **ACLIB_Integrator** algorithm.

Table 3-1. API functions summary

ACLIB_Integrator	Frac16 f16X ACLIB_INTEGRATOR_T *pudtIntg	void	The function calculates the algorithm of numerical integrator of its input.

3.2 ACLIB_PMSMBemfObsrvAB

The function calculates the algorithm of back electromotive force observer in stationary reference frame.

3.2.1 Synopsis

```
#include ”aclib.h”
void ACLIB_PMSMBemfObsrvAB(
MCLIB_2_COOR_SYST_ALPHA_BETA_T *pudtCurrentAlphaBeta,
MCLIB_2_COOR_SYST_ALPHA_BETA_T *pudtVoltageAlphaBeta,
Frac16 f16Speed,
ACLIB_BEMF_OBSRV_AB_T *pudtCtrl)
```

3.2.2 Prototype

```
asm void ACLIB_PMSMBemfObsrvABFAsm(MCLIB_2_COOR_SYST_ALPHA_BETA_T
*pudtCurrentAlphaBeta, MCLIB_2_COOR_SYST_ALPHA_BETA_T
*pudtVoltageAlphaBeta, Frac16 f16Speed, ACLIB_BEMF_OBSRV_AB_T *pudtCtrl)
```

V3 core version:

```
asm void ACLIB_V3PMSMBemfObsrvABFAsm(MCLIB_2_COOR_SYST_ALPHA_BETA_T
*pudtCurrentAlphaBeta, MCLIB_2_COOR_SYST_ALPHA_BETA_T
*pudtVoltageAlphaBeta, Frac16 f16Speed, ACLIB_BEMF_OBSRV_AB_T *pudtCtrl)
```

3.2.3 Arguments

Table 3-2. Function Arguments

Name	In/ Out	Format	Valid Range	Description
*pudtCurrentAlphaBeta	in	MCLIB_2_COOR_SYST_ALPHA_BETA_T	N/A	Input signal of alpha/beta current components.
*pudtVoltageAlphaBeta	in	MCLIB_2_COOR_SYST_ALPHA_BETA_T	N/A	Input signal of alpha/beta voltage components.
f16Speed	in	SF16	$8000... $7FFF	Fraction value of electrical speed.
*pudtCtrl	in/out	ACLIB_BEMF_OBSRV_AB_T	N/A	Pointer to an observer structure, which contains coefficients.

Table 3-3. User Types

Typedef	Name	Format	Valid Range	Description
ACLIB_BEMF_OBSRV_AB_T	udtEObsrv.f32Alpha	SF32	0x80000000... 0x7FFFFFFF	Estimated back-EMF voltage in beta axis.
	udtEObsrv.f32Beta	SF32	0x80000000... 0x7FFFFFFF	Estimated back-EMF voltage in beta axis.
	udtIObsrv.f32Alpha	SF32	0x80000000... 0x7FFFFFFF	Estimated current in alpha axis.
	udtIObsrv.f32Beta	SF32	0x80000000... 0x7FFFFFFF	Estimated current in beta axis.
	udtCtrl.f32IAlpha_1	SF32	0x80000000... 0x7FFFFFFF	State variable in alpha part of the observer; integral part at step k-1.
	udtCtrl.f32IBeta_1	SF32	0x80000000... 0x7FFFFFFF	State variable in beta part of the observer; integral part at step k-1.
	udtCtrl.f16PropGain	SF16	$8000... $7FFF	Observer proportional gain.
	udtCtrl.i16PropGainShift	SI16	-F...F	Observer proportional gain shift.
	udtCtrl.f16IntegGain	SF16	$8000... $7FFF	Observer integral gain.
	udtCtrl.i16IntegGainShift	SI16	-F...F	Observer integral gain shift.
	mcUnityVctr.f16Sin	MCLIB_ANGLE_T	$8000... $7FFF	Sine component of estimated unity vector.
	mcUnityVctr.f16Cos	MCLIB_ANGLE_T	$8000... $7FFF	Cosine component of estimated unity vector.
	f16IGain	SF16	$8000... $7FFF	Scaling coefficient for current I_{FRAC}.
	f16UGain	SF16	$8000... $7FFF	Scaling coefficient for voltage U_{FRAC}.
	f16WIGain	SF16	$8000... $7FFF	Scaling coefficient for angular speed WI_{FRAC}.
	f16EGain	SF16	$8000... $7FFF	Scaling coefficient for back-EMF E_{FRAC}.

3.2.4 Availability

This library module is available in the C-callable interface assembly formats.

This module is targeted for the 56800E and 56800Ex platforms.

3.2.5 Dependencies

List of all dependent files is as follows:

- 56800E_types.h
- 56800E_MCLIB library
- 56800E_GFLIB library
- ACLIB_AngleTrackObsrvAsm.h
- aclib.h

3.2.6 Description

This back-EMF observer is realized within stationary α, β reference frame.

$$\begin{bmatrix} u_\alpha \\ u_\beta \end{bmatrix} = R_S \begin{bmatrix} i_\alpha \\ i_\beta \end{bmatrix} + \begin{bmatrix} sL_D & \Delta L\omega_r \\ -\Delta L_D\omega_r & sL_D \end{bmatrix} \cdot \begin{bmatrix} i_\alpha \\ i_\beta \end{bmatrix} + (\Delta L \cdot (\omega_e i_D - i_Q') + k_e\omega_r) \cdot \begin{bmatrix} -\sin(\theta_r) \\ \cos(\theta_r) \end{bmatrix} \qquad \textbf{Eqn. 3-1}$$

where,

- R_s stator resistance
- L_d, L_q - D-axis and Q-axis inductance
- k_e back-EMF constant
- ω_e rotor angular speed
- u_α, u_β components of stator voltage vector
- i_α, i_β components of stator current vector
- s operator of derivative
- i_q' first derivative of i_q current
- $\Delta L = (L_D - L_Q)$ motor saliency

This extended back-EMF model includes both position information from the conventionally defined back-EMF and the stator inductance as well. This allows to extracts the rotor position and velocity information by estimating the extended back-EMF only.

Both alpha and beta axes consists of the stator current observer based on RL motor circuit which requires motor parameters.

The current observer input is the sum of actual applied motor voltage and cross-coupled rotational term, which corresponds to the motor saliency $(L_d - L_q)$ and compensator corrective output. The observer provides back-EMF signals as disturbance because back-EMF is not included in observer model.

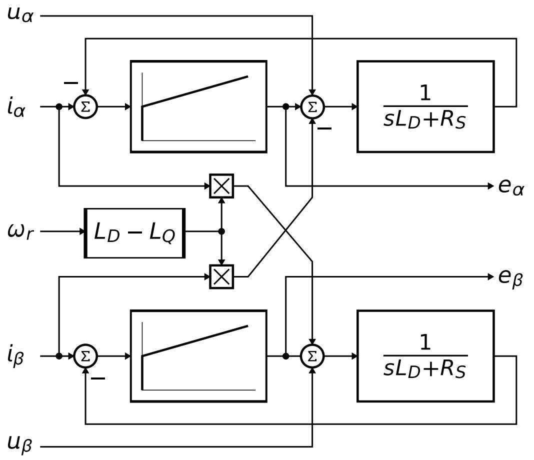


Figure 3-1. Block diagram of back-EMF observer

It is obvious that the accuracy of the back-EMF estimates is determined by the correctness of used motor parameters (R, L) by fidelity of the reference stator voltage and by quality of compensator such as bandwidth, phase lag, and so on.

Appropriate dynamic behavior of the back-EMF observer is achieved by placement of the poles of the stator current observer characteristic polynomial. This general method is based on matching the coefficients of the characteristic polynomial with the coefficients of the general second-order system.

$$\hat{E}_{\alpha\beta}(s) = -E_{\alpha\beta}(s) \cdot \left[\frac{F_c(s)}{sL_D + R_S + F_C(s)}\right] \qquad \text{Eqn. 3-2}$$

Back-EMF observer is Luenberger type observer with motor model which is realized in fixed point arithmetic transformed using backward Euler transformation.

$$\boldsymbol{i}_{FRFAC}(k) = U_{FRAC} \cdot \boldsymbol{u}_{FRAC}(k) + E_{FRAC} \cdot \boldsymbol{e}_{FRAC}(k) - WI_{FRAC} \cdot \omega_{eFRAC}(k)\boldsymbol{i'}_{FRAC}(k) + I_{FRAC} \cdot \boldsymbol{i}_{FRAC}(k-1) \qquad \text{Eqn. 3-3}$$

where,

- $\boldsymbol{i}_{FRFAC}(k) = [i_\alpha, i_\beta]$ is fractional representation of stator current vector
- $\boldsymbol{u}_{FRAC}(k) = [u_\alpha, u_\beta]$ is fractional representation of stator voltage vector
- $\boldsymbol{e}_{FRAC}(k) = [e_\alpha, e_\beta]$ is fractional representation of stator back-EMF voltage vector
- $\boldsymbol{i'}_{FRFAC}(k) = [i_\beta, -i_\alpha]$ is fractional representation of complementary stator current vector
- $\omega_{FRFAC}(k)$ is fractional representation of angular speed

Scaling coefficients relating to maximal values are expressed as follows:

$$U_{FRAC} = \frac{\Delta T_S}{L_d + \Delta T_S R_S} \cdot \frac{U_{MAX}}{I_{MAX}} \qquad \text{Eqn. 3-4}$$

$$E_{FRAC} = \frac{\Delta T_S}{L_d + \Delta T_S R_S} \cdot \frac{E_{MAX}}{I_{MAX}} \qquad \textit{Eqn. 3-5}$$

$$WI_{FRAC} = \frac{\Delta L \cdot \Delta T_S}{L_d + \Delta T_S R_S} \cdot \Omega_{MAX} \qquad \textit{Eqn. 3-6}$$

$$I_{FRAC} = \frac{L_d}{L_d + \Delta T_S R_S} \qquad \textit{Eqn. 3-7}$$

where,

- ΔT_S sampling time in [sec]
- I_{MAX} maximal peak current in [A]
- E_{MAX} maximal peak back-EMF voltage in [V]
- U_{MAX} maximal peak stator voltage in [V]
- Ω_{MAX} maximal angular speed in [rad/sec]

If a Luenberger type stator current observer is properly designed in the stationary reference frame, the back-EMF can be estimated as a disturbance, produced by the observer controller. This is valid only when the back-EMF term is not included in the observer model. The observer is actually a closed loop current observer so it acts as a state filter for the back-EMF term.

The estimate of extended EMF term can be derived from Equation 3-2 as follows:

$$-\frac{\hat{E}_{\alpha\beta}(s)}{E_{\alpha\beta}(s)} = \frac{sK_P + K_I}{s^2 L_D + sR_S + sK_P + K_I} \qquad \textit{Eqn. 3-8}$$

The observer controller can be designed by comparing the closed loop characteristic polynomial with that of a standard second order system as:

$$s^2 + \frac{K_P + R_S}{L_D} \cdot s + \frac{K_I}{L_D} = s^2 + 2\xi\omega_0 s + \omega_0^2 \qquad \textit{Eqn. 3-9}$$

where,

- ω_0 is the natural frequency of the closed loop system (loop bandwidth)
- ξ is the loop attenuation.

3.2.7 Returns

The function returns a unity vector representing the estimated value of sine and cosine values of back-EMF.

3.2.8 Implementation

Example 3-1. Implementation Code

```
#include "gflib.h"
#include "mclib.h"
#include "aclib.h"

MCLIB_2_COOR_SYST_ALPHA_BETA_T     mcI, mcU;
ACLIB_BEMF_OBSRV_AB_T              acBemfObsrv;

void Isr(void);

void main (void)
{
acBemfObsrv.udtEObsrv.f32Alpha      = FRAC32(0.0);
acBemfObsrv.udtEObsrv.f32Beta       = FRAC32(0.0);
acBemfObsrv.udtIObsrv.f32Alpha      = FRAC32(0.0);
acBemfObsrv.udtIObsrv.f32Beta       = FRAC32(0.0);
acBemfObsrv.udtCtrl.f32IAlpha_1     = FRAC32(0.0);
acBemfObsrv.udtCtrl.f32IBeta_1      = FRAC32(0.0);
acBemfObsrv.udtCtrl.f16PropGain     = BEMFOBSRV_AB_PROP_GAIN_SCALED;
acBemfObsrv.udtCtrl.i16PropGainShift = BEMFOBSRV_AB_PROP_GAIN_SHIFT;
acBemfObsrv.udtCtrl.f16IntegGain    = BEMFOBSRV_AB_INTEG_GAIN_SCALED;
acBemfObsrv.udtCtrl.i16IntegGainShift = BEMFOBSRV_AB_INTEG_GAIN_SHIFT;
acBemfObsrv.f16IGain                = BEMFOBSRV_AB_I_SCALED;
acBemfObsrv.f16UGain                = BEMFOBSRV_AB_U_SCALED;
acBemfObsrv.f16EGain                = BEMFOBSRV_AB_E_SCALED;
acBemfObsrv.f16WIGain               = BEMFOBSRV_AB_WI_SCALED;
}

/* Periodical function or interrupt */
void ISR(void)
{
ACLIB_PMSMBemfObsrvAB(&mcI,&mcU,f16Speed,&acBemfObsrv);
}
```

3.2.9 Performance

Table 3-4. Performance of ACLIB_PMSMBemfObsrvAB function

Code Size (words)	V2: 184 + 65, V3: 169 + 65 (GFLIB_SqrtPoly)	
Data Size (words)	0 + 34 (GFLIB_SqrtPoly)	
Execution Clock	Min	V2: 340, V3: 328 cycles
	Max	V2: 340, V3: 328 cycles

3.3 ACLIB_PMSMBemfObsrv12AB

The function calculates the algorithm of back electromotive force observer in stationary reference frame.This version uses the quicker 12-bit precision sine calculation; therefore, it is quicker but with reduced precision in comparison to **ACLIB_PMSMBemfObsrvAB**.

3.3.1 Synopsis

```
#include ”aclib.h”
void ACLIB_PMSMBemfObsrv12AB(
MCLIB_2_COOR_SYST_ALPHA_BETA_T *pudtCurrentAlphaBeta,
MCLIB_2_COOR_SYST_ALPHA_BETA_T *pudtVoltageAlphaBeta,
Frac16 f16Speed,
ACLIB_BEMF_OBSRV_AB_T *pudtCtrl)
```

3.3.2 Prototype

```
asm void ACLIB_PMSMBemfObsrv12ABFAsm(MCLIB_2_COOR_SYST_ALPHA_BETA_T
*pudtCurrentAlphaBeta, MCLIB_2_COOR_SYST_ALPHA_BETA_T
*pudtVoltageAlphaBeta, Frac16 f16Speed, ACLIB_BEMF_OBSRV_AB_T *pudtCtrl)
```

V3 core version:

```
asm void ACLIB_V3PMSMBemfObsrv12ABFAsm(MCLIB_2_COOR_SYST_ALPHA_BETA_T
*pudtCurrentAlphaBeta, MCLIB_2_COOR_SYST_ALPHA_BETA_T
*pudtVoltageAlphaBeta, Frac16 f16Speed, ACLIB_BEMF_OBSRV_AB_T *pudtCtrl)
```

3.3.3 Arguments

Table 3-5. Function Arguments

Name	In/ Out	Format	Valid Range	Description
*pudtCurrentAlphaBeta	in	MCLIB_2_COOR_SYST_ALPHA_BETA_T	N/A	Input signal of alpha/beta current components.
*pudtVoltageAlphaBeta	in	MCLIB_2_COOR_SYST_ALPHA_BETA_T	N/A	Input signal of alpha/beta voltage components.
f16Speed	in	SF16	$8000... $7FFF	Fraction value of electrical speed.
*pudtCtrl	in/out	ACLIB_BEMF_OBSRV_AB_T	N/A	Pointer to an observer structure, which contains coefficients.

Table 3-6. User Types

Typedef	Name	Format	Valid Range	Description
ACLIB_BEMF_OBSRV_AB_T	udtEObsrv.f32Alpha	SF32	0x80000000... 0x7FFFFFFF	Estimated back-EMF voltage in beta axis.
	udtEObsrv.f32Beta	SF32	0x80000000... 0x7FFFFFFF	Estimated back-EMF voltage in beta axis.
	udtIObsrv.f32Alpha	SF32	0x80000000... 0x7FFFFFFF	Estimated current in alpha axis.
	udtIObsrv.f32Beta	SF32	0x80000000... 0x7FFFFFFF	Estimated current in beta axis.
	udtCtrl.f32IAlpha_1	SF32	0x80000000... 0x7FFFFFFF	State variable in alpha part of the observer; integral part at step k-1.
	udtCtrl.f32IBeta_1	SF32	0x80000000... 0x7FFFFFFF	State variable in beta part of the observer; integral part at step k-1.
	udtCtrl.f16PropGain	SF16	$8000... $7FFF	Observer proportional gain.
	udtCtrl.i16PropGainShift	SI16	-F...F	Observer proportional gain shift.
	udtCtrl.f16IntegGain	SF16	$8000... $7FFF	Observer integral gain.
	udtCtrl.i16IntegGainShift	SI16	-F...F	Observer integral gain shift.
	mcUnityVctr.f16Sin	MCLIB_ANGLE_T	$8000... $7FFF	Sine component of estimated unity vector.
	mcUnityVctr.f16Cos	MCLIB_ANGLE_T	$8000... $7FFF	Cosine component of estimated unity vector.
	f16IGain	SF16	$8000... $7FFF	Scaling coefficient for current I_{FRAC}.
	f16UGain	SF16	$8000... $7FFF	Scaling coefficient for voltage U_{FRAC}.
	f16WIGain	SF16	$8000... $7FFF	Scaling coefficient for angular speed WI_{FRAC}.
	f16EGain	SF16	$8000... $7FFF	Scaling coefficient for back-EMF E_{FRAC}.

3.3.4 Availability

This library module is available in the C-callable interface assembly formats.

This module is targeted for the 56800E and 56800Ex platforms.

3.3.5 Dependencies

List of all dependent files is as follows:

- 56800E_types.h
- 56800E_MCLIB library
- 56800E_GFLIB library
- ACLIB_AngleTrackObsrvAsm.h
- aclib.h

3.3.6 Description

This back-EMF observer is realized within stationary α, β reference frame.

$$\begin{bmatrix} u_\alpha \\ u_\beta \end{bmatrix} = R_S \begin{bmatrix} i_\alpha \\ i_\beta \end{bmatrix} + \begin{bmatrix} sL_D & \Delta L\omega_r \\ -\Delta L_D\omega_r & sL_D \end{bmatrix} \cdot \begin{bmatrix} i_\alpha \\ i_\beta \end{bmatrix} + (\Delta L \cdot (\omega_e i_D - i_Q') + k_e\omega_r) \cdot \begin{bmatrix} -\sin(\theta_r) \\ \cos(\theta_r) \end{bmatrix} \qquad \textbf{\textit{Eqn. 3-10}}$$

where,

- R_s stator resistance
- L_d, L_q - D-axis and Q-axis inductance
- k_e back-EMF constant
- ω_e rotor angular speed
- u_α, u_β components of stator voltage vector
- i_α, i_β components of stator current vector
- s operator of derivative
- i_q' first derivative of i_q current
- $\Delta L = (L_D - L_Q)$ motor saliency

This extended back-EMF model includes both position information from the conventionally defined back-EMF and the stator inductance as well. This allows to extracts the rotor position and velocity information by estimating the extended back-EMF only.

Both alpha and beta axes consists of the stator current observer based on RL motor circuit which requires motor parameters.

The current observer input is the sum of actual applied motor voltage and cross-coupled rotational term, which corresponds to the motor saliency $(L_d - L_q)$ and compensator corrective output. The observer provides back-EMF signals as disturbance because back-EMF is not included in observer model.

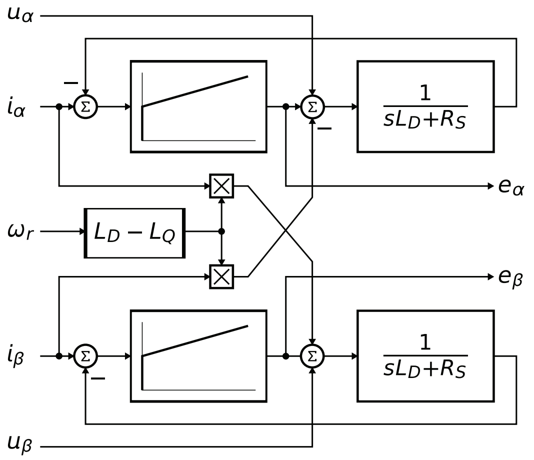


Figure 3-2. Block diagram of back-EMF observer

It is obvious that the accuracy of the back-EMF estimates is determined by the correctness of used motor parameters (R, L) by fidelity of the reference stator voltage and by quality of compensator such as bandwidth, phase lag, and so on.

Appropriate dynamic behavior of the back-EMF observer is achieved by placement of the poles of the stator current observer characteristic polynomial. This general method is based on matching the coefficients of the characteristic polynomial with the coefficients of the general second-order system.

$$\hat{E}_{\alpha\beta}(s) = -E_{\alpha\beta}(s)\cdot\left[\frac{F_c(s)}{sL_D + R_S + F_C(s)}\right] \qquad \textbf{\textit{Eqn. 3-11}}$$

Back-EMF observer is Luenberger type observer with motor model which is realized in fixed point arithmetic transformed using backward Euler transformation.

$$\boldsymbol{i}_{FRFAC}(k) = U_{FRAC}\cdot\boldsymbol{u}_{FRAC}(k) + E_{FRAC}\cdot\boldsymbol{e}_{FRAC}(k) - WI_{FRAC}\cdot\omega_{eFRAC}(k)\boldsymbol{i'}_{FRAC}(k) + I_{FRAC}\cdot\boldsymbol{i}_{FRAC}(k-1) \qquad \textbf{\textit{Eqn. 3-12}}$$

where,

- $\boldsymbol{i}_{FRFAC}(k) = [i_\alpha, i_\beta]$ is fractional representation of stator current vector
- $\boldsymbol{u}_{FRAC}(k) = [u_\alpha, u_\beta]$ is fractional representation of stator voltage vector
- $\boldsymbol{e}_{FRAC}(k) = [e_\alpha, e_\beta]$ is fractional representation of stator back-EMF voltage vector
- $\boldsymbol{i'}_{FRFAC}(k) = [i_\beta, -i_\alpha]$ is fractional representation of complementary stator current vector
- $\omega_{FRFAC}(k)$ is fractional representation of angular speed

Scaling coefficients relating to maximal values are expressed as follows:

$$U_{FRAC} = \frac{\Delta T_S}{L_d + \Delta T_S R_S}\cdot\frac{U_{MAX}}{I_{MAX}} \qquad \textbf{\textit{Eqn. 3-13}}$$

$$E_{FRAC} = \frac{\Delta T_S}{L_d + \Delta T_S R_S} \cdot \frac{E_{MAX}}{I_{MAX}}$$ **Eqn. 3-14**

$$WI_{FRAC} = \frac{\Delta L \cdot \Delta T_S}{L_d + \Delta T_S R_S} \cdot \Omega_{MAX}$$ **Eqn. 3-15**

$$I_{FRAC} = \frac{L_d}{L_d + \Delta T_S R_S}$$ **Eqn. 3-16**

where,

- ΔT_S sampling time in [sec]
- I_{MAX} maximal peak current in [A]
- E_{MAX} maximal peak back-EMF voltage in [V]
- U_{MAX} maximal peak stator voltage in [V]
- Ω_{MAX} maximal angular speed in [rad/sec]

If a Luenberger type stator current observer is properly designed in the stationary reference frame, the back-EMF can be estimated as a disturbance, produced by the observer controller. This is valid only when the back-EMF term is not included in the observer model. The observer is actually a closed loop current observer so it acts as a state filter for the back-EMF term.

The estimate of extended EMF term can be derived from Equation 3-11 as follows:

$$-\frac{\hat{E}_{\alpha\beta}(s)}{E_{\alpha\beta}(s)} = \frac{sK_P + K_I}{s^2 L_D + sR_S + sK_P + K_I}$$ *Eqn. 3-17*

The observer controller can be designed by comparing the closed loop characteristic polynomial with that of a standard second order system as:

$$s^2 + \frac{K_P + R_S}{L_D} \cdot s + \frac{K_I}{L_D} = s^2 + 2\xi\omega_0 s + \omega_0^2$$ *Eqn. 3-18*

where,

- ω_0 is the natural frequency of the closed loop system (loop bandwidth)
- ξ is the loop attenuation.

3.3.7 Returns

The function returns a unity vector representing the estimated value of sine and cosine values of back-EMF.

3.3.8 Implementation

Example 3-2. Implementation Code

```
#include "gflib.h"
#include "mclib.h"
#include "aclib.h"

MCLIB_2_COOR_SYST_ALPHA_BETA_T      mcI, mcU;
ACLIB_BEMF_OBSRV_AB_T               acBemfObsrv;

void Isr(void);

void main (void)
{
acBemfObsrv.udtEObsrv.f32Alpha      = FRAC32(0.0);
acBemfObsrv.udtEObsrv.f32Beta       = FRAC32(0.0);
acBemfObsrv.udtIObsrv.f32Alpha      = FRAC32(0.0);
acBemfObsrv.udtIObsrv.f32Beta       = FRAC32(0.0);
acBemfObsrv.udtCtrl.f32IAlpha_1     = FRAC32(0.0);
acBemfObsrv.udtCtrl.f32IBeta_1      = FRAC32(0.0);
acBemfObsrv.udtCtrl.f16PropGain     = BEMFOBSRV_AB_PROP_GAIN_SCALED;
acBemfObsrv.udtCtrl.i16PropGainShift = BEMFOBSRV_AB_PROP_GAIN_SHIFT;
acBemfObsrv.udtCtrl.f16IntegGain    = BEMFOBSRV_AB_INTEG_GAIN_SCALED;
acBemfObsrv.udtCtrl.i16IntegGainShift = BEMFOBSRV_AB_INTEG_GAIN_SHIFT;
acBemfObsrv.f16IGain                = BEMFOBSRV_AB_I_SCALED;
acBemfObsrv.f16UGain                = BEMFOBSRV_AB_U_SCALED;
acBemfObsrv.f16EGain                = BEMFOBSRV_AB_E_SCALED;
acBemfObsrv.f16WIGain               = BEMFOBSRV_AB_WI_SCALED;
}

/* Periodical function or interrupt */
void ISR(void)
{
ACLIB_PMSMBemfObsrv12AB(&mcI,&mcU,f16Speed,&acBemfObsrv);
}
```

3.3.9 Performance

Table 3-7. Performance of ACLIB_PMSMBemfObsrv12AB function

Code Size (words)	V2: 182 + 28, V3: 167 + 28 (GFLIB_SqrtIter)	
Data Size (words)	0	
Execution Clock	Min	V2: 312, V3: 296 cycles
	Max	V2: 312, V3: 296 cycles

3.4 ACLIB_AngleTrackObsrv

The function calculates angle tracking observer for determination angular speed and position of input functional signal.

3.4.1 Synopsis

```
#include"aclib.h"
Frac16 ACLIB_AngleTrackObsrv(MCLIB_ANGLE_T *pudtSinCos,
ACLIB_ANGLE_TRACK_OBSRV_T * pudtCtrl)
```

3.4.2 Prototype

```
asm Frac16 ACLIB_AngleTrackObsrvFAsm(MCLIB_ANGLE_T *pudtSinCos,
ACLIB_ANGLE_TRACK_OBSRV_T * const pudtCtrl)
```

V3 core version:

```
asm Frac16 ACLIB_V3AngleTrackObsrvFAsm(MCLIB_ANGLE_T *pudtSinCos,
ACLIB_ANGLE_TRACK_OBSRV_T * const pudtCtrl)
```

3.4.3 Arguments

Table 3-8. Function Arguments

Name	In/ Out	Format	Valid Range	Description
*pudtSinCos	in	MCLIB_ANGLE_T	N/A	input signal of sine, cosine components to be filtered
*pudtCtrl	in/out	ACLIB_ANGLE_TRACK_OBSRV_T	N/A	pointer to an angle tracking observer structure ACLIB_ANGLE_TRACK_OBSRV_T, which contains algorithm coefficients

Table 3-9. User type definitions

Typedef	Name	In/ Out	Format	Valid Range	Description
MCLIB_ANGLE_T	f16Sin	In	SF16	$8000... $7FFF	sine component to be estimated
	f16Cos	In	SF16	$8000... $7FFF	cosine component to be estimated
ACLIB_ANGLE_TRACK_OBSRV_T	f32Speed	in/out	SF32	0x80000000... 0x7FFFFFFF	Estimated speed as output of the first numerical integrator
	f32A2	in/out	SF32	0x80000000... 0x7FFFFFFF	Output of the second numerical integrator
	f16Theta	in/out	SF16	$8000... $7FFF	Estimated position
	f16SinEstim	in	SF16	$8000... $7FFF	Sine signal to be estimated
	f16CosEstim	in	SF16	$8000... $7FFF	Cosine signal to be estimated
	f16K1Gain	in	SF16	$8000... $7FFF	K1 coefficient scaled to fractional range
	i16K1GainShift	in	SI16	-F...F	Scaling shift
	f16K2Gain	in	SF16	$8000... $7FFF	K2 coefficient scaled to fractional range
	i16K2GainShift	in	SI16	-F...F	Scaling shift
	f16A2Gain	in	SF16	$8000... $7FFF	Scaling coefficient due to numerical integration
	i16A2GainShift	in	SI16	-F...F	Scaling shift

3.4.4 Availability

This library module is available in the C-callable interface assembly formats.

This module is targeted for the 56800E and 56800Ex platforms.

3.4.5 Dependencies

List of all dependent files is as follows:

- 56800E_types.h
- 56800E_MCLIB library
- 56800E_GFLIB library
- ACLIB_AngleTrackObsrvAsm.h
- aclib.h

3.4.6 Description

This function calculates the angle tracking observer algorithm. It is recommended to call this function at every sampling period. It requires two input arguments as sine and cosine samples. The practical implementation of the angle tracking observer algorithm is described below.

The angle tracking observer compares values of the input signals $\sin(\theta)$, $\cos(\theta)$ with their corresponding estimations $\sin(\hat{\theta})$, $\cos(\hat{\theta})$. As in any common closed-loop systems, the intent is to minimize observer error towards zero value. The observer error is given here by subtraction of the estimated resolver rotor angle $\hat{\theta}$ from the actual rotor angle θ (see **Figure 3-3**).

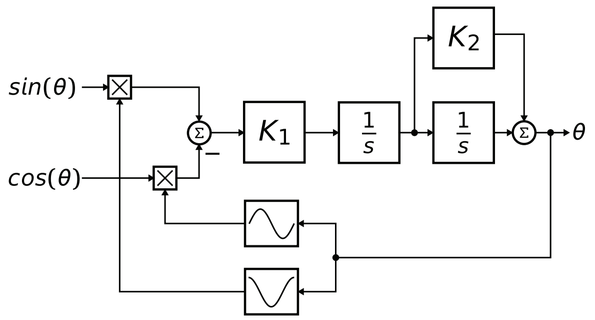


Figure 3-3. Block scheme of the angle tracking observer

Note that mathematical expression of observer error is known as the formula of the difference of two angles:

$$\sin(\theta - \hat{\theta}) = \sin(\theta) \cdot \cos(\hat{\theta}) - \cos(\theta) \cdot \sin(\hat{\theta}) \qquad \textbf{\textit{Eqn. 3-19}}$$

If the deviation between the estimated and the actual angle is very small, then the observer error may be expressed using the following equation:

$$\sin(\theta - \hat{\theta}) \approx \theta - \hat{\theta} \qquad \textbf{\textit{Eqn. 3-20}}$$

The primary benefit of the angle tracking observer utilization, in comparison with the trigonometric method, is its smoothing capability. This filtering is achieved by the integrator and proportional and integral controller, which are connected in series and closed by a unit feedback loop. This block diagram tracks actual rotor angle and speed, and continuously updates their estimations. The angle tracking observer transfer function is expressed as follows:

$$\frac{\hat{\theta}(s)}{\theta(s)} = \frac{K_1(1 + K_2 s)}{s^2 + K_1 K_2 s + K_1} \qquad \textbf{\textit{Eqn. 3-21}}$$

The characteristic polynomial of the angle tracking observer corresponds to the denominator of the following transfer function:

$$s^2 + K_1 K_2 s + K_1 \qquad \textbf{\textit{Eqn. 3-22}}$$

Appropriate dynamic behavior of the angle tracking observer is achieved by placement of the poles of characteristic polynomial. This general method is based on matching the coefficients of characteristic polynomial with the coefficients of general second-order system.

The analog integrators in Figure 3-1, marked as $1/s$ are replaced by an equivalent of the discrete-time integrator using the backward Euler integration method. The discrete-time block diagram of the angle tracking observer is shown in **Figure 3-4**.

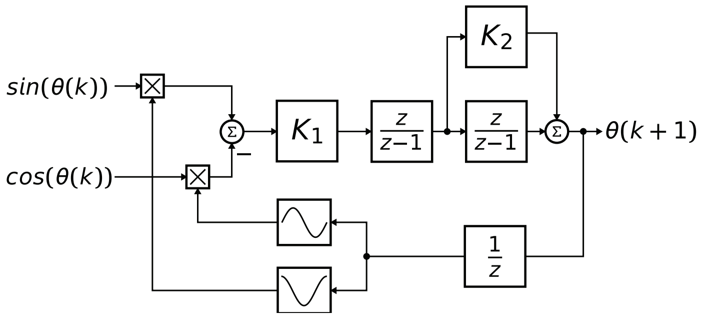


Figure 3-4. Block scheme of discrete-time tracking observer

The essential equations for implementation of the angle tracking observer, according to block scheme in **Figure 3-4**, are as follows:

$$e(k) = \sin(k) \cdot \cos(\hat{\theta}(k)) - \cos(k) \cdot \sin(\hat{\theta}(k))$$ ***Eqn. 3-23***

$$\omega(k) = \omega(k-1) + K_1 \cdot \Delta T_S \cdot e(k)$$ ***Eqn. 3-24***

$$a_2(k) = a_2(k-1) + \Delta T_S \cdot \omega(k)$$ ***Eqn. 3-25***

$$\theta(k) = K_2 \cdot \omega(k) + a_2(k)$$ ***Eqn. 3-26***

In equations Equation 3-23 to Equation 3-26, there are coefficients and quantities that might be greater than one (for example, the actual rotor speed $\omega(k)$) or that are too small to be precisely represented within 16-bit fractional value. Due to this fact a special transformation of equations Equation 3-23 to Equation 3-26 have to be carried out in order to be successfully implemented using fractional arithmetic.

$$K_{1FRAC} = \Delta T_S \cdot \frac{K_1}{\Omega_{MAX}}$$ ***Eqn. 3-27***

$$K_{2FRAC} = K_2 \cdot \frac{\Omega_{MAX}}{\Theta_{MAX}}$$ ***Eqn. 3-28***

$$A_{2FRAC} = \Delta T_S \cdot \frac{\Omega_{MAX}}{\Theta_{MAX}}$$ ***Eqn. 3-29***

where the variables of the angle tracking observer are as follows:

- $e(k)$ is observer error in step k.
- ΔT_S is the sampling period [s].
- $\omega(k)$ is the actual rotor speed [rad/s] in step k.
- $\theta(k)$ is the actual rotor angle [rad] in step k.
- $a_2(k)$ is the actual rotor angle [rad] without scaled addition of speed in step k.

The scaled coefficients which are suitable for implementation on the DSP core are as follows:

$$\mathrm{f16K1Scaled} = K_{1FRAC} \cdot 2^{-\mathrm{i16K1Shift}} \qquad \textbf{Eqn. 3-30}$$

$$\mathrm{f16K2Scaled} = K_{2FRAC} \cdot 2^{-\mathrm{i16K2Shift}} \qquad \textbf{Eqn. 3-31}$$

$$\mathrm{f16A2Scaled} = A_{2FRAC} \cdot 2^{-\mathrm{i16A2Shift}} \qquad \textbf{Eqn. 3-32}$$

3.4.7 Return

The function returns an estimation of the actual rotor angle as 16-bit fractional value.

3.4.8 Range Issues

The function works with the 16-bit signed fractional values in the range <-1,1).

3.4.9 Special Issues

The **ACLIB_AngleTrackObsrv** function requires the saturation mode to be turned on.

3.4.10 Implementation

Example 3-3. implementation Code

```
#include "gflib.h"
#include "mclib.h"
#include "aclib.h"

MCLIB_ANGLE_T              mcAngle;
ACLIB_ANGLE_TRACK_OBSRV_T  acAngleTrackObsrv;
Frac16                     f16PositionOut;

void main (void)
{

acAngleTrckObsrv.f32Speed           = FRAC32(0);
acAngleTrckObsrv.f32A2                = FRAC32(0);
acAngleTrckObsrv.f16Theta             = FRAC16(0);
acAngleTrckObsrv.f16SinEstim          = FRAC16(0);
acAngleTrckObsrv.f16CosEstim          = FRAC16(0);
acAngleTrckObsrv.f16K1Gain            = ANGLETRACKOBSRV_K1_SCALED;
acAngleTrckObsrv.i16K1GainShift       = ANGLETRACKOBSRV_K1_SHIFT;
acAngleTrckObsrv.f16K2Gain            = ANGLETRACKOBSRV_K2_SCALED;
acAngleTrckObsrv.i16K2GainShift       = ANGLETRACKOBSRV_K2_SHIFT;
acAngleTrckObsrv.f16A2Gain            = ANGLETRACKOBSRV_A2_SCALED;
acAngleTrckObsrv.i16A2GainShift       = ANGLETRACKOBSRV_A2_SHIFT;

}

/* Periodical function or interrupt */
void ISR(void)
{
f16PositionOut = ACLIB_AngleTrackObsrv(&mcAngle, &acAngleTrackObsrv);
}
```

3.4.11 Performance

Table 3-10. Performance of ACLIB_AngleTrackObsrv function

Code Size (words)	V2: 80 + 38, V3: 73 + 28 (GFLIB_SinTlr)	
Data Size (words)	0 + 10 (GFLIB_SinTlr)	
Execution Clock	Min	V2: 203, V3: 179 cycles
	Max	V2: 203, V3: 179 cycles

3.5 ACLIB_AngleTrackObsrv12

The function calculates angle tracking observer for determination angular speed and position of input functional signal. This version uses the quicker 12-bit precision sine calculation therefore it is quicker but with reduced precision in comparison to **ACLIB_AngleTrackObsrv**.

3.5.1 Synopsis

```
#include"aclib.h"
Frac16 ACLIB_AngleTrackObsrv12(MCLIB_ANGLE_T *pudtSinCos,
ACLIB_ANGLE_TRACK_OBSRV_T * pudtCtrl)
```

3.5.2 Prototype

```
asm Frac16 ACLIB_AngleTrackObsrv12FAsm(MCLIB_ANGLE_T *pudtSinCos,
ACLIB_ANGLE_TRACK_OBSRV_T * const pudtCtrl)
```

V3 core version:

```
asm Frac16 ACLIB_V3AngleTrackObsrv12FAsm(MCLIB_ANGLE_T *pudtSinCos,
ACLIB_ANGLE_TRACK_OBSRV_T * const pudtCtrl)
```

3.5.3 Arguments

Table 3-11. Function Arguments

Name	In/ Out	Format	Valid Range	Description
*pudtSinCos	in	MCLIB_ANGLE_T	N/A	Input signal of sine, cosine components to be filtered
*pudtCtrl	in/out	ACLIB_ANGLE_TRACK_OBSRV_T	N/A	Pointer to an angle tracking observer structure ACLIB_ANGLE_TRACK_OBSRV_T, which contains algorithm coefficients

Table 3-12. User type definitions

Typedef	Name	In/ Out	Format	Valid Range	Description
MCLIB_ANGLE_T	f16Sin	In	SF16	$8000... $7FFF	Sine component to be estimated
	f16Cos	In	SF16	$8000... $7FFF	Cosine component to be estimated
ACLIB_ANGLE_TRACK_OBSRV_T	f32Speed	in/out	SF32	0x80000000... 0x7FFFFFFF	Estimated speed as output of the first numerical integrator
	f32A2	in/out	SF32	0x80000000... 0x7FFFFFFF	Output of the second numerical integrator
	f16Theta	in/out	SF16	$8000... $7FFF	Estimated position
	f16SinEstim	in	SF16	$8000... $7FFF	Sine signal to be estimated
	f16CosEstim	in	SF16	$8000... $7FFF	Cosine signal to be estimated
	f16K1Gain	in	SF16	$8000... $7FFF	K1 coefficient scaled to fractional range
	i16K1GainShift	in	SI16	-F...F	Scaling shift
	f16K2Gain	in	SF16	$8000... $7FFF	K2 coefficient scaled to fractional range
	i16K2GainShift	in	SI16	-F...F	Scaling shift
	f16A2Gain	in	SF16	$8000... $7FFF	Scaling coefficient due to numerical integration
	i16A2GainShift	in	SI16	-F...F	Scaling shift

3.5.4 Availability

This library module is available in the C-callable interface assembly formats.

This module is targeted for the 56800E and 56800Ex platforms.

3.5.5 Dependencies

List of all dependent files is as follows:

- 56800E_types.h
- 56800E_MCLIB library
- 56800E_GFLIB library
- ACLIB_AngleTrackObsrvAsm.h
- aclib.h

3.5.6 Description

This function calculates the angle tracking observer algorithm. It is recommended to call this function at every sampling period. It requires two input arguments as sine and cosine samples. The practical implementation of the angle tracking observer algorithm is described below.

The angle tracking observer compares values of the input signals $\sin(\theta)$, $\cos(\theta)$ with their corresponding estimations $\sin(\hat{\theta})$, $\cos(\hat{\theta})$. As in any common closed-loop systems, the intent is to minimize observer error towards zero value. The observer error is given here by subtraction of the estimated resolver rotor angle $\hat{\theta}$ from the actual rotor angle θ (see **Figure 3-5**).

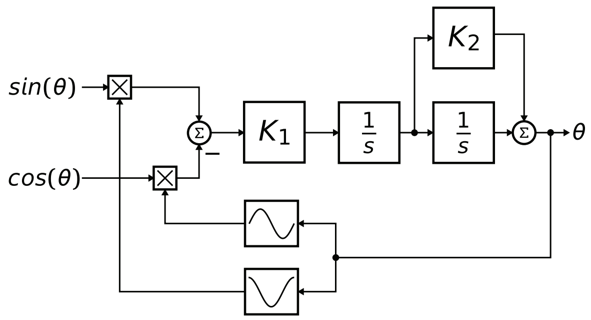


Figure 3-5. Block scheme of the angle tracking observer

Note that mathematical expression of observer error is known as the formula of the difference of two angles:

$$\sin(\theta - \hat{\theta}) = \sin(\theta) \cdot \cos(\hat{\theta}) - \cos(\theta) \cdot \sin(\hat{\theta}) \qquad \textit{Eqn. 3-33}$$

In the case of minimal deviations out of the estimated rotor angle compared to the actual rotor angle, the observer error may be expressed in the following form:

$$\sin(\theta - \hat{\theta}) \approx \theta - \hat{\theta} \qquad \textit{Eqn. 3-34}$$

The primary benefit of the angle tracking observer utilization, in comparison with the trigonometric method, is its smoothing capability. This filtering is achieved by the integrator and proportional and integral controller, which are connected in series and closed by a unit feedback loop. This block diagram tracks actual rotor angle and speed, and continuously updates their estimations. The angle tracking observer transfer function is expressed as follows:

$$\frac{\hat{\theta}(s)}{\theta(s)} = \frac{K_1(1 + K_2 s)}{s^2 + K_1 K_2 s + K_1} \qquad \textit{Eqn. 3-35}$$

The characteristic polynomial of the angle tracking observer corresponds to the denominator of following transfer function:

$$s^2 + K_1 K_2 s + K_1 \qquad \textit{Eqn. 3-36}$$

Appropriate dynamic behavior of the angle tracking observer is achieved by placement of the poles of the characteristic polynomial. This general method is based on matching the coefficients of the characteristic polynomial with the coefficients of the general second-order system.

The analog integrators in Figure 3-5, marked as $1/s$ are replaced by an equivalent of the discrete-time integrator using the backward Euler integration method. The discrete-time block diagram of the angle tracking observer is shown in **Figure 3-6**.

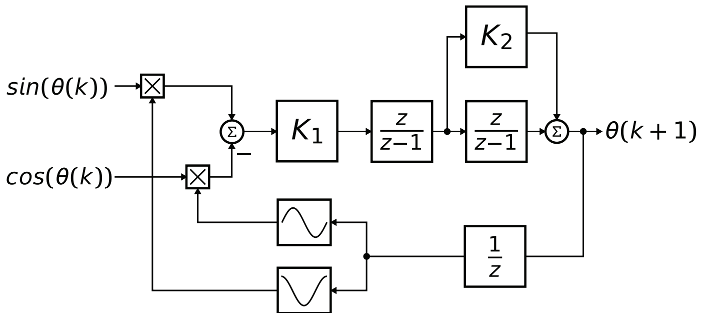


Figure 3-6. Block scheme of discrete-time tracking observer

The essential equations for implementation of the angle tracking observer, according to block scheme in **Figure 3-6**, are as follows:

$$e(k) = \sin(\mathrm{k}) \cdot \cos(\hat{\theta}(k)) - \cos(\mathrm{k}) \cdot \sin(\hat{\theta}(k)) \qquad \textbf{\textit{Eqn. 3-37}}$$

$$\omega(k) = \omega(k-1) + K_1 \cdot \Delta T_S \cdot e(k) \qquad \textbf{\textit{Eqn. 3-38}}$$

$$a_2(k) = a_2(k-1) + \Delta T_S \cdot \omega(k) \qquad \textbf{\textit{Eqn. 3-39}}$$

$$\theta(k) = K_2 \cdot \omega(k) + a_2(k) \qquad \textbf{\textit{Eqn. 3-40}}$$

In equations Equation 3-37 to Equation 3-40, there are coefficients and quantities that might be greater than one (for example, the actual rotor speed $\omega(k)$) or that are too small to be precisely represented within 16-bit fractional value. Due to this fact a special transformation of equations Equation 3-37 to Equation 3-40 have to be carried out in order to be successfully implemented using fractional arithmetic.

$$K_{1FRAC} = \Delta T_S \cdot \frac{K_1}{\Omega_{MAX}} \qquad \textbf{\textit{Eqn. 3-41}}$$

$$K_{2FRAC} = K_2 \cdot \frac{\Omega_{MAX}}{\Theta_{MAX}} \qquad \textbf{\textit{Eqn. 3-42}}$$

$$A_{2FRAC} = \Delta T_S \cdot \frac{\Omega_{MAX}}{\Theta_{MAX}} \qquad \textbf{\textit{Eqn. 3-43}}$$

where the variables of the angle tracking observer are as follows:

- $e(k)$ is observer error in step k.
- ΔT_S is the sampling period [s].
- $\omega(k)$ is the actual rotor speed [rad/s] in step k.
- $\theta(k)$ is the actual rotor angle [rad] in step k.
- $a_2(k)$ is the actual rotor angle [rad] without scaled addition of speed in step k.

The scaled coefficients which are suitable for implementation on the DSP core are as follows:

$$\mathrm{f16K1Scaled} = K_{1FRAC} \cdot 2^{-\mathrm{i16K1Shift}} \qquad \textit{Eqn. 3-44}$$

$$\mathrm{f16K2Scaled} = K_{2FRAC} \cdot 2^{-\mathrm{i16K2Shift}} \qquad \textit{Eqn. 3-45}$$

$$\mathrm{f16A2Scaled} = A_{2FRAC} \cdot 2^{-\mathrm{i16A2Shift}} \qquad \textit{Eqn. 3-46}$$

3.5.7 Return

The function returns an estimation of the actual rotor angle as 16-bit fractional value.

3.5.8 Range Issues

The function works with the 16-bit signed fractional values in the range <-1,1).

3.5.9 Special Issues

The **ACLIB_AngleTrackObsrv12** function requires the saturation mode to be turned on.

3.5.10 Implementation

Example 3-4. implementation Code

```
#include "gflib.h"
#include "mclib.h"
#include "aclib.h"

MCLIB_ANGLE_T             mcAngle;
ACLIB_ANGLE_TRACK_OBSRV_T acAngleTrackObsrv;
Frac16                    f16PositionOut;

void main (void)
{

acAngleTrckObsrv.f32Speed           = FRAC32(0);
acAngleTrckObsrv.f32A2              = FRAC32(0);
acAngleTrckObsrv.f16Theta           = FRAC16(0);
acAngleTrckObsrv.f16SinEstim        = FRAC16(0);
acAngleTrckObsrv.f16CosEstim        = FRAC16(0);
acAngleTrckObsrv.f16K1Gain          = ANGLETRACKOBSRV_K1_SCALED;
acAngleTrckObsrv.i16K1GainShift     = ANGLETRACKOBSRV_K1_SHIFT;
acAngleTrckObsrv.f16K2Gain          = ANGLETRACKOBSRV_K2_SCALED;
acAngleTrckObsrv.i16K2GainShift     = ANGLETRACKOBSRV_K2_SHIFT;
acAngleTrckObsrv.f16A2Gain          = ANGLETRACKOBSRV_A2_SCALED;
acAngleTrckObsrv.i16A2GainShift     = ANGLETRACKOBSRV_A2_SHIFT;

}

/* Periodical function or interrupt */
void ISR(void)
{
f16PositionOut = ACLIB_AngleTrackObsrv12(&mcAngle, &acAngleTrackObsrv);
}
```

3.5.11 Performance

Table 3-13. Performance of ACLIB_AngleTrackObsrv12 function

Code Size (words)	V2: 80 + 25, V3: 73 + 25 (GFLIB_Sin12Tlr)	
Data Size (words)	0 + 5 (GFLIB_Sin12Tlr)	
Execution Clock	Min	V2: 186, V3: 171 cycles
	Max	V2: 186, V3: 171 cycles

3.6 ACLIB_PMSMBemfObsrvDQ

The function calculates the algorithm of back electromotive force observer in rotating reference frame.

3.6.1 Synopsis

```
#include ”aclib.h”
void ACLIB_PMSMBemfObsrvDQ(MCLIB_2_COOR_SYST_D_Q_T *pudtCurrentDQ,
MCLIB_2_COOR_SYST_D_Q_T *pudtVoltageDQ, Frac16 f16Speed,
ACLIB_BEMF_OBSRV_DQ_T *pudtCtrl)
```

3.6.2 Prototype

```
asm void ACLIB_PMSMBemfObsrvDQFAsm(MCLIB_2_COOR_SYST_D_Q_T
*pudtCurrentDQ, MCLIB_2_COOR_SYST_D_Q_T *pudtVoltageDQ, Frac16 f16Speed,
ACLIB_BEMF_OBSRV_DQ_T *pudtCtrl)
```

V3 core version:

```
asm void ACLIB_V3PMSMBemfObsrvDQFAsm(MCLIB_2_COOR_SYST_D_Q_T
*pudtCurrentDQ, MCLIB_2_COOR_SYST_D_Q_T *pudtVoltageDQ, Frac16 f16Speed,
ACLIB_BEMF_OBSRV_DQ_T *pudtCtrl)
```

3.6.3 Arguments

Table 3-14. Function Arguments

Name	In/ Out	Format	Valid Range	Description
*pudtCurrentDQ	in	MCLIB_2_COOR_SYST_D_Q_T	N/A	Pointer to structure which contain input signal of d/q current components
*pudtVoltageDQ	in	MCLIB_2_COOR_SYST_D_Q_T	N/A	Pointer to structure which contain input signal of d/q voltage components
f16Frac	in/out	SF16	N/A	Fraction value of electrical speed.
*pudtCtrl	in/out	ACLIB_BEMF_OBSRV_DQ_T	N/A	Pointer to an observer structure, which contains coefficients.

Table 3-15. User Types

Typedef	Name	Format	Valid Range	Description
ACLIB_BEMF_OBSRV_AB_T	udtEObsrv.f32D	SF32	0x80000000... 0x7FFFFFFF	Estimated back-EMF voltage in d-axis
	udtEObsrv.f32Q	SF32	0x80000000... 0x7FFFFFFF	Estimated back-EMF voltage in q-axis
	udtIObsrv.f32D	SF32	0x80000000... 0x7FFFFFFF	Estimated current in d-axis
	udtIObsrv.f32Q	SF32	0x80000000... 0x7FFFFFFF	Estimated current in q-axis
	udtCtrl.f32ID_1	SF32	0x80000000... 0x7FFFFFFF	State variable in alpha part of the observer; integral part at step k-1;
	udtCtrl.f32IQ_1	SF32	0x80000000... 0x7FFFFFFF	State variable in beta part of the observer; integral part at step k-1;
	udtCtrl.f16PropGain	SF16	$8000... $7FFF	Observer proportional gain
	udtCtrl.i16PropGainShift	SI16	-F...F	Observer proportional gain shift
	udtCtrl.f16IntegGain	SF16	$8000... $7FFF	Observer integral gain
	udtCtrl.i16IntegGainShift	SI16	-F...F	Observer integral gain shift
	f16Error	SF16	$8000... $7FFF	Estimated phase error between real d/q frame system and estimated d/q reference system
	f16IGain	SF16	$8000... $7FFF	Scaling coefficient for current I_{FRAC}
	f16UGain	SF16	$8000... $7FFF	Scaling coefficient for voltage U_{FRAC}
	f16WIGain	SF16	$8000... $7FFF	Scaling coefficient for angular speed WI_{FRAC}
	f16EGain	SF16	$8000... $7FFF	Scaling coefficient for back-EMF E_{FRAC}

3.6.4 Availability

This library module is available in the C-callable interface assembly formats.

This module is targeted for the 56800E and 56800Ex platforms.

3.6.5 Dependencies

List of all dependent files is as follows:

- 56800E_types.h

- 56800E_MCLIB library
- 56800E_GFLIB library
- ACLIB_PMSMBemfObsrvDQAsm.h
- aclib.h

3.6.6 Description

The estimation method for the rotor position and angular speed is based on the motor mathematical model of interior PMSM motor with an extended electromotive force function which is realized in estimated quasi synchronous reference frame $\gamma\delta$ as depicted on Figure 3-7.

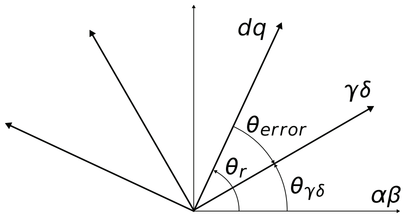


Figure 3-7. Estimated $\gamma\delta$ and real rotor dq synchronous reference frames

The back-EMF observer detects the generated motor voltages induced by the permanent magnets. A tracking observer uses the back-EMF signals to calculate the position and speed of the rotor. The transformed model is then derived as follows:

$$\begin{bmatrix} u_\gamma \\ u_\delta \end{bmatrix} = \begin{bmatrix} R_S + sL_D & -\omega_r L_Q \\ \omega_r L_Q & R_S + sL_D \end{bmatrix} \begin{bmatrix} i_\gamma \\ i_\delta \end{bmatrix} + (\Delta L \cdot (\omega_e i_D - i_Q') + k_e \omega_e) \cdot \begin{bmatrix} -\sin(\theta_{error}) \\ \cos(\theta_{error}) \end{bmatrix} \quad \textbf{\textit{Eqn. 3-47}}$$

where,

- R_s stator resistance
- L_D, L_Q - D-axis and Q-axis inductance
- k_e back-EMF constant
- ω_e angular electrical speed
- u_D, u_Q stator voltages
- i_D, i_Q stator currents
- s operator of derivative
- i_q' - first derivative of i_q current

Block diagram of the observer in the estimated reference frame is shown on Figure 3-8. The observer compensator is substituted by a standard PI controller. As can be noted from Figure 3-8, observer model and hence also PI controller gains in both axis are identical to each other.

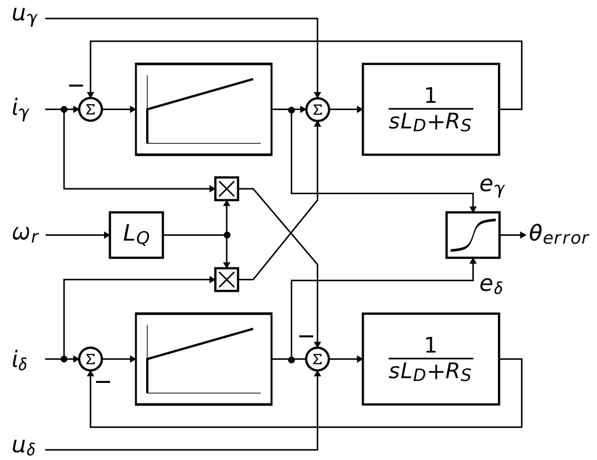


Figure 3-8. Block diagram of proposed Luenberger type stator current observer acting as state filter for back-EMF.

The position estimation can now be performed by extracting the θ_{error} term from the model and adjusting the position of the estimated reference frame such as to achieve $\theta_{error} = 0$. Because the θ_{error} term is only included in the saliency-based EMF component of both u_γ, u_δ axis voltage equations, the Luenberger based disturbance observer is designed to observe these voltage components u_γ, u_δ. The position displacement information θ_{error} is then obtained from estimated back-EMFs as follows:

$$\theta_{error} = \operatorname{atan}\left(\frac{-u_\gamma}{u_\delta}\right) \qquad \textbf{\textit{Eqn. 3-48}}$$

The estimated position $\hat{\theta}_r$ can be obtained by driving the position of the estimated reference frame such as to achieve zero displacement $\theta_{error} = 0$. The phase locked loop mechanism can be adopted, where the loop compensator ensures correct tracking of the actual rotor flux position by keeping the error signal θ_{error} to be zeroed, $\theta_{error} = 0$.

A perfect match between the actual and estimated motor model parameters is assumed, and then back-EMF transfer function is simplified as follows:

$$\hat{E}_{\alpha\beta}(s) = -E_{\alpha\beta}(s) \cdot \left[\frac{F_c(s)}{sL_D + R_S + F_C(s)}\right] \qquad \textbf{\textit{Eqn. 3-49}}$$

Appropriate dynamic behavior of the back-EMF observer is achieved by placement of the poles of the stator current observer characteristic polynomial. This general method is based on matching the coefficients of the characteristic polynomial with the coefficients of the general second-order system.

Back-EMF observer is Luenberger type observer with motor model which is realized in fixed point arithmetic transformed using backward Euler transformation.

$$\boldsymbol{i}_{FRFAC}(k) = U_{FRAC} \cdot \boldsymbol{u}_{FRAC}(k) + E_{FRAC} \cdot \boldsymbol{e}_{FRAC}(k) + WI_{FRAC} \cdot \omega_{eFRAC}(k) \cdot \boldsymbol{i'}_{FRAC}(k) + I_{FRAC} \cdot \boldsymbol{i}_{FRAC}(k-1) \qquad \text{Eqn. 3-50}$$

where,

- $\boldsymbol{i}_{FRFAC}(k) = [i_\gamma, i_\delta]$ is fractional representation of stator current vector.
- $\boldsymbol{u}_{FRAC}(k) = [u_\gamma, u_\delta]$ is fractional representation of stator voltage vector.
- $\boldsymbol{e}_{FRAC}(k) = [e_\gamma, e_\delta]$ is fractional representation of stator back-EMF voltage vector.
- $\boldsymbol{i'}_{FRFAC}(k) = [i_\delta, -i_\gamma]$ is fractional representation of complementary stator current vector.
- $\omega_{FRFAC}(k)$ is fractional representation of angular speed.

Scaling coefficients relating to maximal values are expressed as follows:

$$U_{FRAC} = \frac{\Delta T_S}{L_D + \Delta T_S R_S} \cdot \frac{U_{MAX}}{I_{MAX}} \qquad \text{Eqn. 3-51}$$

$$E_{FRAC} = \frac{\Delta T_S}{L_D + \Delta T_S R_S} \cdot \frac{E_{MAX}}{I_{MAX}} \qquad \text{Eqn. 3-52}$$

$$WI_{FRAC} = \frac{L_Q \cdot \Delta T_S}{L_D + \Delta T_S R_S} \cdot \Omega_{MAX} \qquad \text{Eqn. 3-53}$$

$$I_{FRAC} = \frac{L_D}{L_D + \Delta T_S R_S} \qquad \text{Eqn. 3-54}$$

where,

- ΔT_S sampling time in [sec].
- I_{MAX} maximal peak current in [A].
- E_{MAX} maximal peak back-EMF voltage in [V].
- U_{MAX} maximal peak stator voltage in [V].
- Ω_{MAX} maximal angular speed in [rad/sec].

If a Luenberger type stator current observer is properly designed in the stationary reference frame, the back-EMF can be estimated as a disturbance, produced by the observer controller. This is valid only when the back-EMF term is not included in the observer model. The observer is actually a closed loop current observer so it acts as a state filter for the back-EMF term.

The estimate of extended EMF term can be derived from Equation 3-49 as follows:

$$-\frac{\hat{E}_{\gamma\delta}(s)}{E_{\gamma\delta}(s)} = \frac{sK_P + K_I}{s^2 L_D + sR_S + sK_P + K_I} \qquad \text{Eqn. 3-55}$$

The observer controller can be designed by comparing the closed loop characteristic polynomial with that of a standard second order system as:

$$s^2 + \frac{K_P + R_S}{L_D} \cdot s + \frac{K_I}{L_D} = s^2 + 2\xi\omega_0 s + \omega_0^2 \qquad \textit{Eqn. 3-56}$$

where,

- ω_0 is the natural frequency of the closed loop system (loop bandwidth).
- ξ is the loop attenuation.

3.6.7 Returns

The function returns a phase error between real rotating reference frame and estimated one.

3.6.8 Range Issues

The function works with the 16-bit signed fractional values in the range <-1,1).

3.6.9 Special Issues

The **ACLIB_PMSMBemfObsrvDQ** function requires the saturation mode to be turned on.

3.6.10 Implementation

Example 3-5. Implementation Code

```
#include "gflib.h"
#include "mclib.h"
#include "aclib.h"

MCLIB_2_COOR_SYST_D_Q_T   mcIdq,mcUdq;
ACLIB_BEMF_OBSRV_DQ_T     acBemfObsrv;
Frac16                    f16Speed;

void main (void)
{

acBemfObsrv.udtIObsrv.f32D = FRAC32(0.0);
acBemfObsrv.udtIObsrv.f32Q = FRAC32(0.0);
acBemfObsrv.udtEObsrv.f32D = FRAC32(0.0);
acBemfObsrv.udtEObsrv.f32Q = FRAC32(0.0);
acBemfObsrv.udtCtrl.f32ID_1= FRAC32(0.0);
acBemfObsrv.udtCtrl.f32IQ_1= FRAC32(0.0);
acBemfObsrv.udtCtrl.f16PropGain = BEMFOBSRV_DQ_PROP_GAIN_SCALED;
acBemfObsrv.udtCtrl.i16PropGainShift = BEMFOBSRV_DQ_PROP_GAIN_SHIFT;
acBemfObsrv.udtCtrl.f16IntegGain = BEMFOBSRV_DQ_INTEG_GAIN_SCALED;
acBemfObsrv.udtCtrl.i16IntegGainShift = BEMFOBSRV_DQ_INTEG_GAIN_SHIFT;
acBemfObsrv.f16IGain = BEMFOBSRV_DQ_I_SCALED;
acBemfObsrv.f16UGain = BEMFOBSRV_DQ_U_SCALED;
acBemfObsrv.f16EGain = BEMFOBSRV_DQ_E_SCALED;
acBemfObsrv.f16WIGain = BEMFOBSRV_DQ_WI_SCALED;

}

/* Periodical function or interrupt */
void ISR(void)
{
ACLIB_PMSMBemfObsrvDQ(&mcIdq, &mcUdq, f16Speed, &acBemfObsrv);
}
```

3.6.11 Performance

Table 3-16. Performance of ACLIB_PMSMBemfObsrvDQ function

Code Size (words)	V2: 158 + 102, V3: 139 + 102 (GFLIB_AtanYX)	
Data Size (words)	0 + 33 (GFLIB_AtanYX)	
Execution Clock	Min	V2: 259, V3: 243 cycles
	Max	V2: 335, V3: 319 cycles

3.7 ACLIB_TrackObsrv

The function calculates tracking observer for determination angular speed and position of input error functional signal.

3.7.1 Synopsis

```
#include"aclib.h"
Frac16 ACLIB_TrackObsrv(Frac16 f16Error, ACLIB_TRACK_OBSRV_T *pudtCtrl)
```

3.7.2 Prototype

```
asm Frac16 ACLIB_TrackObsrvFAsm(Frac16 f16Error, ACLIB_TRACK_OBSRV_T
*pudtCtrl)
```

V3 core version:

```
asm Frac16 ACLIB_V3TrackObsrvFAsm(Frac16 f16Error, ACLIB_TRACK_OBSRV_T
*pudtCtrl)
```

3.7.3 Arguments

Table 3-17. Function Arguments

Name	In/ Out	Format	Valid Range	Description
f16Error	in	SF16	$8000... $7FFF	input signal representing phase error of system to be estimated
*pudtCtrl	in/out	ACLIB_TRACK_OBSRV_T	N/A	pointer to a racking observer structure ACLIB_TRACK_OBSRV_T, which contains algorithm coefficients

Table 3-18. User type definitions

Typedef	Name	In/ Out	Format	Valid Range	Description
ACLIB_TRACK_OBSRV_T	f32Theta	in/out	SF32	0x80000000... 0x7FFFFFFF	Estimated position as output of the second numerical integrator
	f32Speed	in/out	SF32	0x80000000... 0x7FFFFFFF	Estimated speed as output of the first numerical integrator
ACLIB_TRACK_OBSRV_T	f32I_1	in/out	SF32	0x80000000... 0x7FFFFFFF	State variable in controller part of the observer; integral part at step k-1
	f16PropGain	in	SF16	$8000... $7FFF	Observer proportional gain
	i16PropGainShift	in	SI16	-F...F	Observer proportional gain shift
	f16IntegGain	in	SF16	$8000... $7FFF	Observer integral gain
	i16IntegGainShift	in	SI16	-F...F	Observer integral gain shift
	f16ThGain	in	SF16	$8000... $7FFF	Scaling coefficient for output integrator of position
	i16ThGainShift	in	SI16	-F...F	Scaling coefficient shift for output integrator of position

3.7.4 Availability

This library module is available in the C-callable interface assembly formats.

This module is targeted for the 56800E and 56800Ex platforms.

3.7.5 Dependencies

List of all dependent files is as follows:

- 56800E_types.h
- 56800E_GFLIB library
- 56800E_MCLIB library
- ACLIB_TrackObsrvAsm.h
- aclib.h

3.7.6 Description

This function calculates the tracking observer algorithm where phase locked loop mechanism is adopted. It is recommended to call this function at every sampling period. It requires a single input argument as phase error. Such phase tracking observer, with standard PI controller used as the loop compensator, is depicted on Figure 3-9.

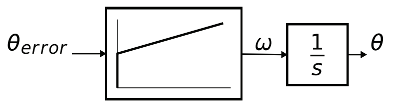


Figure 3-9. Block diagram of proposed PLL scheme for position estimation

Depicted tracking observer structure has the transfer function as follows:

$$\frac{\hat{\theta}(s)}{\theta(s)} = \frac{sK_p + K_i}{s^2 + sK_p + K_i} \qquad \textbf{Eqn. 3-57}$$

where, the controller gains K_p and K_i are calculated by comparing the characteristic polynomial of the resulting transfer function to a standard second order system polynomial.

The essential equations for implementation of the tracking observer, according to block scheme in Figure 3-9, are as follows:

$$\begin{aligned} \omega(k) &= K_p \cdot e(k) + \Delta T_S \cdot K_i \cdot e(k) + I(k-1) \\ I(k) &= \Delta T_S \cdot K_i \cdot e(k) + I(k-1) \end{aligned} \qquad \textbf{Eqn. 3-58}$$

$$\theta(k) = \theta(k-1) + \Delta T_S \cdot \omega(k) \qquad \textbf{Eqn. 3-59}$$

In equations Equation 3-58 and Equation 3-59, there are coefficients and quantities that might be greater than one (for example, the actual rotor speed $\omega(k)$) or that are too small to be precisely represented within 16-bit fractional value. Due to this fact a special transformation have to be carried out in order to be successfully implemented using fractional arithmetic.

$$K_{pFRAC} = \frac{K_p}{\Omega_{MAX}} \qquad \textbf{Eqn. 3-60}$$

$$K_{iFRAC} = \Delta T_S \cdot \frac{K_i}{\Omega_{MAX}} \qquad \textbf{Eqn. 3-61}$$

$$T_{hFRAC} = \Delta T_S \cdot \frac{\Omega_{MAX}}{\Theta_{MAX}} \qquad \textbf{Eqn. 3-62}$$

where the variables of the angle tracking observer are as follows:

- $e(k)$ is observer error in step k.
- ΔT_S is the sampling period $[\mathrm{s}]$.
- $\omega(k)$ is the actual rotor speed $[\mathrm{rad/s}]$ in step k.
- $\theta(k)$ is the actual rotor angle [rad] in step k.

The scaled coefficients which are suitable for implementation on the DSP core are as follows:

$$\text{f16KPScaled} = K_{pFRAC} \cdot 2^{-\text{i16KPShift}}$$ ***Eqn. 3-63***

$$\text{f16KIScaled} = K_{iFRAC} \cdot 2^{-\text{i16KIShift}}$$ ***Eqn. 3-64***

$$\text{f16ThScaled} = T_{hFRAC} \cdot 2^{-\text{i16ThShift}}$$ ***Eqn. 3-65***

3.7.7 Returns

The function returns an estimation of the actual rotor angle as 16-bit fractional value.

3.7.8 Range Issues

The function works with the 16-bit signed fractional values in the range <-1,1).

3.7.9 Special Issues

The **ACLIB_TrackObsrv** function requires the saturation mode to be turned on. Upon completion of the function the saturation mode is set off.

3.7.10 Implementation

Example 3-6. Implementation Code

```
#include "aclib.h"

ACLIB_TRACK_OBSRV_T acTo;
Frac16           f16ThetaError;
Frac16           f16PositionEstim;

void main (void)
{
acTo.f32Theta = FRAC32(0.0);
acTo.f32Speed = FRAC32(0.0);
acTo.f32I_1 = FRAC32(0.0);
acTo.f16PropGain = TRACKOBSRV_PROP_GAIN_SCALED;
acTo.i16PropGainShift = TRACKOBSRV_PROP_GAIN_SHIFT;
acTo.f16IntegGain = TRACKOBSRV_INTEG_GAIN_SCALED;
acTo.i16IntegGainShift = TRACKOBSRV_INTEG_GAIN_SHIFT;
acTo.f16ThGain = TRACKOBSRV_TH_SCALED;
acTo.i16ThGainShift = TRACKOBSRV_TH_SHIFT;
}

/* Periodical function or interrupt */
void ISR(void)
{
f16PositionEstim = ACLIB_TrackObsrv(f16ThetaError, &acTo);
}
```

3.7.11 See Also

3.7.12 Performance

Table 3-19. Performance of ACLIB_TrackObsrv function

Code Size (words)	V2: 51, V3: 49	
Data Size (words)	0	
Execution Clock	Min	V2: 85, V3: 78 cycles
	Max	V2: 85, V3: 78 cycles

3.8 ACLIB_IntegratorInitVal

The function initializes the initial value of the **ACLIB_Integrator** algorithm.

3.8.1 Synopsis

```
#include"aclib.h"
Frac16 ACLIB_IntegratorInitVal(Frac16 f16InitVal, ACLIB_INTEGRATOR_T
*pudtIntg)
```

3.8.2 Prototype

```
asm Frac16 ACLIB_IntegratorInitValFAsm(Frac16 f16InitVal,
ACLIB_INTEGRATOR_T *pudtIntg)
```

3.8.3 Arguments

Table 3-20. Function Arguments

Name	In/ Out	Format	Valid Range	Description
f16InitVal	in	SF16	$8000... $7FFF	Initial value of the integrator
*pudtIntg	in/out	ACLIB_INTEGRATOR_T	N/A	pointer to structure which contain parameters of numerical integrator

Table 3-21. User type definitions

Typedef	Name	In/ Out	Format	Valid Range	Description
ACLIB_INTEGRATOR_T	f32Integ_1	in	SF32	0x80000000... 0x7FFFFFFF	state variable of integration in step k-1
	f16IntegScaled	in	SF16	$8000... $7FFF	scaling coefficient of integrator gain
	i16IntegShift	in	SI16	-F...F	integrator gain shift

3.8.4 Availability

This library module is available in the C-callable interface assembly formats.

This module is targeted for the 56800E and 56800Ex platforms.

3.8.5 Dependencies

List of all dependent files is as follows:

- 56800E_types.h

- 56800E_MCLIB library
- 56800E_GFLIB library
- ACLIB_IntegratorAsm.h
- aclib.h

3.8.6 Description

The **ACLIB_IntegratorInitVal** function initializes the integral portion of the **ACLIB_Integrator** algorithm so as the output is this value in the first step.

3.8.7 Returns

None.

3.8.8 Range Issues

The function works with the 16-bit signed fractional values in the range <-1,1).

3.8.9 Special Issues

The **ACLIB_IntegratorInitVal** function is saturation mode independent.

3.8.10 Returns

The function returns an integrated value of its input variable.

3.8.11 Implementation

Example 3-7. Implementation Code

```
#include "aclib.h"

ACLIB_INTEGRATOR_T        acIntegrator;
Frac16                    f16X;
Frac16                    f16Intg;

void main (void)
{
acIntegrator.f32I_1 = FRAC32(0.0);
acIntegrator.f16IntegScaled = INTEG_GAIN_SCALED;
acIntegrator.i16IntegShift = INTEG_GAIN_SHIFT;

/* Integral portion initialization to zero */
ACLIB_IntegratorInitVal(0, &acIntegrator);

}

/* Periodical function or interrupt */
void ISR(void)
{
```

```
f16Intg = ACLIB_Integrator(f16X, &acIntegrator);
}
```

3.8.12 Performance

Table 3-22. Performance of ACLIB_IntegratorInitVal function

Code Size (words)	4	
Data Size (words)	0	
Execution Clock	Min	23 cycles
	Max	23 cycles

3.9 ACLIB_Integrator

The function calculates the algorithm of numerical integrator of its input.

3.9.1 Synopsis

```
#include”aclib.h”
Frac16 ACLIB_Integrator(Frac16 f16Xinp, ACLIB_INTEGRATOR_T *pudtIntg)
```

3.9.2 Prototype

```
asm Frac16 ACLIB_IntegratorFAsm(Frac16 f16Xinp, ACLIB_INTEGRATOR_T
*pudtIntg)
```

3.9.3 Arguments

Table 3-23. Function Arguments

Name	In/ Out	Format	Valid Range	Description
f16Xinp	in	SF16	$8000... $7FFF	input variable to be integrated
*pudtIntg	in/out	ACLIB_INTEGRATOR_T	N/A	pointer to structure which contain parameters of numerical integrator

Table 3-24. User type definitions

Typedef	Name	In/ Out	Format	Valid Range	Description
ACLIB_INTEGRATOR_T	f32Integ_1	in	SF32	0x80000000... 0x7FFFFFFF	state variable of integration in step k-1
	f16IntegGain	in	SF16	$8000... $7FFF	scaling coefficient of integrator gain
	i16IntegGainShift	in	SI16	-F...F	integrator gain shift

3.9.4 Availability

This library module is available in the C-callable interface assembly formats.

This module is targeted for the 56800E and 56800Ex platforms.

3.9.5 Dependencies

List of all dependent files is as follows:

- 56800E_types.h
- 56800E_MCLIB library

- 56800E_GFLIB library
- ACLIB_IntegratorAsm.h
- aclib.h

3.9.6 Description

Numerical integration is the approximate computation of an integral using numerical techniques. The integrator is approximated by the backward Euler method, also known as backward rectangular or right-hand approximation as follows.

$$I(k) = \Delta T_S \cdot in(k) + I(k-1) \qquad \textbf{Eqn. 3-66}$$

where, the variables of the angle tracking observer are as follows:

- $in(k)$ is integrator input in step *k*.
- ΔT_S is the sampling period $[\mathrm{s}]$.
- $I(k)$ is the integrator value in step *k*.

The integrator coefficient might be greater than one or that is too small to be precisely represented within 16-bit fractional value. Due to this fact a special transformation have to be carried out in order to be successfully implemented using fractional arithmetic.

$$I_{FRAC} = \Delta T_S \cdot \frac{IN_{MAX}}{OUT_{MAX}} \qquad \textbf{Eqn. 3-67}$$

The scaled coefficient which is suitable for implementation on the DSP core is follows:

$$\mathrm{f16IScaled} = I_{FRAC} \cdot 2^{-\mathrm{i16IShift}} \qquad \textbf{Eqn. 3-68}$$

3.9.7 Returns

The integrated value.

3.9.8 Range Issues

The function works with the 16-bit signed fractional values in the range <-1,1).

3.9.9 Special Issues

The **ACLIB_Integrator** function requires the saturation mode to be turned on if the output variable is required to be limited otherwise output variable is naturally wrapped around.

3.9.10 Returns

The function returns an integrated value of its input variable.

3.9.11 Implementation

Example 3-8. Implementation Code

```
#include "aclib.h"

ACLIB_INTEGRATOR_T        acIntegrator;
Frac16                    f16X;
Frac16                    f16Intg;

void main (void)
{
acIntegrator.f32I_1 = FRAC32(0.0);
acIntegrator.f16IntegGain = INTEG_GAIN_SCALED;
acIntegrator.i16IntegGainShift = INTEG_GAIN_SHIFT;

/* Integral portion initialization to zero */
ACLIB_IntegratorInitVal(0, &acIntegrator);

}

/* Periodical function or interrupt */
void ISR(void)
{
f16Intg = ACLIB_Integrator(f16X, &acIntegrator);
}
```

3.9.12 Performance

Table 3-25. Performance of ACLIB_Integrator function

Code Size (words)	17	
Data Size (words)	0	
Execution Clock	Min	39 cycles
	Max	39 cycles

Appendix A Revision History

Table 0-1. Revision history

Revision number	Date	Subsequent changes
0	02/2014	Initial release

56800Ex_ACLIB
Rev. 0, 02/2014