AMCLIB User's Guide

DSP56800EX

Document Number: DSP56800EXAMCLIBUG
Rev. 4, 05/2019

Contents

Section number **Title** **Page**

Chapter 1
Library

Chapter 2
Algorithms in detail

Chapter 1
Library

1.1 Introduction

1.1.1 Overview

This user's guide describes the Advanced Motor Control Library (AMCLIB) for the family of DSP56800EX core-based digital signal controllers. This library contains optimized functions.

1.1.2 Data types

AMCLIB supports several data types: (un)signed integer, fractional, and accumulator. The integer data types are useful for general-purpose computation; they are familiar to the MPU and MCU programmers. The fractional data types enable powerful numeric and digital-signal-processing algorithms to be implemented. The accumulator data type is a combination of both; that means it has the integer and fractional portions.

The following list shows the integer types defined in the libraries:

- Unsigned 16-bit integer —<0 ; 65535> with the minimum resolution of 1
- Signed 16-bit integer —<-32768 ; 32767> with the minimum resolution of 1
- Unsigned 32-bit integer —<0 ; 4294967295> with the minimum resolution of 1
- Signed 32-bit integer —<-2147483648 ; 2147483647> with the minimum resolution of 1

The following list shows the fractional types defined in the libraries:

- Fixed-point 16-bit fractional —$<-1 ; 1 - 2^{-15}>$ with the minimum resolution of 2^{-15}
- Fixed-point 32-bit fractional —$<-1 ; 1 - 2^{-31}>$ with the minimum resolution of 2^{-31}

The following list shows the accumulator types defined in the libraries:

- Fixed-point 16-bit accumulator —<-256.0 ; 256.0 - 2^{-7}> with the minimum resolution of 2^{-7}
- Fixed-point 32-bit accumulator —<-65536.0 ; 65536.0 - 2^{-15}> with the minimum resolution of 2^{-15}

1.1.3 API definition

AMCLIB uses the types mentioned in the previous section. To enable simple usage of the algorithms, their names use set prefixes and postfixes to distinguish the functions' versions. See the following example:

```
f32Result = MLIB_Mac_F32lss(f32Accum, f16Mult1, f16Mult2);
```

where the function is compiled from four parts:

- MLIB—this is the library prefix
- Mac—the function name—Multiply-Accumulate
- F32—the function output type
- lss—the types of the function inputs; if all the inputs have the same type as the output, the inputs are not marked

The input and output types are described in the following table:

Table 1-1. Input/output types

Type	Output	Input
frac16_t	F16	s
frac32_t	F32	l
acc32_t	A32	a

1.1.4 Supported compilers

AMCLIB for the DSP56800EX core is written in assembly language with C-callable interface. The library is built and tested using the following compilers:

- CodeWarrior™ Development Studio

For the CodeWarrior™ Development Studio, the library is delivered in the *amclib.lib* file.

The interfaces to the algorithms included in this library are combined into a single public interface include file, *amclib.h*. This is done to lower the number of files required to be included in your application.

1.1.5 Library configuration

1.1.6 Special issues

1. The equations describing the algorithms are symbolic. If there is positive 1, the number is the closest number to 1 that the resolution of the used fractional type allows. If there are maximum or minimum values mentioned, check the range allowed by the type of the particular function version.
2. The library functions require the core saturation mode to be turned off, otherwise the results can be incorrect. Several specific library functions are immune to the setting of the saturation mode.
3. The library functions round the result (the API contains Rnd) to the nearest (two's complement rounding) or to the nearest even number (convergent round). The mode used depends on the core option mode register (OMR) setting. See the core manual for details.
4. All non-inline functions are implemented without storing any of the volatile registers (refer to the compiler manual) used by the respective routine. Only the 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.

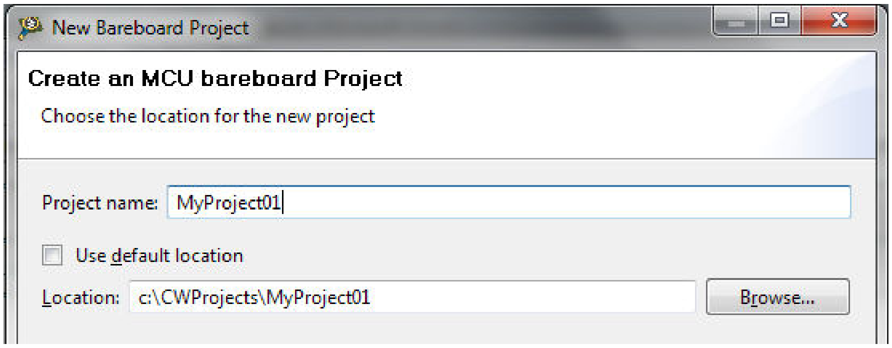
1.2 Library integration into project (CodeWarrior™ Development Studio)

This section provides a step-by-step guide to quickly and easily integrate the AMCLIB into an empty project using CodeWarrior™ Development Studio. This example uses the MC56F84789 part, and the default installation path (C:\NXP\RTCESL \DSP56800EX_RTCESL_4.5) is supposed. If you have a different installation path, you must use that path instead.

1.2.1 New project

To start working on an application, create a new project. If the project already exists and is open, skip to the next section. Follow the steps given below to create a new project.

1. Launch CodeWarrior™ Development Studio.
2. Choose File > New > Bareboard Project, so that the "New Bareboard Project" dialog appears.
3. Type a name of the project, for example, MyProject01.
4. If you don't use the default location, untick the “Use default location” checkbox, and type the path where you want to create the project folder; for example, C:\CWProjects\MyProject01, and click Next. See Figure 1-1.



Create an MCU bareboard Project

Choose the location for the new project

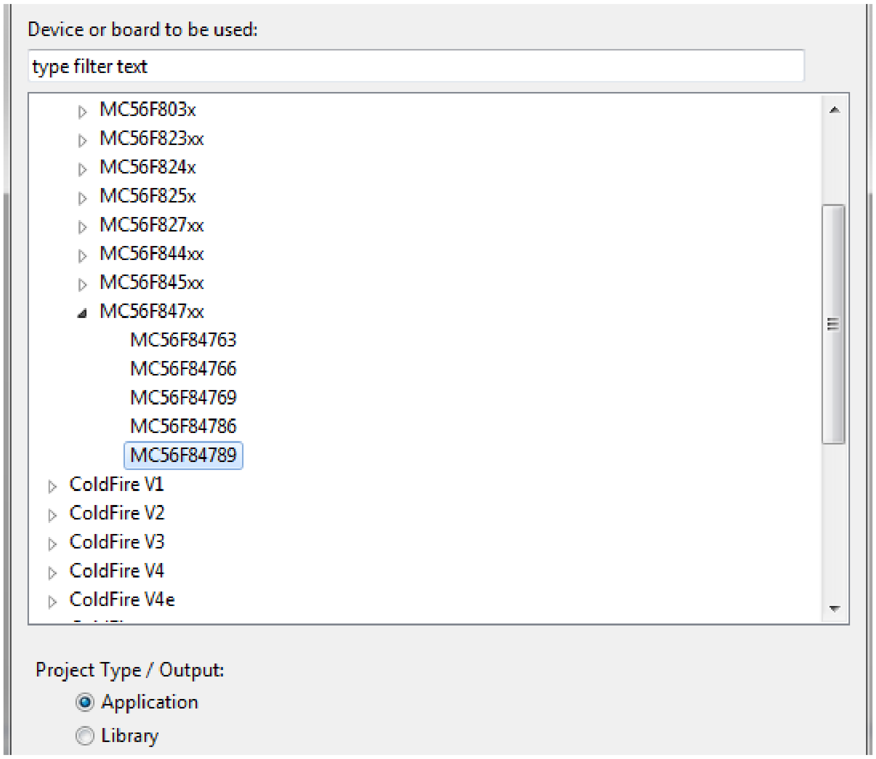
Project name: MyProject01

☐ Use default location

Location: c:\CWProjects\MyProject01

Figure 1-1. Project name and location

5. Expand the tree by clicking the 56800/E (DSC) and MC56F84789. Select the Application option and click Next. See Figure 1-2.



Device or board to be used:

type filter text

- MC56F803x
- MC56F823xx
- MC56F824x
- MC56F825x
- MC56F827xx
- MC56F844xx
- MC56F845xx
- MC56F847xx
  - MC56F84763
  - MC56F84766
  - MC56F84769
  - MC56F84786
  - MC56F84789
- ColdFire V1
- ColdFire V2
- ColdFire V3
- ColdFire V4
- ColdFire V4e

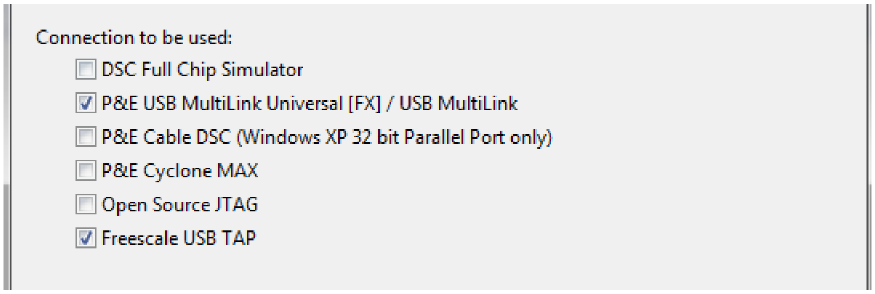
Project Type / Output:

● Application

○ Library

Figure 1-2. Processor selection

6. Now select the connection that will be used to download and debug the application. In this case, select the option P&E USB MultiLink Universal[FX] / USB MultiLink and Freescale USB TAP, and click Next. See Figure 1-3.



Connection to be used:

☐ DSC Full Chip Simulator

☑ P&E USB MultiLink Universal [FX] / USB MultiLink

☐ P&E Cable DSC (Windows XP 32 bit Parallel Port only)

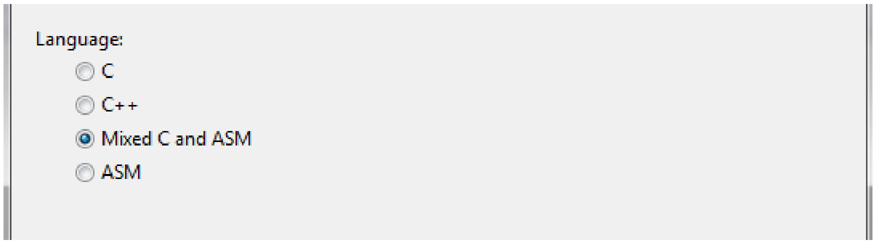
☐ P&E Cyclone MAX

☐ Open Source JTAG

☑ Freescale USB TAP

Figure 1-3. Connection selection

7. From the options given, select the Simple Mixed Assembly and C language, and click Finish. See Figure 1-4.



Language:

○ C

○ C++

● Mixed C and ASM

○ ASM

Figure 1-4. Language choice

The new project is now visible in the left-hand part of CodeWarrior™ Development Studio. See Figure 1-5.

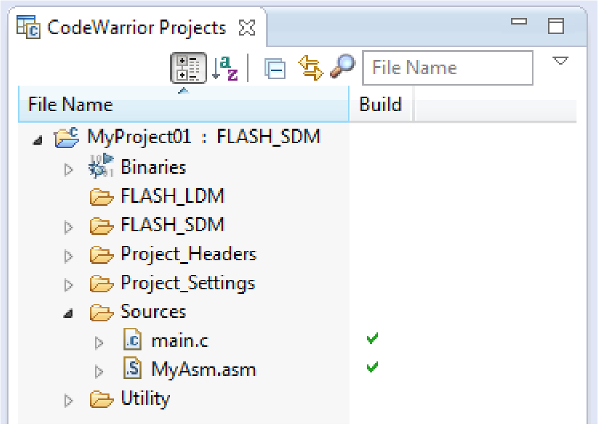
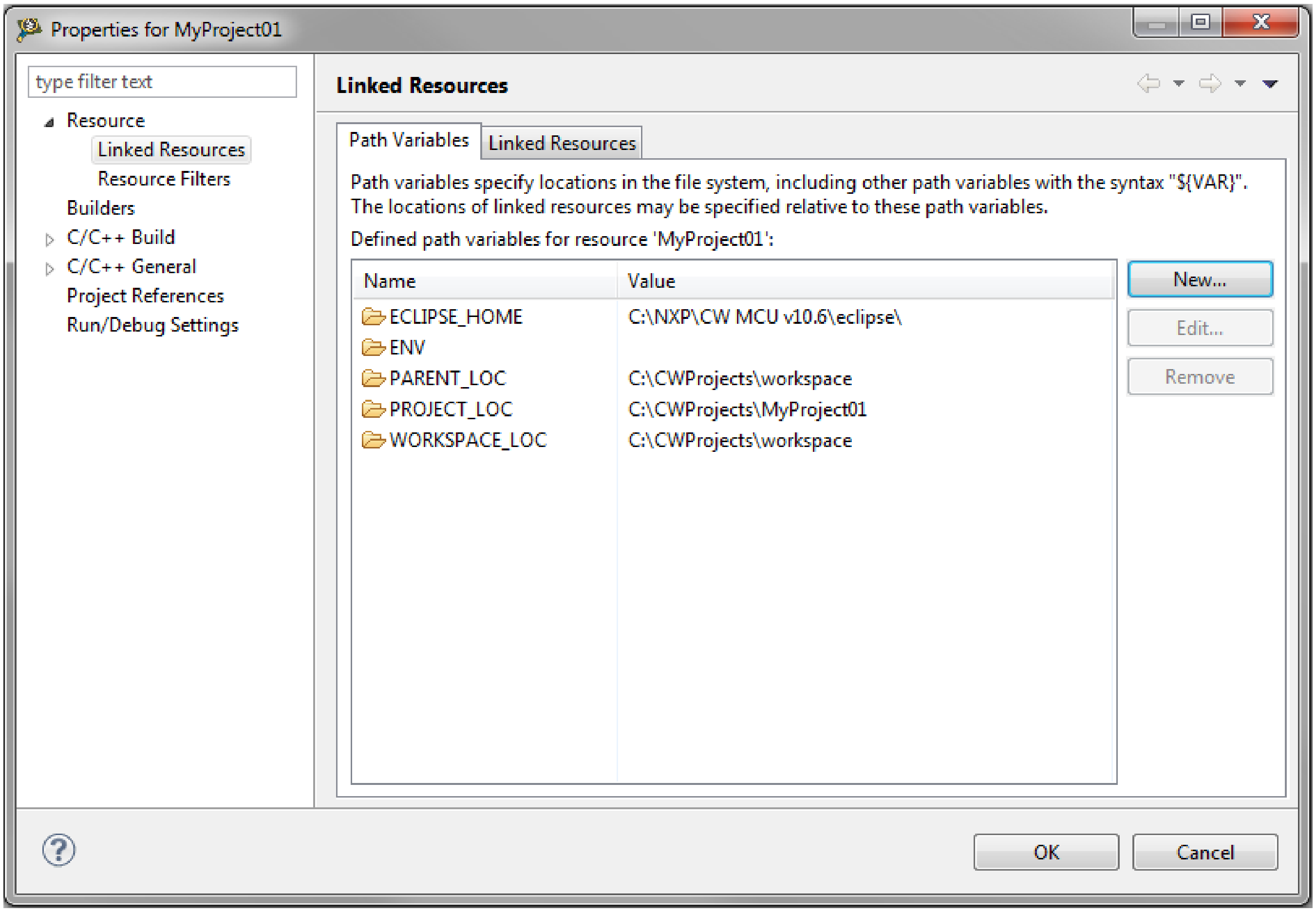


Figure 1-5. Project folder

1.2.2 Library path variable

To make the library integration easier, create a variable that will hold the information about the library path.

1. Right-click the MyProject01 node in the left-hand part and click Properties, or select Project > Properties from the menu. The project properties dialog appears.
2. Expand the Resource node and click Linked Resources. See Figure 1-6.



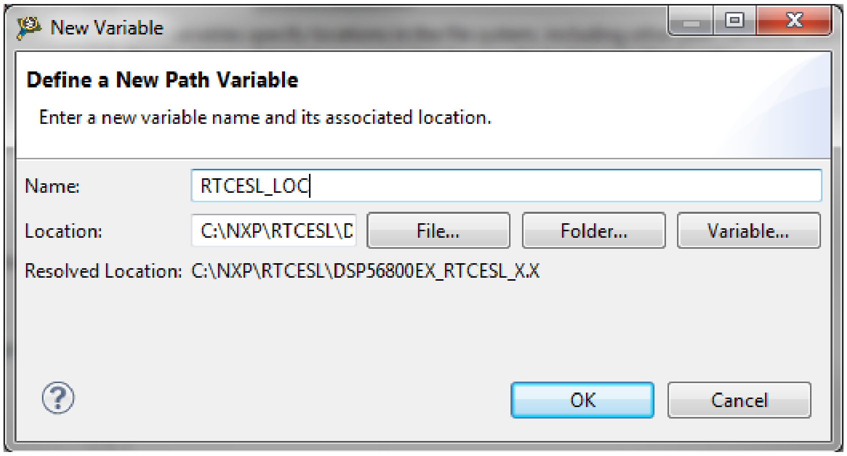
Path variables specify locations in the file system, including other path variables with the syntax "${VAR}". The locations of linked resources may be specified relative to these path variables.

Defined path variables for resource 'MyProject01':

Name	Value
ECLIPSE_HOME	C:\NXP\CW MCU v10.6\eclipse\
ENV	
PARENT_LOC	C:\CWProjects\workspace
PROJECT_LOC	C:\CWProjects\MyProject01
WORKSPACE_LOC	C:\CWProjects\workspace

Figure 1-6. Project properties

3. Click the 'New…' button on the right-hand side.
4. In the dialog that appears (see Figure 1-7), type this variable name into the Name box: RTCESL_LOC
5. Select the library parent folder by clicking 'Folder…' or just typing the following path into the Location box: C:\NXP\RTCESL\DSP56800EX_RTCESL_4.5_CW and click OK.
6. Click OK in the previous dialog.



Define a New Path Variable

Enter a new variable name and its associated location.

Name: RTCESL_LOC

Location: C:\NXP\RTCESL\D

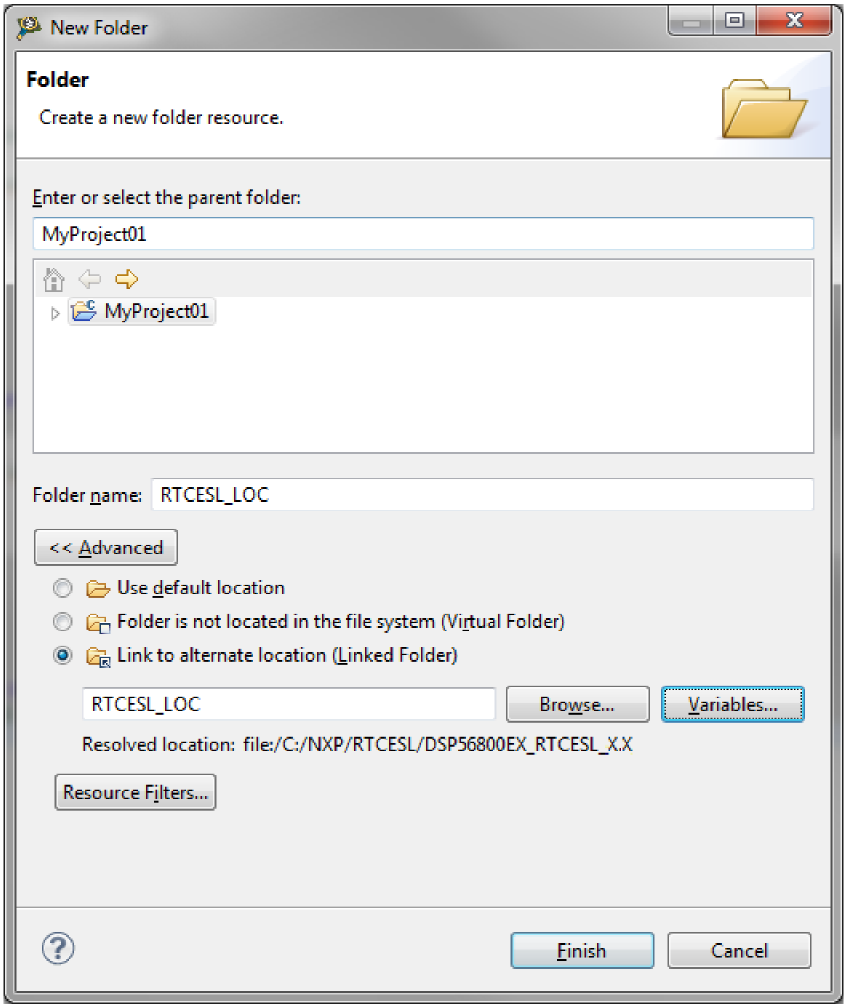
Resolved Location: C:\NXP\RTCESL\DSP56800EX_RTCESL_X.X

Figure 1-7. New variable

1.2.3 Library folder addition

To use the library, add it into the CodeWarrior Project tree dialog.

1. Right-click the MyProject01 node in the left-hand part and click New > Folder, or select File > New > Folder from the menu. A dialog appears.
2. Click Advanced to show the advanced options.
3. To link the library source, select the third option—Link to alternate location (Linked Folder).
4. Click Variables…, and select the RTCESL_LOC variable in the dialog that appears, click OK, and/or type the variable name into the box. See Figure 1-8.
5. Click Finish, and you will see the library folder linked in the project. See Figure 1-9



Enter or select the parent folder: MyProject01

▷ MyProject01

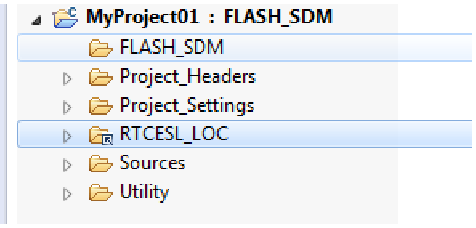
Folder name: RTCESL_LOC

○ Use default location
○ Folder is not located in the file system (Virtual Folder)
● Link to alternate location (Linked Folder)

RTCESL_LOC

Resolved location: file:/C:/NXP/RTCESL/DSP56800EX_RTCESL_X.X

Figure 1-8. Folder link



◢ MyProject01 : FLASH_SDM
- FLASH_SDM
- ▷ Project_Headers
- ▷ Project_Settings
- ▷ RTCESL_LOC
- ▷ Sources
- ▷ Utility

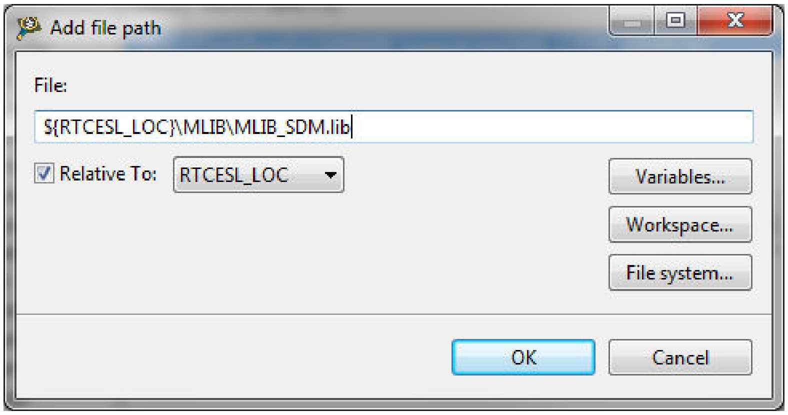
Figure 1-9. Projects libraries paths

1.2.4 Library path setup

AMCLIB requires MLIB and GFLIB and GMCLIB to be included too. Therefore, the following steps show the inclusion of all dependent modules.

1. Right-click the MyProject01 node in the left-hand part and click Properties, or select Project > Properties from the menu. A dialog with the project properties appears.
2. Expand the C/C++ Build node, and click Settings.

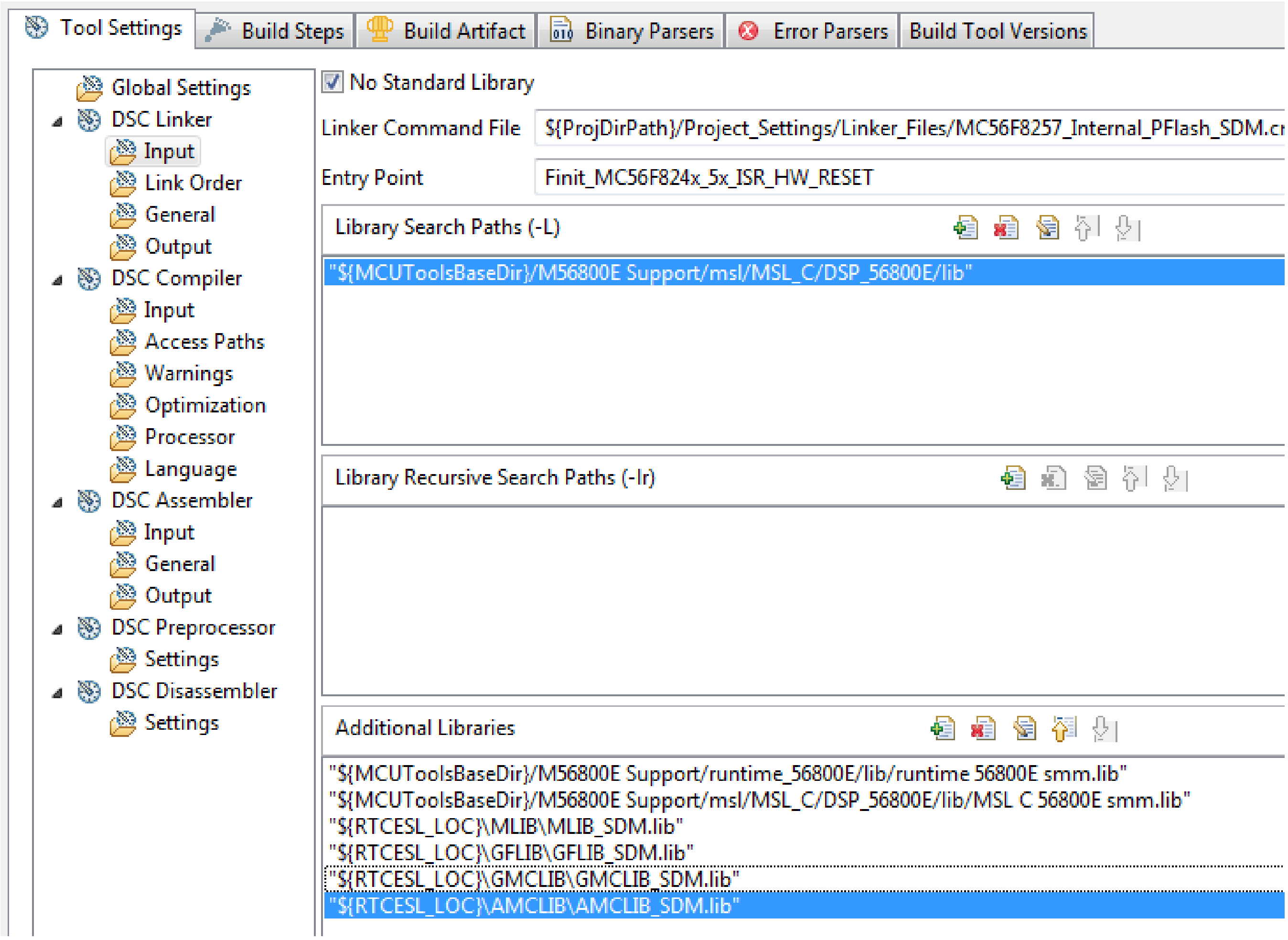
3. In the right-hand tree, expand the DSC Linker node, and click Input. See Figure 1-11.
4. In the third dialog Additional Libraries, click the 'Add…' icon, and a dialog appears.
5. Look for the RTCESL_LOC variable by clicking Variables…, and then finish the path in the box by adding one of the following:
   - ${RTCESL_LOC}\MLIB\mlib_SDM.lib—for small data model projects
   - ${RTCESL_LOC}\MLIB\mlib_LDM.lib—for large data model projects
6. Tick the box Relative To, and select RTCESL_LOC next to the box. See Figure 1-9. Click OK.
7. Click the 'Add…' icon in the third dialog Additional Libraries.
8. Look for the RTCESL_LOC variable by clicking Variables…, and then finish the path in the box by adding one of the following:
   - ${RTCESL_LOC}\GFLIB\gflib_SDM.lib—for small data model projects
   - ${RTCESL_LOC}\GFLIB\gflib_LDM.lib—for large data model projects
9. Tick the box Relative To, and select RTCESL_LOC next to the box. Click OK.
10. Click the 'Add…' icon in the Additional Libraries dialog.
11. Look for the RTCESL_LOC variable by clicking Variables…, and then finish the path in the box by adding one of the following:
    - ${RTCESL_LOC}\GMCLIB\gmclib_SDM.lib—for small data model projects
    - ${RTCESL_LOC}\GMCLIB\gmclib_LDM.lib—for large data model projects
12. Tick the box Relative To, and select RTCESL_LOC next to the box. Click OK.
13. Click the 'Add…' icon in the Additional Libraries dialog.
14. Look for the RTCESL_LOC variable by clicking Variables…, and then finish the path in the box by adding one of the following:
    - ${RTCESL_LOC}\AMCLIB\amclib_SDM.lib—for small data model projects
    - ${RTCESL_LOC}\AMCLIB\amclib_LDM.lib—for large data model projects
15. Now, you will see the libraries added in the box. See Figure 1-11.



File: ${RTCESL_LOC}\MLIB\MLIB_SDM.lib

☑ Relative To: RTCESL_LOC

Figure 1-10. Library file inclusion



☑ No Standard Library

Linker Command File: ${ProjDirPath}/Project_Settings/Linker_Files/MC56F8257_Internal_PFlash_SDM.cr

Entry Point: Finit_MC56F824x_5x_ISR_HW_RESET

Library Search Paths (-L)

"${MCUToolsBaseDir}/M56800E Support/msl/MSL_C/DSP_56800E/lib"

Library Recursive Search Paths (-lr)

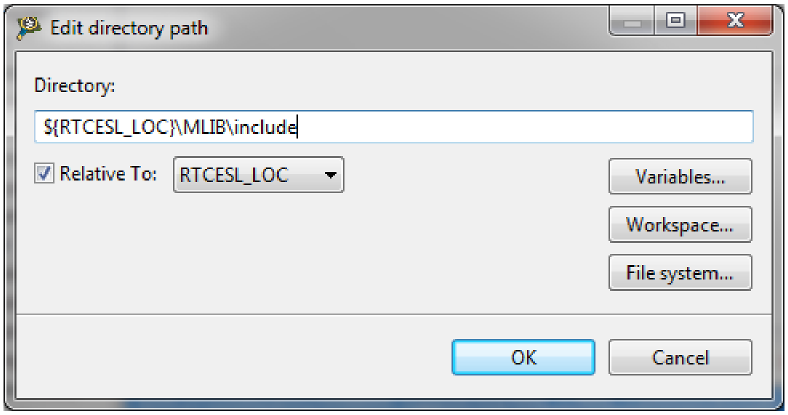
Additional Libraries

"${MCUToolsBaseDir}/M56800E Support/runtime_56800E/lib/runtime 56800E smm.lib"
"${MCUToolsBaseDir}/M56800E Support/msl/MSL_C/DSP_56800E/lib/MSL C 56800E smm.lib"
"${RTCESL_LOC}\MLIB\MLIB_SDM.lib"
"${RTCESL_LOC}\GFLIB\GFLIB_SDM.lib"
"${RTCESL_LOC}\GMCLIB\GMCLIB_SDM.lib"
"${RTCESL_LOC}\AMCLIB\AMCLIB_SDM.lib"

Figure 1-11. Linker setting

16. In the tree under the DSC Compiler node, click Access Paths.
17. In the Search User Paths dialog (#include "…"), click the 'Add…' icon, and a dialog will appear.
18. Look for the RTCESL_LOC variable by clicking Variables…, and then finish the path in the box to be: ${RTCESL_LOC}\MLIB\include.
19. Tick the box Relative To, and select RTCESL_LOC next to the box. See Figure 1-12. Click OK.
20. Click the 'Add…' icon in the Search User Paths dialog (#include "…").
21. Look for the RTCESL_LOC variable by clicking Variables…, and then finish the path in the box to be: ${RTCESL_LOC}\GFLIB\include.
22. Tick the box Relative To, and select RTCESL_LOC next to the box. Click OK.
23. Click the 'Add…' icon in the Search User Paths dialog (#include "…").
24. Look for the RTCESL_LOC variable by clicking Variables…, and then finish the path in the box to be: ${RTCESL_LOC}\GMCLIB\include.
25. Tick the box Relative To, and select RTCESL_LOC next to the box. Click OK.
26. Click the 'Add…' icon in the Search User Paths dialog (#include "…").

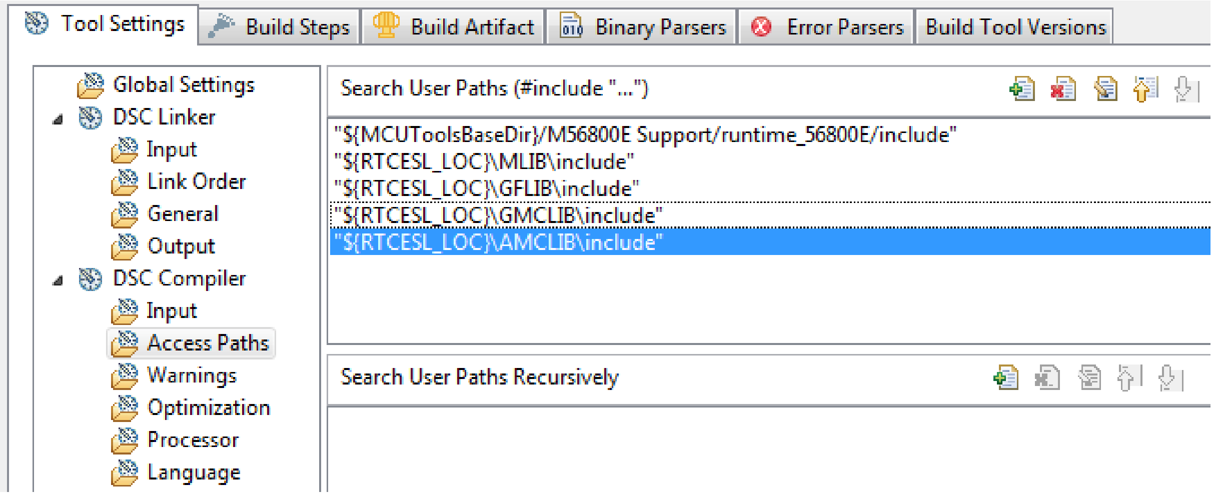
27. Look for the RTCESL_LOC variable by clicking Variables…, and then finish the path in the box to be: ${RTCESL_LOC}\AMCLIB\include.
28. Tick the box Relative To, and select RTCESL_LOC next to the box. Click OK.
29. Now you will see the paths added in the box. See Figure 1-13. Click OK.



Directory: ${RTCESL_LOC}\MLIB\include

☑ Relative To: RTCESL_LOC

Figure 1-12. Library include path addition



Search User Paths (#include "...")

```
"${MCUToolsBaseDir}/M56800E Support/runtime_56800E/include"
"${RTCESL_LOC}\MLIB\include"
"${RTCESL_LOC}\GFLIB\include"
"${RTCESL_LOC}\GMCLIB\include"
"${RTCESL_LOC}\AMCLIB\include"
```

Figure 1-13. Compiler setting

The final step is typing the #include syntax into the code. Include the library into the *main.c* file. In the left-hand dialog, open the Sources folder of the project, and double-click the *main.c* file. After the *main.c* file opens up, include the following lines into the #include section:

```
#include "mlib.h"
#include "gflib.h"
#include "gmclib.h"
#include "amclib.h"
```

When you click the Build icon (hammer), the project will be compiled without errors.

Chapter 2
Algorithms in detail

2.1 AMCLIB_ACIMCtrlMTPA

The AMCLIB_ACIMCtrlMTPA function enables to minimize the ACIM losses by applying the Max Toque per Ampere (MTPA) strategy. The principle is derived from the ACIM torque equation:

$$T(\Theta_I) = \frac{3}{2} \cdot P_P \cdot \frac{L_m^2}{L_r} \cdot i_{sd}(\Theta_I) \cdot i_{sq}(\Theta_I) = \frac{3}{4} \cdot P_P \cdot \frac{L_m^2}{L_r} \cdot |i_{sdq}| \cdot sin(2 \cdot \Theta_I)$$

Equation 1

where:

- i_{sd} is the D component of the stator current vector
- i_{sq} is the Q component of the stator current vector
- i_{sdq} is the stator current vector
- θ_I is the angle of stator the current vector
- L_r is the rotor equivalent inductance
- L_m is the mutual equivalent inductance
- P_P is the motor pole pair number constant
- T is the motor mechanic torque

Motor torque depends on the angle of the stator current vector. Maximum eficency (minimum stator joule losses) can be calculated when motor torque differential is equal zero:

$$\frac{dT(\Theta_I)}{d\Theta_I} = \frac{3}{4} \cdot P_P \cdot \frac{L_m^2}{L_r} \cdot |i_{sdq}| \cdot cos(2 \cdot \Theta_I) = 0 \Rightarrow \Theta_I = \frac{\pi}{4}$$

Equation 2

It is clear that the stator current components must be the same values to achieve the$\theta_I = \pi/4$ angle. The MTPA stator current vector trajectory in consideration of the i_{sd} limits given by the minimal field excitation and current limitations is shown in Figure 2-1).

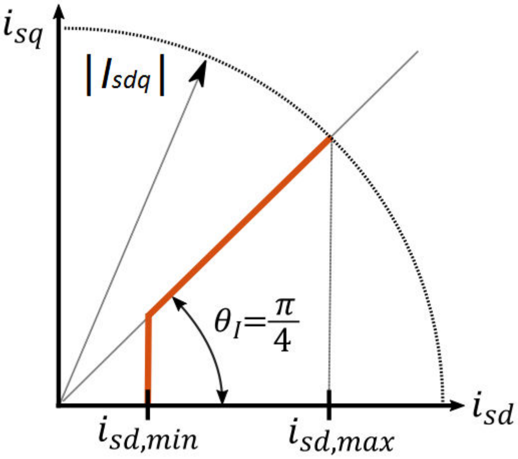


Figure 2-1. Minimal losses stator current vector trajectory with limits

2.1.1 Available versions

The available versions of the AMCLIB_ACIMCtrlMTPA function are shown in the following table:

Table 2-1. Init function versions

Function name	Input type		Parameters	Result type
	IdMin	IdMax		
AMCLIB_ACIMCtrlMTPAInit_F16	frac16_t	frac16_t	AMCLIB_ACIM_CTRL_MTPA_T_F32 *	void
	The input arguments are the 16-bit fractional type values that contain the limits for i_{sd}. They both are positive values (the minimum must be lower than the maximum) and the pointers to a structure that contains the parameters defined in AMCLIB_ACIM_CTRL_MTPA_T_F32 type description.			

Table 2-2. Function version

Function name	Input type	Parameters	Result type
AMCLIB_ACIMCtrlMTPA_F16	frac16_t	AMCLIB_ACIM_CTRL_MTPA_T_F32 *	frac16_t
	The input arguments are the 16-bit fractional type values that contain the limits for i_{sd}. They both are positive values (the minimum must be lower than the maximum) and the pointers to a structure that contains the parameters defined in AMCLIB_ACIM_CTRL_MTPA_T_F32 type description.		

2.1.2 AMCLIB_ACIM_CTRL_MTPA_T_F32 type description

Variable name	Data type	Description
sIdExpParam	GDFLIB_FILTER_EXP_T_F32	The exponential filter structure for the i_{sd} current filtration. Set by the user.
f16LowerLim	frac16_t	The minimal output limit of i_{sd}. Usually determined from the minimum ACIM rotor flux excitation, as shown in Figure 2-1. Set by the user, must be a positive value lower than the upper limit.
f16UpperLim	frac16_t	The maximal output limit of i_{sd}. Usually determined from the maximum (typically nominal) ACIM current, as shown in Figure 2-1. Set by the user, must be a positive value higher than the lower limit.

2.1.3 Declaration

The available AMCLIB_ACIMCtrlMTPAInit functions have the following declarations:

```
void AMCLIB_ACIMCtrlMTPAInit_F16(frac16_t f16IDMin,frac16_t
f16IDMax,AMCLIB_ACIM_CTRL_MTPA_T_F32 *psCtrl)
```

The available AMCLIB_ACIMCtrlMTPA functions have the following declarations:

```
frac16_t AMCLIB_ACIMCtrlMTPA_F16(frac16_t f16Iq,AMCLIB_ACIM_CTRL_MTPA_T_F32 *psCtrl)
```

2.1.4 Function use

The use of the AMCLIB_ACIMCtrlMTPA function is shown in the following examples:

Fixed-point version:

```
#include "amclib.h"

static AMCLIB_ACIM_CTRL_MTPA_T_F32 sMTPAParam;
static frac16_t f16Isd;
static frac16_t f16Isq;
static frac16_t f16IDMin;
static frac16_t f16IDMax;

void Isr(void);

void main (void)
{
    /* Structure parameter setting */
    sMTPAParam.sIdExpParam.f16A = FRAC16(0.05);
```

```
    f16IDMin = FRAC16(0.1);
    f16IDMax = FRAC16(0.2);

    /* Initialization of the ACIMCtrlMTPA's structure */
   AMCLIB_ACIMCtrlMTPAInit_F16 (f16IDMin, f16IDMax, &sMTPAParam);

    /* Assign Iq value */
    f16Iq = FRAC16(-0.6);
}

/* Periodical function or interrupt */
void Isr(void)
{
    /* Calculating required Isd by MTPA algorithm */
    f16Isd = AMCLIB_ACIMCtrlMTPA_F16(f16Iq, &sMTPAParam);
}
```

2.2 AMCLIB_ACIMRotFluxObsrv

The AMCLIB_ACIMRotFluxObsrv function calculates the ACIM flux estimate and its position (angle) from the available measured signals (currents and voltages). In the case of ACIM FOC, the rotor flux position (angle) is needed to perform the Park transformation.

The closed-loop flux observer is formed from the two most desirable open-loop estimators, which are referred to as the voltage model and the current model (as shown in Figure 2-2). The current model is used for low-speed operation and the voltage model is used for high-speed operation. A smooth transition between these two models is ensured by the PI controller.

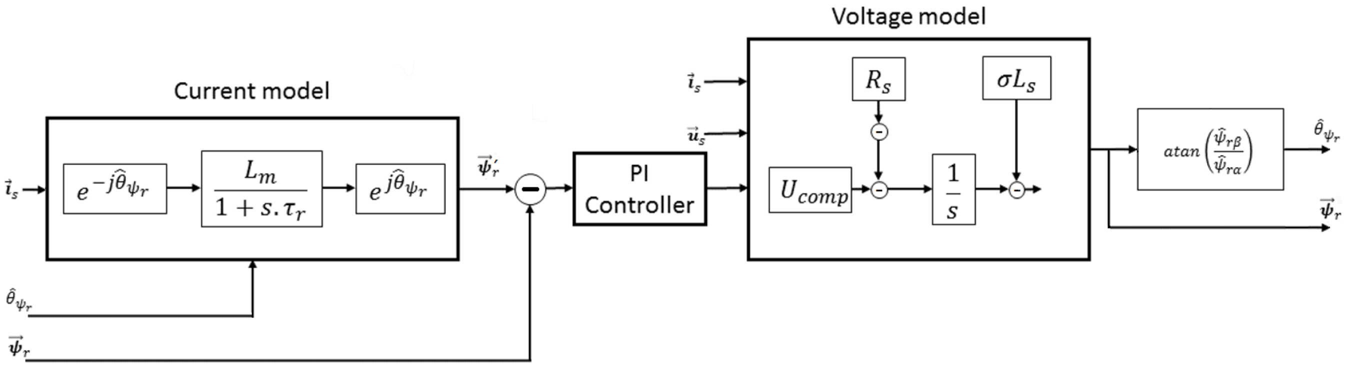


Figure 2-2. ACIM rotor flux observer block diagram

The voltage model (stator model) is used to estimate the stator flux-linkage vector or the rotor flux-linkage vector without a speed signal. The voltage model is derived by integrating the stator voltage equation in the stator stationary coordinates as:

$$
\begin{aligned}
\vec{u}_s &= R_s \cdot \vec{i}_s + \frac{d\vec{\psi}_s}{dt} \\
\vec{\psi}_s &= \int \left(\vec{u}_s - R_s \cdot \vec{i}_s\right) dt \\
\vec{\psi}_r &= \frac{L_r}{L_m}\left(\vec{\psi}_s - L_s \cdot \sigma \cdot \vec{i}_s\right)
\end{aligned}
$$

Equation 3

Expressed in discrete form as:

$$
\begin{aligned}
\psi_{s\alpha}(k) &= \frac{\tau_1}{\tau_1 + T_s}\left[\psi_{s\alpha}(k-1) + T_s \cdot \left(u_{s\alpha}(k) - R_s \cdot i_{s\alpha}(k)\right)\right] \\
\psi_{s\beta}(k) &= \frac{\tau_1}{\tau_1 + T_s}\left[\psi_{s\beta}(k-1) + T_s \cdot \left(u_{s\beta}(k) - R_s \cdot i_{s\beta}(k)\right)\right] \\
\psi_{r\alpha}(k) &= \frac{L_r}{L_m}\left(\psi_{s\alpha}(k) - L_s \cdot \sigma \cdot i_{s\alpha}(k)\right) \\
\psi_{r\beta}(k) &= \frac{L_r}{L_m}\left(\psi_{s\beta}(k) - L_s \cdot \sigma \cdot i_{s\beta}(k)\right)
\end{aligned}
$$

Equation 4

where:

- u_s is the stator voltage vector
- i_s is the stator current vector
- Ψ_s is the stator flux-linkage vector
- Ψ_r is the rotor flux-linkage vector
- ω_r is the rotor electrical angular speed
- ω_s is the electrical angular slip speed
- R_s is the stator resistance
- R_r is the rotor equivalent resistance
- L_s is the stator equivalent inductance
- L_r is the rotor equivalent inductance
- L_m is the mutual equivalent inductance
- τ_r is the motor electrical time constant
- T_s is the sample time
- σ is the motor leakage coefficient

These equations show that the rotor flux linkage is basically the difference between the stator flux-linkage and the leakage flux. The rotor flux equation is used to estimate the respective flux-linkage vector, corresponding angle. The argument Ψ_r of the rotor flux-linkage vector is the rotor field angle θ_{Ψ_r} calculated as:

$$
\theta_{\psi_r} = \operatorname{atan}\left(\frac{\psi_{r\beta}}{\psi_{r\alpha}}\right)
$$

Equation 5

The voltage model (stator model) is sufficiently robust and accurate at higher stator frequencies. Two basic deficiencies can degrade this model as the speed reduces: the integration problem, and model's sensitivity to stator resistance mismatch.

The current model (rotor model) is derived from the differential equation of the rotor winding. The stator coordinate implementation is:

$$\frac{d\overrightarrow{\psi_r}}{dt} = \frac{L_m}{\tau_r}\overrightarrow{i_s} - \frac{1}{\tau_r}\overrightarrow{\psi_r} - j\omega_{slip}\cdot\overrightarrow{\psi_r}$$

Equation 6

When applying field-oriented control assumptions (such as $\Psi_{rq} = 0$), then the rotor flux estimated by the current model in the synchronous rotating frame is:

$$\frac{d\overrightarrow{\psi_{rd}}}{dt} = -\frac{1}{\tau_r}\overrightarrow{\psi_{rd}} + \frac{L_m}{\tau_r}\overrightarrow{i_{sd}}$$

Equation 7

In discrete form:

$$\psi_{rd}(k) = \frac{\tau_r}{\tau_r + T_s}\left[\psi_{rd}(k-1) + T_s\frac{L_m}{\tau_r}i_{sd}(k)\right]$$

Equation 8

The accuracy of the rotor model depends on correct model parameters. It is the rotor time constant in particular that determines the accuracy of the estimated field angle (the most critical variable in a vector-controlled drive).

2.2.1 Available versions

The available versions of the AMCLIB_ACIMRotFluxObsrv function are shown in the following table:

Table 2-3. Init version

Function name	Parameters	Result type
AMCLIB_ACIMRotFluxObsrvInit_F16	AMCLIB_ACIM_ROT_FLUX_OBSRV_T_A32 *	void
	The initialization does not have any input.	

Table 2-4. Function version

Function name	Input/output type		Result type
AMCLIB_ACIMRotFluxObsrv_F16	**Input**	GMCLIB_2COOR_ALBE_T_F16 *	void
		GMCLIB_2COOR_ALBE_T_F16 *	

Table continues on the next page...

Table 2-4. Function version (continued)

Function name	Input/output type		Result type
	Parameters	AMCLIB_ACIM_ROT_FLUX_OBSRV_T_A32 *	
	Rotor flux observer with a 16-bit fractional type inputs: stator current and voltage in alpha-beta coordinates. All are within the full range. The function does not return anything. All calculated variables are stored in the AMCLIB_ACIM_ROT_FLUX_OBSRV_T_A32 structure.		

2.2.2 AMCLIB_ACIM_ROT_FLUX_OBSRV_T_A32 type description

Variable name		Data type	Description
sPsiRotRDQ		GMCLIB_2COOR_DQ_T_F32	The output rotor flux estimated structure calculated from the current model. The structure consists of the D and Q rotor flux components stored for the next steps. The quadrature component is forced to zero value - required by FOC. Calculated by the algorithm for next steps
sPsiRotSAlBe		GMCLIB_2COOR_ALBE_T_F32	The output rotor flux estimated structure calculated from the voltage model. The structure consists of the alpha and beta rotor flux components stored for the next steps. Calculated by the algorithm for next steps
sPsiStatSAlBe		GMCLIB_2COOR_ALBE_T_F32	The output stator flux estimated structure calculated from the voltage model. The structure consists of the alpha and beta stator flux components stored for the next steps. Calculated by the algorithm for next steps
sCtrl	f32CompAlphaInteg_1	frac32_t	The state variable in the alpha part of the controller; integral part at step k-1. Calculated by the algorithm for next steps.
	f32CompBetaInteg_1	frac32_t	The state variable in the beta part of the controller; integral part at step k-1. Calculated by the algorithm for next steps.
	a32PGain	acc32_t	The proportional gain Kp for the stator model PI correction. The parameter is within the range <0 ; 65536.0). Set by the user.
	a32IGain	acc32_t	The integration gain Ki for the stator model PI correction. The parameter is within the range <0 ; 65536.0). Set by the user.
f32KPsiRA1Gain		frac32_t	The gain is defined as: $\frac{\tau_r}{\tau_r + T_s}$ where: $\tau_r = \frac{L_r}{R_r}$ The parameter is within the range <0 ; 1.0). Set by the user.
f32KPsiRB1Gain		frac32_t	The coefficient gain is defined as: $\frac{L_m \cdot T_s}{\tau_r} \cdot \frac{i_{max}}{u_{max}} where : \tau_r = \frac{L_r}{R_r}$ The parameter is within the range <0 ; 1.0). Set by the user.
f32KPsiSA1Gain		frac32_t	The gain is defined as: $\frac{1}{1 + T_s \cdot 2\pi \cdot f_{integ}}$

Table continues on the next page...

Variable name	Data type	Description
		The f_{integ} is a cut-off frequency of a low-pass filter approximation of a pure integrator. The parameter is within the range <0 ; 1.0). Set by the user.
f32KPsiSA2Gain	frac32_t	The coefficient gain is defined as: $\frac{T_s}{1+T_s \cdot 2\pi \cdot f_{integ}}$ The f_{integ} is a cut-off frequency of a low-pass filter approximation of a pure integrator. The parameter is within the range <0 ; 1.0). Set by the user.
a32KrInvGain	acc32_t	The gain is defined as: $\frac{L_r}{L_m}$ The parameter is within the range <0 ; 65536.0). Set by the user.
a32KrLsTotLeakGain	acc32_t	The coefficient gain is defined as: $\frac{L_s \cdot L_r - \cdot L_m^2}{L_m} \cdot \frac{I_{max}}{U_{max}}$ The parameter is within the range <0 ; 65536.0). Set by the user.
a32TorqueGain	acc32_t	The torque constant coefficient gain is defined as: $\frac{3 \cdot P_P \cdot L_m}{2 \cdot L_r \cdot I_{max}}$ The P_P is a number of motor pole-pairs. The parameter is within the range <0 ; 65536.0). Set by the user.
f16Torque	frac16_t	The output estimated motor torque calculated as: $T = \frac{3 \cdot P_P \cdot L_m \cdot (\Psi_{r\alpha} \cdot I_{s\beta} - \Psi_{r\beta} \cdot I_{s\alpha})}{2 \cdot I_{max}}$ The result is within the range <-1 ; 1.0). Calculated by the algorithm.
f16KRsEst	frac16_t	The stator resistance parameter calculated as: $R_S \cdot \frac{I_{max}}{U_{max}}$ The parameter is within the range <0 ; 65536.0). Set by the user.
f16RotFluxPos	frac16_t	The output rotor flux estimated electric position (angle) - a 16-bit fractional type is normalized to the range <-1 ; 1) that represents an angle (in radians) within the range <-π ; π).

2.2.3 Declaration

The available AMCLIB_ACIMRotFluxObsrvInit function has the following declarations:

```
void AMCLIB_ACIMRotFluxObsrvInit_F16(AMCLIB_ACIM_ROT_FLUX_OBSRV_T_A32 *psCtrl)
```

The available AMCLIB_ACIMRotFluxObsrv function has the following declarations:

```
void AMCLIB_ACIMRotFluxObsrv_F16(const GMCLIB_2COOR_ALBE_T_F16 *psISAlBe, const
GMCLIB_2COOR_ALBE_T_F16 *psUSAlBe, AMCLIB_ACIM_ROT_FLUX_OBSRV_T_A32 *psCtrl)
```

2.2.4 Function use

The use of the AMCLIB_ACIMRotFluxObsrv function is shown in the following examples:

Fixed-point version:

```
#include "amclib."

static GMCLIB_2COOR_ALBE_T_F16 sIsAlBe, sUsAlBe;
static AMCLIB_ACIM_ROT_FLUX_OBSRV_T_A32 sRfoParam;

void Isr(void);

void main (void)
{
    sRfoParam.sCtrl.a32PGain     = ACC32(25.0);;
    sRfoParam.sCtrl.a32IGain     = ACC32(0.01);;
    sRfoParam.a32KrInvGain       = ACC32(1.096509240246);;
    sRfoParam.a32KrLsTotLeakGain = ACC32(0.003153149897);;
    sRfoParam.f32KPsiRA1Gain     = FRAC32(0.031726651724);;
    sRfoParam.f32KPsiRB1Gain     = FRAC32(0.004160019072);;
    sRfoParam.f32KPsiSA1Gain     = FRAC32(0.998744940093);;
    sRfoParam.f32KPsiSA2Gain     = FRAC32(0.000199748988);;
    sRfoParam.f16KRsEst          = FRAC16(0.807136);;

    /* Initialization of the RFO's structure */
    AMCLIB_ACIMRotFluxObsrvInit_F16 (&sRfoParam);

    sIsAlBe.f32Alpha = FRAC16(0.05);;
    sIsAlBe.f32Beta  = FRAC16(0.1);;
    sUsAlBe.f32Alpha = FRAC16(0.2);;
    sUsAlBe.f32Beta  = FRAC16(-0.1);;
}

/* Periodical function or interrupt */
void Isr(void)
{
    /* Rotor flux observer calculation */
    AMCLIB_ACIMRotFluxObsrv_F16(&sIsAlBe, &sUsAlBe, &sRfoParam);
}
```

2.3 AMCLIB_ACIMSpeedMRAS

The AMCLIB_ACIMSpeedMRAS function is based on the model reference approach (MRAS), and it uses the redundancy of two machine models of different structures that estimate the same state variable based on different sets of input variables. It means that the rotor speed can obtained using an estimator with MRAS principle, in which the error vector is formed from the outputs of two models (both dependent on different motor parameters) - as shown in Figure 2-3.

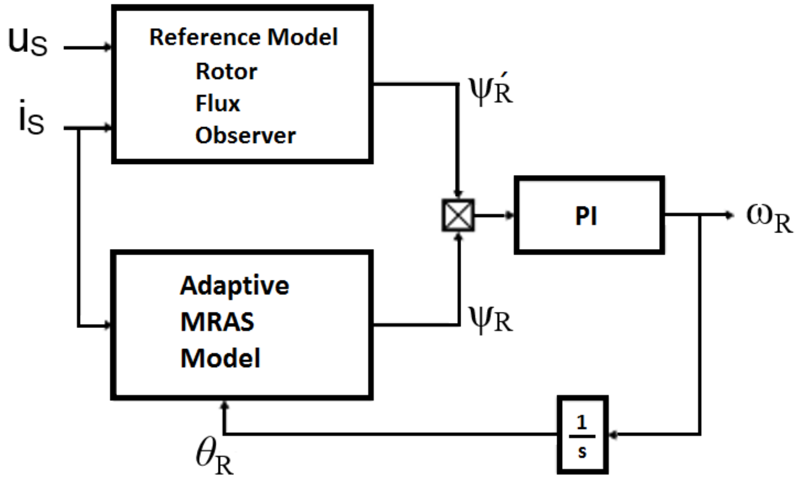


Figure 2-3. The estimated and real rotor *dq* synchronous reference frames

The closed-loop flux observer provides a stationary-axis-based rotor flux Ψ_R from RFO as a reference for the MRAS model, whereas the adaptive model of MRAS is the current-mode flux observer, which provides adjustable stationary-axis-based rotor flux:

$$\frac{d\overrightarrow{\psi_r^{MRAS}}}{dt} = -\frac{1}{\tau_r} \cdot \overrightarrow{\psi_r^{MRAS}} + \frac{L_m}{\tau_r}\overrightarrow{i_s}$$

Equation 9

where:

- i_s is the stator current vector
- Ψ_r is the rotor flux-linkage vector
- ω_r is the rotor electrical angular speed
- τ_r is the rotor electrical time constant
- L_m is the mutual equivalent inductance

The phase angle between the two estimated rotor flux vectors is used to correct the adaptive model, according to:

$$e_{MRAS} = \overrightarrow{\psi_{r\alpha}^{RFO}} \cdot \overrightarrow{\psi_{r\beta}^{MRAS}} - \overrightarrow{\psi_{r\beta}^{RFO}} \cdot \overrightarrow{\psi_{r\alpha}^{MRAS}}$$

Equation 10

The estimated speed ω_R is adjusted by a PI regulator.

2.3.1 Available versions

The available versions of the AMCLIB_ACIMSpeedMRAS function are shown in the following table:

Table 2-5. Init version

Function name	Parameters	Result type
AMCLIB_ACIMSpeedMRASInit_F16	AMCLIB_ACIM_SPEED_MRAS_T_F32 *	void
	The initialization does not have an input.	

Table 2-6. Function version

Function name	Input/output type		Result type
AMCLIB_ACIMSpeedMRAS_F16	**Input**	GMCLIB_2COOR_ALBE_T_F16 *	void
		GMCLIB_2COOR_ALBE_T_F32 *	
		frac16_t	
	Parameters	AMCLIB_ACIMSpeedMRAS_T_F32 *	
	The AMCLIB_ACIMSpeedMRAS_F16 function with a 16-bit and 32-bit fractional type inputs: stator current and voltage in alpha-beta coordinates.		

2.3.2 AMCLIB_ACIM_SPEED_MRAS_T_F32 type description

Variable name	Data type	Description
sSpeedElIIR1Param	GDFLIB_FILTER_IIR1_T_F32	The IIR1 filter structure for estimated speed filtration. Set by the user.
sPsiRotRDQ	GMCLIB_2COOR_DQ_T_F32	The output rotor flux estimated structure from the current model. The structure consists of the D and Q rotor flux components stored for the next step by the algorithm.
sSpeedInteg	GFLIB_INTEGRATOR_T_A32	The speed integral part - state variable at step k-1 of the electrical speed controller.
f32KPsiRA1Gain	frac32_t	The coefficient gain is defined as: $\frac{\tau_r}{\tau_r + T_s}$ where: $\tau_r = \frac{L_r}{R_r}$ The parameter is within the range <0 ; 1.0). Set by the user.
f32KPsiRB1Gain	frac32_t	The coefficient gain is defined as: $\frac{L_m \cdot T_s}{\tau_r} \cdot \frac{i_{max}}{u_{max}}$ where : $\tau_r = \frac{L_r}{R_r}$ The parameter is within the range <0 ; 1.0). Set by the user.
f32KImaxGain	frac32_t	Constant determined by: 1/i_max. The parameter is within the range <0 ; 1.0). Set by the user.
f32Error	frac32_t	The output error variable defined as:

Table continues on the next page...

Variable name		Data type	Description
			$e_{MRAS} = \overrightarrow{\psi_{r\alpha}^{RFO}} \cdot \overrightarrow{\psi_{r\beta}^{MRAS}} - \overrightarrow{\psi_{r\beta}^{RFO}} \cdot \overrightarrow{\psi_{r\alpha}^{MRAS}}$ The result is within the range <-1 ; 1.0).
f32Ts		frac32_t	The sample time constant - the time between the steps. The parameter is within the range (0 ; 1.0). Set by the user.
f16RotPos		frac16_t	The output rotor estimated electric position (angle) - a 32-bit accumulator is normalized to the range <-1 ; 1) that represents an angle (in radians) within the range <-π ; π).
f16SpeedEl		frac16_t	Rotor estimated electric speed, the output variable within the range <-1 ; 1.0).
f16SpeedElIIR1		frac16_t	The output rotor estimated electrical speed filtered. The result is within the range <-1 ; 1.0). Calculated by the algorithm.
sCtrl	f32SpeedElInteg_1	frac32_t	The speed integral part - state variable at step k-1 of the electrical speed controller. Calculated by the algorithm for next steps.
	f32SpeedElErr_1	frac32_t	The speed error - state variable at step k-1 of the electrical speed controller. Calculated by the algorithm for next steps.
	a32PGain	acc32_t	The MRAS proportional gain coefficient. The parameter is within the range <0 ; 65536.0). Set by the user.
	a32IGain	acc32_t	The MRAS integral gain coefficient. The parameter is within the range <0 ; 65536.0). Set by the user.

2.3.3 Declaration

The available AMCLIB_ACIMSpeedMRASInit function have the following declarations:

```
void AMCLIB_ACIMSpeedMRASInit_F16(AMCLIB_ACIM_SPEED_MRAS_T_F32 *psCtrl)
```

The available AMCLIB_ACIMSpeedMRAS function have the following declarations:

```
void AMCLIB_ACIMSpeedMRAS_F16(const GMCLIB_2COOR_ALBE_T_F16 *psISAlBe, const
GMCLIB_2COOR_ALBE_T_F32 *psPsiRAlBe, frac16_t f16RotPos, AMCLIB_ACIM_SPEED_MRAS_T_F32
*psCtrl)
```

2.3.4 Function use

The use of the AMCLIB_ACIMSpeedMRAS function is shown in the following examples:

Fixed-point version:

```
#include "amclib.h"

static GMCLIB_2COOR_ALBE_T_F16 sIsAlBe, sPsiRAlBe;
static AMCLIB_ACIM_SPEED_MRAS_T_F32 sMrasParam;
static frac16_t f16RotPosIn;

void Isr(void);

void main (void)
{
    sMrasParam.sCtrl.a32PGain  = ACC32(32750.0);;
    sMrasParam.sCtrl.a32IGain  = ACC32(12500.0);;
    sMrasParam.f32KPsiRA1Gain   = FRAC32(0.9914578663826716);;
    sMrasParam.f32KPsiRB1Gain   = FRAC32(0.004160019071638958);;
    sMrasParam.f32Ts           = FRAC32(0.0001);;

    /* Initialization of the MRAS's structure */
    AMCLIB_ACIMSpeedMRASInit_F16 (&sMrasParam);

    sIsAlBe.f16Alpha   = FRAC16(0.05);;
    sIsAlBe.f16Beta    = FRAC16(0.1);;
    sPsiRAlBe.f16Alpha = FRAC16(0.2);;
    sPsiRAlBe.f16Beta  = FRAC16(-0.1);;
}

/* Periodical function or interrupt */
void Isr(void)
{
    /* Speed estimation calculation based on MRAS */
    AMCLIB_ACIMSpeedMRAS_F16(&sIsAlBe, &sPsiRAlBe, f16RotPosIn, &sMrasParam);
}
```

2.4 AMCLIB_AngleTrackObsrv

The AMCLIB_TrackObsrv function calculates an angle-tracking observer for determination of angular speed and position of the input signal. 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(θ), cos(θ) with their corresponding estimations. As in any common closed-loop systems, the intent is to minimize the observer error towards zero value. The observer error is given here by subtracting the estimated resolver rotor angle from the actual rotor angle.

The tracking-observer algorithm uses the phase-locked loop mechanism. It is recommended to call this function at every sampling period. It requires a single input argument as phase error. A phase-tracking observer with standard PI controller used as the loop compensator is shown in Figure 2-4.

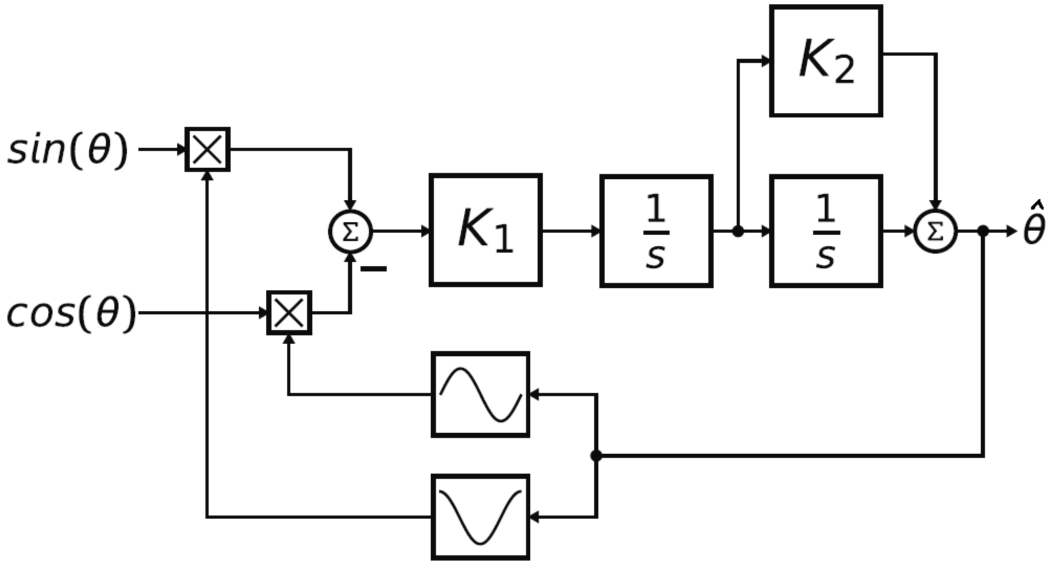


Figure 2-4. Block diagram of proposed PLL scheme for position estimation

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

$$\sin(\theta - \hat{\theta}) = \sin(\theta) \cdot \cos(\hat{\theta}) - \cos(\theta) \cdot \sin(\hat{\theta})$$

Equation 11

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

$$\sin(\theta - \hat{\theta}) \approx \theta - \hat{\theta}$$

Equation 12

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 the proportional and integral controllers, which are connected in series and closed by a unit feedback loop. This block diagram tracks the actual rotor angle and speed, and continuously updates their estimations. The angle-tracking observer transfer function is expressed as follows:

$$\frac{\hat{\theta}(s)}{\theta(s)} = \frac{K_1(1 + sK_2)}{s^2 + sK_1K_2 + K_1}$$

Equation 13

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

$$s^2 + sK_1K_2 + K_1$$

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

The analog integrators in the previous figure (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 the following figure:

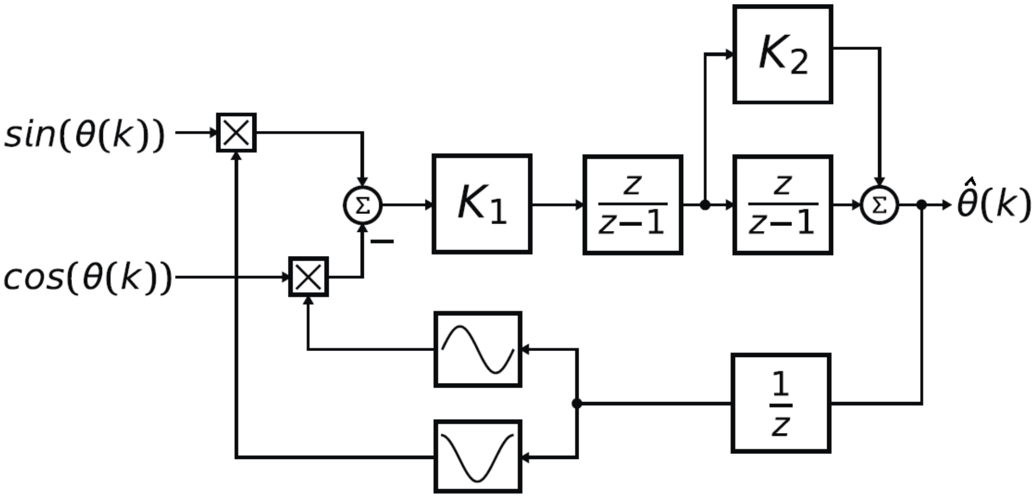


Figure 2-5. Block scheme of discrete-time tracking observer

The essential equations for implementating the angle-tracking observer (according to this block scheme) are as follows:

$$e(k) = \sin(\theta(k)) \cdot \cos(\hat{\theta}(k-1)) - \cos(\theta(k)) \cdot \sin(\hat{\theta}(k-1))$$

Equation 14

$$\omega(k) = T_s \bullet K_1 \bullet e(k) + \omega(k-1)$$

Equation 15

$$a_2(k) = T_s \bullet \omega(k) + a_2(k-1)$$

Equation 16

$$\hat{\theta}(k) = K_2 \cdot \omega(k) + a_2(k)$$

Equation 17

where:

- K_1 is the integral gain of the I controller
- K_2 is the proportional gain of the PI controller

- T_s is the sampling period [s]
- $e(k)$ is the position error in step k
- $\omega(k)$ is the rotor speed [rad / s] in step k
- $\omega(k - 1)$ is the rotor speed [rad / s] in step k - 1
- $a(k)$ is the integral output of the PI controler [rad / s] in step k
- $a(k - 1)$ is the integral output of the PI controler [rad / s] in step k - 1
- $\theta(k)$ is the rotor angle [rad] in step k
- $\theta(k - 1)$ is the rotor angle [rad] in step k - 1
- $\theta(k)$ is the estimated rotor angle [rad] in step k
- $\theta(k - 1)$ is the estimated rotor angle [rad] in step k - 1

In the fractional arithmetic, Equation 14 on page 31 to Equation 17 on page 31 are as follows:

$$\omega_{sc}(k) \bullet \omega_{max} = T_s \bullet K_1 \bullet e(k) + \omega_{sc}(k-1) \bullet \omega_{max}$$

Equation 18

$$a_{2sc}(k) \bullet \theta_{max} = T_s \bullet \omega_{sc}(k) \bullet \omega_{max} + a_{2sc}(k-1) \bullet \theta_{max}$$

Equation 19

$$\hat{\theta}_{sc}(k) \bullet \theta_{max} = K_2 \bullet \omega_{sc}(k) \bullet \omega_{max} + a_{2sc}(k) \bullet \theta_{max}$$

Equation 20

where:

- $e_{sc}(k)$ is the scaled position error in step k
- $\omega_{sc}(k)$ is the scaled rotor speed [rad / s] in step k
- $\omega_{sc}(k - 1)$ is the scaled rotor speed [rad / s] in step k - 1
- $a_{sc}(k)$ is the integral output of the PI controler [rad / s] in step k
- $a_{sc}(k - 1)$ is the integral output of the PI controler [rad / s] in step k - 1
- $\theta_{sc}(k)$ is the scaled rotor angle [rad] in step k
- $\theta_{sc}(k - 1)$ is the scaled rotor angle [rad] in step k - 1
- $\theta_{sc}(k)$ is the scaled rotor angle [rad] in step k
- $\theta_{sc}(k - 1)$ is the scaled rotor angle [rad] in step k - 1
- ω_{max} is the maximum speed
- θ_{max} is the maximum rotor angle (typicaly π)

2.4.1 Available versions

The function is available in the following versions:

- Fractional output - the output is the fractional portion of the result; the result is within the range <-1 ; 1).

The available versions of the AMCLIB_AngleTrackObsrv function are shown in the following table:

Table 2-7. Init versions

Function name	Init angle	Parameters	Result type
AMCLIB_AngleTrackObsrvInit_F16	frac16_t	AMCLIB_ANGLE_TRACK_OBSRV_T_F32 *	void
	The input is a 16-bit fractional value of the angle normalized to the range <-1 ; 1) that represents an angle in (radians) within the range <-π ; π).		

Table 2-8. Function versions

Function name	Input type	Parameters	Result type
AMCLIB_AngleTrackObsrv_F16	GMCLIB_2COOR_SINCOS_T_F16 *	AMCLIB_ANGLE_TRACK_OBSRV_T_F32 *	frac16_t
	Angle-tracking observer with a two-componenent (sin/cos) 16-bit fractional position input within the range <-1 ; 1). The output from the observer is a 16-bit fractional position normalized to the range <-1 ; 1) that represents an angle (in radians) within the range <-π ; π).		

2.4.2 AMCLIB_ANGLE_TRACK_OBSRV_T_F32

Variable name	Input type	Description
f32Speed	frac32_t	Estimated speed as the output of the first numerical integrator. The parameter is within the range <-1 ; 1). Controlled by the AMCLIB_AngleTrackObsrv_F16 algorithm; cleared by the AMCLIB_AngleTrackObsrvInit_F16 function.
f32A2	frac32_t	Output of the second numerical integrator. The parameter is within the range <-1 ; 1). Controlled by the AMCLIB_AngleTrackObsrv_F16 and AMCLIB_AngleTrackObsrvInit_F16 algorithms.
f16Theta	frac16_t	Estimated position as the output of the observer. The parameter is normalized to the range <-1 ; 1) that represents an angle (in radians) within the range <-π ; π). Controlled by the AMCLIB_AngleTrackObsrv_F16 and AMCLIB_AngleTrackObsrvInit_F16 algorithms.
f16SinEstim	frac16_t	Sine of the estimated position as the output of the actual step. Keeps the sine of the position for the next step. The parameter is within the range <-1 ; 1). Controlled by the AMCLIB_AngleTrackObsrv_F16 and AMCLIB_AngleTrackObsrvInit_F16 algorithms.
f16CosEstim	frac16_t	Cosine of the estimated position as the output of the actual step. Keeps the cosine of the position for the next step. The parameter is within the range <-1 ; 1). Controlled by the AMCLIB_AngleTrackObsrv_F16 and AMCLIB_AngleTrackObsrvInit_F16 algorithms.
f16K1Gain	frac16_t	Observer K1 gain is set up according to Equation 18 on page 32 as:

Table continues on the next page...

Variable name	Input type	Description
		$T_s \cdot K_1 \cdot \frac{1}{\omega_{max}} \cdot 2^{-K1sh}$ The parameter is a 16-bit fractional type within the range <0 ; 1). Set by the user.
i16K1GainSh	int16_t	Observer K2 gain shift takes care of keeping the f16K1Gain variable within the fractional range <-1 ; 1). The shift is determined as: $\log_2(T_s \cdot K_1 \cdot \frac{1}{\omega_{max}}) - \log_2 1 < K1sh \le \log_2(T_s \cdot K_1 \cdot \frac{1}{\omega_{max}}) - \log_2 0.5$ The parameter is a 16-bit integer type within the range <-15 ; 15>. Set by the user.
f16K2Gain	frac16_t	Observer K2 gain is set up according to Equation 20 on page 32 as: $K_2 \cdot \frac{\omega_{max}}{\theta_{max}} \cdot 2^{-K2sh}$ The parameter is a 16-bit fractional type within the range <0 ; 1). Set by the user.
i16K2GainSh	int16_t	Observer K2 gain shift takes care of keeping the f16K2Gain variable within the fractional range <-1 ; 1). The shift is determined as: $\log_2(K_2 \cdot \frac{\omega_{max}}{\theta_{max}}) - \log_2 1 < K2sh \le \log_2(K_2 \cdot \frac{\omega_{max}}{\theta_{max}}) - \log_2 0.5$ The parameter is a 16-bit integer type within the range <-15 ; 15>. Set by the user.
f16A2Gain	frac16_t	Observer A2 gain for the output position is set up according to Equation 19 on page 32 as: $T_s \cdot \frac{\omega_{max}}{\theta_{max}} \cdot 2^{-A2sh}$ The parameter is a 16-bit fractional type within the range <0 ; 1). Set by the user.
i16A2GainSh	int16_t	Observer A2 gain shift for the position integrator takes care of keeping the f16A2Gain variable within the fractional range <-1 ; 1). The shift is determined as: $\log_2(T_s \cdot \frac{\omega_{max}}{\theta_{max}}) - \log_2 1 < A2sh \le \log_2(T_s \cdot \frac{\omega_{max}}{\theta_{max}}) - \log_2 0.5$ The parameter is a 16-bit integer type within the range <-15 ; 15>. Set by the user.

2.4.3 Declaration

The available AMCLIB_AngleTrackObsrvInit functions have the following declarations:

```
void AMCLIB_AngleTrackObsrvInit_F16(frac16_t f16ThetaInit, AMCLIB_ANGLE_TRACK_OBSRV_T_F32
*psCtrl)
```

The available AMCLIB_AngleTrackObsrv functions have the following declarations:

```
frac16_t AMCLIB_AngleTrackObsrv_F16(const GMCLIB_2COOR_SINCOS_T_F16 *psAnglePos,
AMCLIB_ANGLE_TRACK_OBSRV_T_F32 *psCtrl)
```

2.4.4 Function use

The use of the AMCLIB_AngleTrackObsrvInit and AMCLIB_AngleTrackObsrv functions is shown in the following example:

```
#include "amclib.h"

static AMCLIB_ANGLE_TRACK_OBSRV_T_F32  sAto;
static GMCLIB_2COOR_SINCOS_T_F16 sAnglePos;
static frac16_t       f16PositionEstim, f16PositionInit;

void Isr(void);

void main(void)
{
  sAto.f16K1Gain     = FRAC16(0.6434);
  sAto.i16K1GainSh   = -9;
  sAto.f16K2Gain     = FRAC16(0.6801);
  sAto.i16K2GainSh   = -2;
  sAto.f16A2Gain     = FRAC16(0.6400);
  sAto.i16A2GainSh   = -4;

  f16PositionInit = FRAC16(0.0);

  AMCLIB_AngleTrackObsrvInit_F16(f16PositionInit, &sAto);

  sAnglePos.f16Sin   = FRAC16(0.0);
  sAnglePos.f16Cos   = FRAC16(1.0);
}

/* Periodical function or interrupt */
void Isr(void)
{
  /* Angle tracking observer calculation */
  f16PositionEstim = AMCLIB_AngleTrackObsrv_F16(&sAnglePos, &sAto);
}
```

2.5 AMCLIB_CtrlFluxWkng

The AMCLIB_CtrlFluxWkng function controls the motor magnetizing flux for a speed exceeding above the nominal speed of the motor. Where a higher maximum motor speed is required, the flux (field) weakening technique must be used. The basic task of the function is to maintain the motor magnetizing flux below the nominal level which does not require a higher supply voltage when the motor rotates above the nominal motor speed. The lower magnetizing flux is provided by maintaining the flux-producing current component i_D in the flux-weakening region, as shown in Figure 2-6).

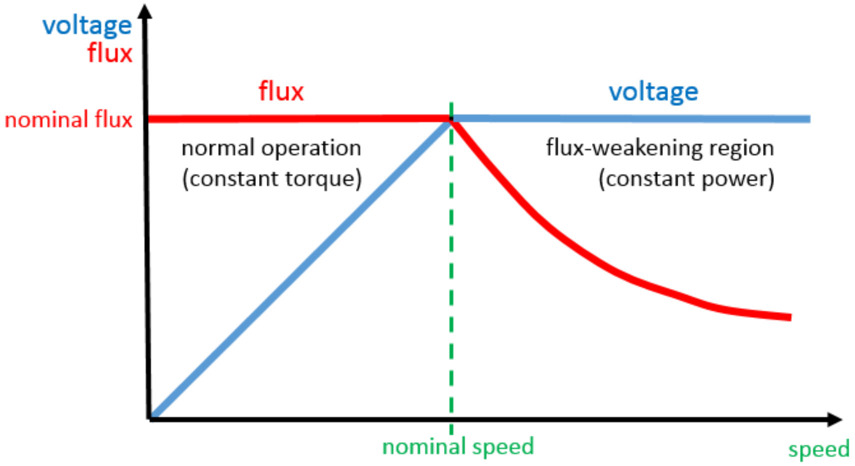


Figure 2-6. Flux weakening operating range

The AMCLIB_CtrlFluxWkng function processes the magnetizing flux by the PI controller function with the anti-windup functionality and output limitation. The controller integration can be stopped if the system is saturated by the input flag pointer in the flux-weakening controller structure. The flux-weakening controller algorithm is executed in the following steps:

1. The voltage error calculation from the voltage limit and the required voltage.

$$u_{err} = (u_{QLim} - |u_{Qreq}|) \cdot \frac{I_{gain}}{U_{gain}}$$

Equation 21.

   where:

   - u_{err} is the voltage error
   - u_{QLim} is the Q voltage limit component
   - u_{Qreq} is the Q required voltage component
   - I_{gain} is the voltage scale - max. value (for fraction gain = 1)
   - U_{gain} is the current scale - max. value (for fraction gain = 1)
2. The input Q current error component must be positive and filtered by the infinite impulse response first-order filter.

$$i_{QerrIIR} = IIR1(|i_{Qerr}|)$$

Equation 22.

   where:

   - $i_{QerrIIR}$ is the Q current error component filtered by the first-order IIR
   - i_{Qerr} is the input Q current error component (calculated before calling the AMCLIB_CtrlFluxWkng function from the measured and limited required Q current component value).

3. The flux error is obtained from the previously calculated voltage and current errors as follows:

$$i_{err} = i_{QerrIIR} - u_{err}$$

Equation 23.

   where:
   - i_{err} is the Q current error component for the flux PI controller
   - $i_{QerrIIR}$ is the current error component filtered by the first-order IIR
   - u_{err} is the voltage error for the flux PI controller
4. Finally, the flux error (corresponding the I_D) is processed by the flux PI controller:

$$i_{Dreq} = CtrlPIpAW(i_{err})$$

Equation 24.

   where:
   - i_{Dreq} is the required D current component for the current control
   - i_{err} is the flux error (corresponding the D current component) for the flux PI controller

The controller output should be used as the required D current component in the fast control loop and concurrently used as an input for the GFLIB_VectorLimit1 function which limits the I_Q controller as follows:

$$i_{Qreq} \leq \sqrt{i_{max}^2 - i_{Dreq}^2}$$

Equation 25.

where:

- i_{Qreq} is the required Q current component for the current control
- i_{max} is application current limit
- i_{Dreq} is the required D current component for the current control

The following figure shows an example of applying the flux-weakening controller function in the control structure. The flux controller starts to operate when the I_Q controller is not able to compensate the $I_{Q\ err}$ and creates a deviation between its input and ouput. The flux controller processes the deviation and decreases the flux excititation (for ACIM, or starts to create the flux extitation against a permanent magnet flux in case of PMSM). A lower BEMF causes a higher I_Q and the motor speed increases. The speed controller with $I_{Q\ reg}$ on the output should be limited by the vector limit1 function because a part of the current is used for flux excitation.

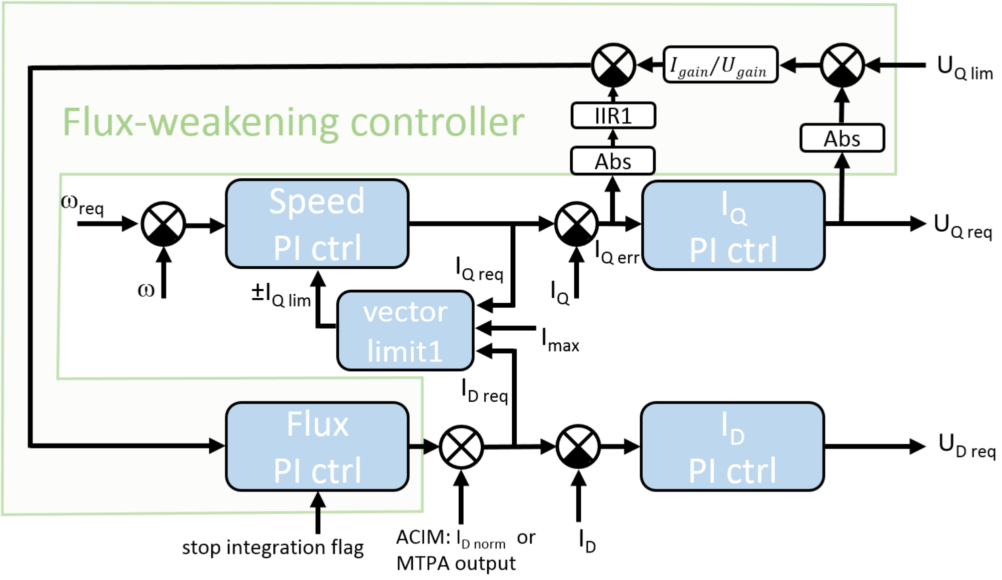


Figure 2-7. Flux weakening function in control block structure

2.5.1 Available versions

This function is available in the following versions:

- Fractional output - the output is the fractional portion of the result; the result is within the range <-1 ; 1) in case of no limitation. The parameters are of fractional or accumulator types.

The available versions of the AMCLIB_CtrlFluxWkngInit function are shown in the following table:

Table 2-9. Init function versions

Function name	Input type	Parameters	Result type
AMCLIB_CtrlFluxWkngInit_F16	frac16_t	AMCLIB_CtrlFluxWkngInit_A32 *	void
	The inputs are a 16-bit fractional initial value for the flux PI controller integrating the part state and a pointer to the flux-weakening controller's parameters structure. The function initializes the flux PI controller and the IIR1 filter.		

The available versions of the AMCLIB_CtrlFluxWkng function are shown in the following table:

Table 2-10. Function versions

Function name	Input type			Parameters	Result type
	Q current error	Q required voltage	Q voltage limit		
AMCLIB_CtrlFluxWkng_F16	frac16_t	frac16_t	frac16_t	AMCLIB_CTRL_FLUX_WKNG_T_A32 *	frac16_t
	The Q current error component value input (I_Q controller input) and the Q required voltage value input (I_Q controller output) are 16-bit fractional values within the range <-1 ; 1). The Q voltage limit value input (constant value) is a 16-bit fractional value within the range (0 ; 1). The parameters are pointed to by an input pointer. The function returns a 16-bit fractional value in the range <f16LowerLim ; f16UpperLim>.				

2.5.2 AMCLIB_CTRL_FLUX_WKNG_T_A32

Variable name	Input type	Description
sFWPiParam	GFLIB_CTRL_PI_P_AW_T_A32	The input pointer for the flux PI controller parameter structure. The flux controller output should be negative. Therefore, set at least the following parameters: • a32PGain - proportional gain, the range is <0 ; 65536.0). • a32IGain - integral gain, the range is <0 ; 65536.0). • f16UpperLim - upper limit, the zero value should be set. • f16LowerLim - the lower limit, the range is <-1; 0).
sIqErrIIR1Param	GDFLIB_FILTER_IIR1_T_F32	The input pointer for the IIR1 filter parameter structure. The IIR1 filters the absolute value of the Q current error component for the flux controller. Set at least the following parameters: • sFltCoeff.f32B0 - B0 coefficient, must be divided by 2. • sFltCoeff.f32B1 - B1 coefficient, must be divided by 2. • sFltCoeff.f32A1 - A1 (sign-inverted) coefficient, must be divided by -2 (negative two).
f16IqErrIIR1	frac32_t	The I_Q current error component,filtered by the IIR1 filter for the flux PI controller, as shown in Equation 22 on page 36. The output value calculated by the algorithm.
f16UFWErr	frac16_t	The voltage error, as shown in Equation 21 on page 36. The output value calculated by the algorithm.
f16FWErr	frac16_t	The flux-weakening error, as shown in Equation 23 on page 37. The output value calculated by the algorithm.
*bStopIntegFlag	frac16_t	The integration of the PI controller is suspended if the stop flag is set. When it is cleared, the integration continues. The pointer is set by the user and controlled by the application.

2.5.3 Declaration

The available AMCLIB_CtrlFluxWkngInit functions have the following declarations:

```
void AMCLIB_CtrlFluxWkngInit_F16(frac16_t f16InitVal, AMCLIB_CTRL_FLUX_WKNG_T_A32 *psParam)
```

The available AMCLIB_CtrlFluxWkng functions have the following declarations:

```
frac16_t AMCLIB_CtrlFluxWkng_F16(frac16_t f16IQErr, frac16_t f16UQReq, frac16_t f16UQLim,
AMCLIB_CTRL_FLUX_WKNG_T_A32 *psParam)
```

2.5.4 Function use

The use of the AMCLIB_CtrlFluxWkngInit and AMCLIB_CtrlFluxWkng functions is shown in the following examples:

Fixed-point version:

```
#include "amclib.h"

static AMCLIB_CTRL_FLUX_WKNG_T_A32 sCtrl;
static frac16_t f16IQErr, f16UQReq, f16UQLim;
static frac16_t f16IdReq, f16InitVal;
static bool_t bStopIntegFlag;

void Isr(void);

void main(void)
{
    /* Associate input stop integration flag */
    bStopIntegFlag = FALSE;
    sCtrl.bStopIntegFlag = &bStopIntegFlag;

    /* Set PI controller and IIR1 parameters */
    sCtrl.sFWPiParam.a32PGain = ACC32(0.1);
    sCtrl.sFWPiParam.a32IGain = ACC32(0.2);
    sCtrl.sFWPiParam.f16UpperLim = FRAC16(0.);
    sCtrl.sFWPiParam.f16LowerLim = FRAC16(-0.9);
    sCtrl.sIqErrII1Param.sFltCoeff.f32B0 = FRAC32(0.245237275252786 / 2.0);
    sCtrl.sIqErrII1Param.sFltCoeff.f32B1 = FRAC32(0.245237275252786 / 2.0);
    sCtrl.sIqErrII1Param.sFltCoeff.f32A1 = FRAC32(-0.509525449494429 / -2.0);

    /* Flux weakening controller initialization */
    f16InitVal = FRAC16(0.0);
    AMCLIB_CtrlFluxWkngInit_F16(f16InitVal, &sCtrl);

    /* Assign input variable */
    f16IQErr = FRAC16(-0.1);
    f16UQReq = FRAC16(-0.2);
    f16UQLim = FRAC16(0.8);
}

/* Periodical function or interrupt */
void Isr()
{
    /* Flux weakening controller calculation */
    f16Result = AMCLIB_CtrlFluxWkng_F16(f16IQErr, f16UQReq, f16UQLim, &sCtrl);
}
```

2.6 AMCLIB_PMSMBemfObsrvAB

The AMCLIB_PMSMBemfObsrvAB function calculates the algorithm of the back-electro-motive force (back-EMF) observer in a stationary reference frame. The estimation method for the rotor position and the angular speed is based on the mathematical model of an interior PMSM motor with an extended electro-motive force function, which is realized in the alpha/beta stationary reference frame.

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

$$\begin{bmatrix} u_\alpha \\ u_\beta \end{bmatrix} = \begin{bmatrix} R_S + sL_D & \omega_r \Delta L \\ -\omega_r \Delta L & R_S + sL_D \end{bmatrix} \bullet \begin{bmatrix} i_\alpha \\ i_\beta \end{bmatrix} + \left[\Delta L \bullet \left(\omega_r i_D - s i_Q\right) + \Psi_m \omega_r\right] \bullet \begin{bmatrix} -\sin(\theta_r) \\ \cos(\theta_r) \end{bmatrix}$$

Equation 26

Where:

- R_S is the stator resistance
- L_D and L_Q are the D-axis and Q-axis inductances
- $\Delta L = L_D - L_Q$ is the motor saliency
- Ψ_m is the back-EMF constant
- ω_r is the angular electrical rotor speed
- u_α and u_β are the estimated stator voltages
- i_α and i_β are the estimated stator currents
- θ_r is the estimated rotor electrical position
- s is the operator of the derivative

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

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

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

The block diagram of the observer in the estimated reference frame is shown in Figure 2-8. The observer compensator is substituted by a standard PI controller with following equation in the fractional arithmetic.

$$i_{sc}(k) \cdot i_{max} = K_P \cdot e_{sc}(k) \cdot e_{max} + T_s \cdot K_I \cdot e_{sc}(k) \cdot e_{max} + i_{sc}(k-1) \cdot i_{max}$$

Equation 27

where:

- K_P is the observer proportional gain [-]
- K_I is the observer integral gain [-]
- $i_{sc}(k) = [i_\gamma, i_\delta]$ is the scaled stator current vector in the actual step
- $i_{sc}(k - 1) = [i_\gamma, i_\delta]$ is the scaled stator current vector in the previous step
- $e_{sc}(k) = [e_\gamma, e_\delta]$ is the scaled stator back-EMF voltage vector in the actual step
- i_{max} is the maximum current [A]
- e_{max} is the maximum back-EMF voltage [V]
- T_S is the sampling time [s]

As shown in Figure 2-8, the observer model and hence also the PI controller gains in both axes are identical to each other.

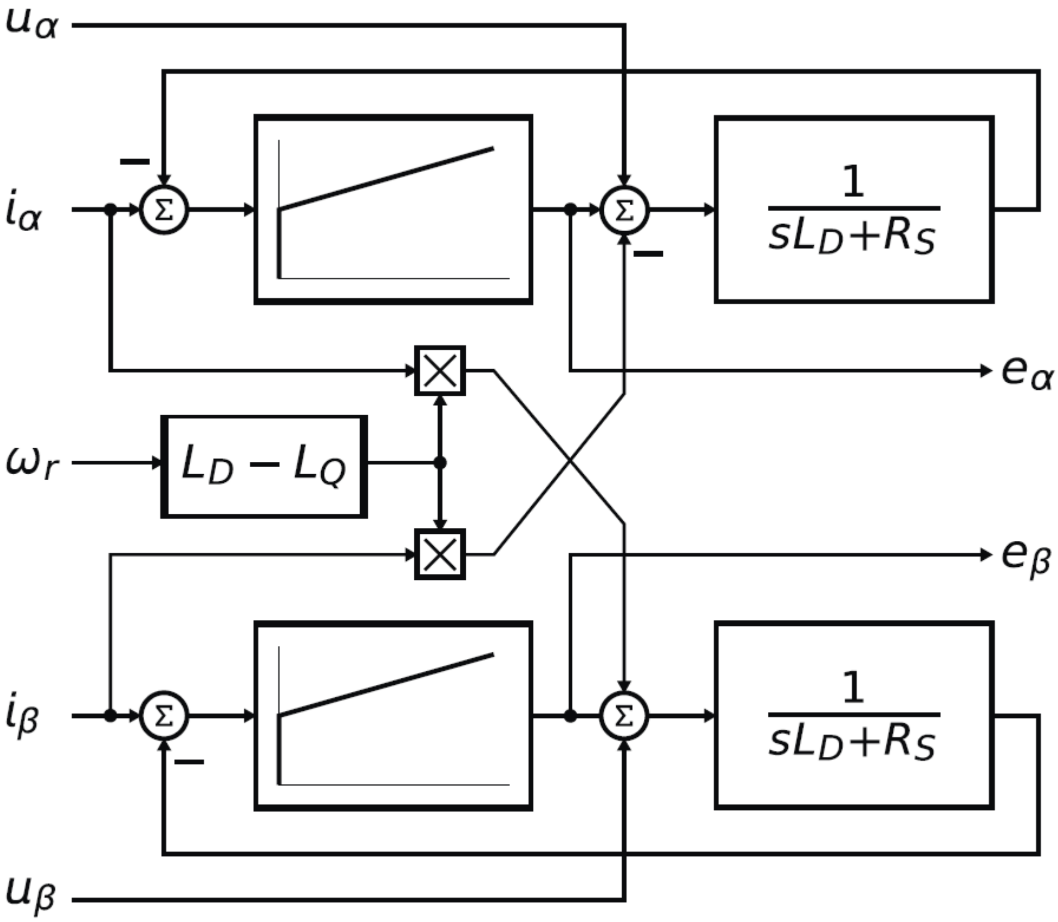


Figure 2-8. Block diagram of back-EMF observer

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

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

$$\hat{E}_{\alpha\beta}(s) = -E_{\alpha\beta}(s) \cdot \frac{F_C(s)}{sL_D + R_S + F_C(s)}$$

Equation 28

The back-EMF observer is a Luenberger-type observer with a motor model, which is implemented using the backward Euler transformation as:

$$i(k) = \frac{T_s}{L_D + T_s R_S} \cdot u(k) + \frac{T_s}{L_D + T_s R_S} \cdot e(k) - \frac{\Delta L T_s}{L_D + T_s R_S} \cdot \omega_e(k) \cdot i'(k) + \frac{L_D}{L_D + T_s R_S} \cdot i(k-1)$$

Equation 29

Where:

- $i(k) = [i_\gamma, i_\delta]$ is the stator current vector in the actual step
- $i(k - 1) = [i_\gamma, i_\delta]$ is the stator current vector in the previous step
- $u(k) = [u_\gamma, u_\delta]$ is the stator voltage vector in the actual step
- $e(k) = [e_\gamma, e_\delta]$ is the stator back-EMF voltage vector in the actual step
- $i'(k) = [i_\gamma, -i_\delta]$ is the complementary stator current vector in the actual step
- $\omega_e(k)$ is the electrical angular speed in the actual step
- T_S is the sampling time [s]

This equation is transformed into the fractional arithmetic as:

$$i_{sc}(k) \bullet i_{max} = \frac{T_s}{L_D + T_s R_S} \bullet u_{sc}(k) \bullet u_{max} + \frac{T_s}{L_D + T_s R_S} \bullet e_{sc}(k) \bullet e_{max} - \frac{\Delta L T_s}{L_D + T_s R_S} \bullet \omega_{esc}(k) \bullet \omega_{max} \bullet i'_{sc}(k) \bullet i_{max} + \frac{L_D}{L_D + T_s R_S} \bullet i_{sc}(k-1) \bullet i_{max}$$

Equation 30

Where:

- $i_{sc}(k) = [i_\gamma, i_\delta]$ is the scaled stator current vector in the actual step
- $i_{sc}(k - 1) = [i_\gamma, i_\delta]$ is the scaled stator current vector in the previous step
- $u_{sc}(k) = [u_\gamma, u_\delta]$ is the scaled stator voltage vector in the actual step
- $e_{sc}(k) = [e_\gamma, e_\delta]$ is the scaled stator back-EMF voltage vector in the actual step
- $i'_{sc}(k) = [i_\gamma, -i_\delta]$ is the scaled complementary stator current vector in the actual step
- $\omega_{esc}(k)$ is the scaled electrical angular speed in the actual step
- i_{max} is the maximum current [A]
- e_{max} is the maximum back-EMF voltage [V]
- u_{max} is the maximum stator voltage [V]
- ω_{max} is the maximum electrical angular speed in [rad / s]

If the 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. However, this is only valid when the back-EMF term is not included in the observer model. The observer is a closed-loop current observer, therefore, it acts as a state filter for the back-EMF term.

The estimate of the extended EMF term can be derived from Equation 28 on page 43 as:

$$-\frac{\hat{E}_{\gamma\delta}(s)}{E_{\gamma\delta}(s)} = \frac{sK_P + K_I}{s^2 L_D + sR_S + sK_P + K_I}$$

Equation 31

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

$$s^2+\frac{K_P+R_S}{L_D}\bullet s+\frac{K_I}{L_D}=s^2+2\xi\omega_0 s+\omega_0^2$$

Equation 32

where:

- ω_0 is the natural frequency of the closed-loop system (loop bandwidth)
- ξ is the loop attenuation
- K_P is the proporional gain
- K_I is the integral gain

2.6.1 Available versions

This function is available in the following versions:

- Fractional output - the output is the fractional portion of the result; the result is within the range <-1 ; 1). The parameters use the accumulator types.

The available versions of the AMCLIB_PMSMBemfObsrvAB function are shown in the following table:

Table 2-11. Init versions

Function name	Parameters	Result type
AMCLIB_PMSMBemfObsrvABInit_F16	AMCLIB_BEMF_OBSRV_AB_T_A32 *	void
	The initialization does not have an input.	

The available versions of the AMCLIB_PMSMBemfObsrvAB function are shown in the following table:

Table 2-12. Function versions

Function name	Input/output type		Result type
AMCLIB_PMSMBemfObsrvAB_F16	**Input**	GMCLIB_2COOR_ALBE_T_F16 *	void
		GMCLIB_2COOR_ALBE_T_F16 *	
		frac16_t	
	Parameters	AMCLIB_BEMF_OBSRV_AB_T_A32 *	
	The back-EMF observer with a 16-bit fractional input Alpha/Beta current and voltage, and a 16-bit electrical speed. All are within the range <-1 ; 1).		

2.6.2 AMCLIB_BEMF_OBSRV_AB_T_A32 type description

Variable name		Data type	Description
sEObsrv		GMCLIB_2COOR_ALBE_T_F32	The estimated back-EMF voltage structure.
sIObsrv		GMCLIB_2COOR_ALBE_T_F32	The estimated current structure.
sCtrl	f32IAlpha_1	frac32_t	The state variable in the alpha part of the observer, integral part at step k-1. The variable is within the range <-1 ; 1).
	f32IBeta_1	frac32_t	The state variable in the beta part of the observer, integral part at step k-1. The variable is within the range <-1 ; 1).
	a32PGain	acc32_t	The observer proportional gain is set up according to Equation 32 on page 45 as: $(2\xi\omega_0 L_D - R_S)\frac{i_{max}}{e_{max}}$ The parameter is within the range <0 ; 65536.0). Set by the user.
	a32IGain	acc32_t	The observer integral gain is set up according to Equation 32 on page 45 as: $\omega_0^2 L_D T_s \frac{i_{max}}{e_{max}}$ The parameter is within the range <0 ; 65536.0). Set by the user.
a32IGain		acc32_t	The current coefficient gain is set up according to Equation 5 as: $\frac{L_D}{L_D+T_sR_S}$ The parameter is within the range <0 ; 65536.0). Set by the user.
a32UGain		acc32_t	The voltage coefficient gain is set up according to Equation 5 as: $\frac{T_s}{L_D+T_sR_S}\cdot\frac{u_{max}}{i_{max}}$ The parameter is within the range <0 ; 65536.0). Set by the user.
a32WIGain		acc32_t	The angular speed coefficient gain is set up according to Equation 5 as: $\frac{\Delta L T_s}{L_D+T_sR_S}\cdot\omega_{max}$ The parameter is within the range <0 ; 65536.0).Set by the user.
a32EGain		acc32_t	The back-EMF coefficient gain is set up according to Equation 5 as: $\frac{T_s}{L_D+T_sR_S}\cdot\frac{e_{max}}{i_{max}}$ The parameter is within the range <0 ; 65536.0). Set by the user.

Table continues on the next page...

Variable name	Data type	Description
sUnityVctr	GMCLIB_2COOR_SINCOS_T_F16	The output - estimated angle as the sin/cos vector.

2.6.3 Declaration

The available AMCLIB_PMSMBemfObsrvABInit functions have the following declarations:

```
void AMCLIB_PMSMBemfObsrvABInit_F16(AMCLIB_BEMF_OBSRV_AB_T_A32 *psCtrl)
```

The available AMCLIB_PMSMBemfObsrvAB functions have the following declarations:

```
void AMCLIB_PMSMBemfObsrvAB_F16(const GMCLIB_2COOR_ALBE_T_F16 *psIAlBe, const
GMCLIB_2COOR_ALBE_T_F16 *psUAlBe, frac16_t f16Speed, AMCLIB_BEMF_OBSRV_AB_T_A32 *psCtrl)
```

2.6.4 Function use

The use of the AMCLIB_PMSMBemfObsrvAB function is shown in the following examples:

Fixed-point version:

```
#include "amclib.h"

static GMCLIB_2COOR_ALBE_T_F16 sIAlBe, sUAlBe;
static AMCLIB_BEMF_OBSRV_AB_T_A32 sBemfObsrv;
static frac16_t f16Speed;

void Isr(void);

void main (void)
{
  sBemfObsrv.sCtrl.a32PGain= ACC32(1.697);
  sBemfObsrv.sCtrl.a32IGain= ACC32(0.134);
  sBemfObsrv.a32IGain = ACC32(0.986);
  sBemfObsrv.a32UGain = ACC32(0.170);
  sBemfObsrv.a32WIGain= ACC32(0.110);
  sBemfObsrv.a32EGain = ACC32(0.116);

  /* Initialization of the observer's structure */
  AMCLIB_PMSMBemfObsrvABInit_F16(&sBemfObsrv);

  sIAlBe.f16Alpha = FRAC16(0.05);
  sIAlBe.f16Beta  = FRAC16(0.1);
  sUAlBe.f16Alpha = FRAC16(0.2);
  sUAlBe.f16Beta  = FRAC16(-0.1);
```

```
}

/* Periodical function or interrupt */
void Isr(void)
{
  /* BEMF Observer calculation */
  AMCLIB_PMSMBemfObsrvAB_F16(&sIAlBe, &sUAlBe, f16Speed, &sBemfObsrv);
}
```

2.7 AMCLIB_PMSMBemfObsrvDQ

The AMCLIB_PMSMBemfObsrvDQ function calculates the algorithm of back-electro-motive force observer in a rotating reference frame. The method for estimating the rotor position and angular speed is based on the mathematical model of an interior PMSM motor with an extended electro-motive force function, which is realized in an estimated quasi-synchronous reference frame γ-δ as shown in Figure 2-9.

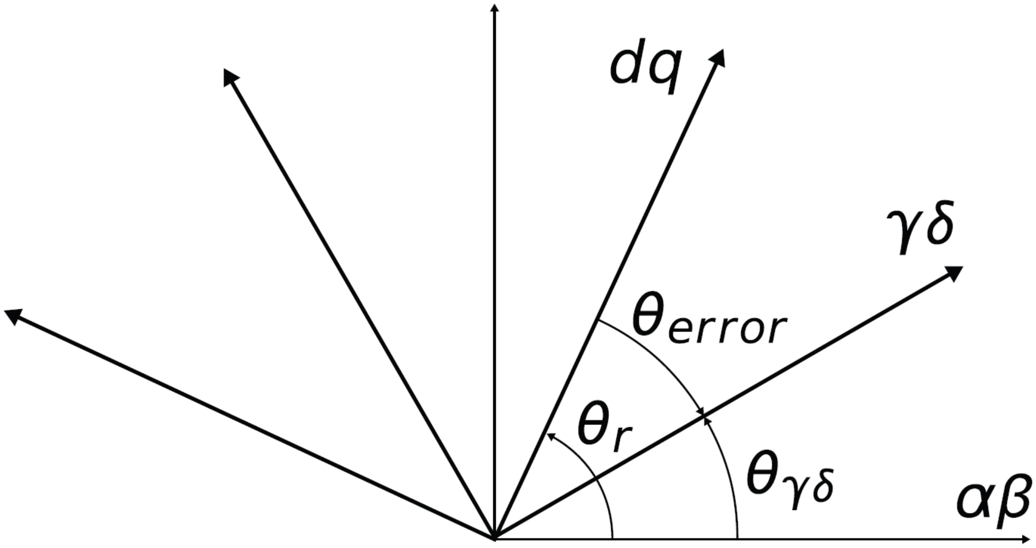


Figure 2-9. The estimated and real rotor *dq* synchronous reference frames

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:

$$\begin{bmatrix} u_\gamma \\ u_\delta \end{bmatrix} = \begin{bmatrix} R_S + sL_D & -\omega_r L_Q \\ \omega_r L_Q & R_S + sL_D \end{bmatrix} \cdot \begin{bmatrix} i_\gamma \\ i_\delta \end{bmatrix} + \left(\Delta L \cdot \left(\omega_r i_D - s i_Q \right) + \Psi_m \omega_r \right) \cdot \begin{bmatrix} -\sin(\theta_{error}) \\ \cos(\theta_{error}) \end{bmatrix}$$

Equation 33

where:

- R_S is the stator resistance
- L_D and L_Q are the D-axis and Q-axis inductances

- Ψ_m is the back-EMF constant
- ω_r is the angular electrical rotor speed
- u_γ and u_δ are the estimated stator voltages
- i_γ and i_δ are the estimated stator currents
- θ_{error} is the error between the actual D-Q frame and the estimated frame position
- s is the operator of the derivative

The block diagram of the observer in the estimated reference frame is shown in Figure 2-10. The observer compensator is substituted by a standard PI controller with following equation in the fractional arithmetic.

$$i_{sc}(k) \cdot i_{max} = K_P \cdot e_{sc}(k) \cdot e_{max} + T_s \cdot K_I \cdot e_{sc}(k) \cdot e_{max} + i_{sc}(k - 1) \cdot i_{max}$$

Equation 34

where:

- K_P is the observer proportional gain [-]
- K_I is the observer integral gain [-]
- $i_{sc}(k) = [i_\gamma, i_\delta]$ is the scaled stator current vector in the actual step
- $i_{sc}(k - 1) = [i_\gamma, i_\delta]$ is the scaled stator current vector in the previous step
- $e_{sc}(k) = [e_\gamma, e_\delta]$ is the scaled stator back-EMF voltage vector in the actual step
- i_{max} is the maximum current [A]
- e_{max} is the maximum back-EMF voltage [V]
- T_S is the sampling time [s]

As shown in Figure 2-10, the observer model and hence also the PI controller gains in both axes are identical to each other.

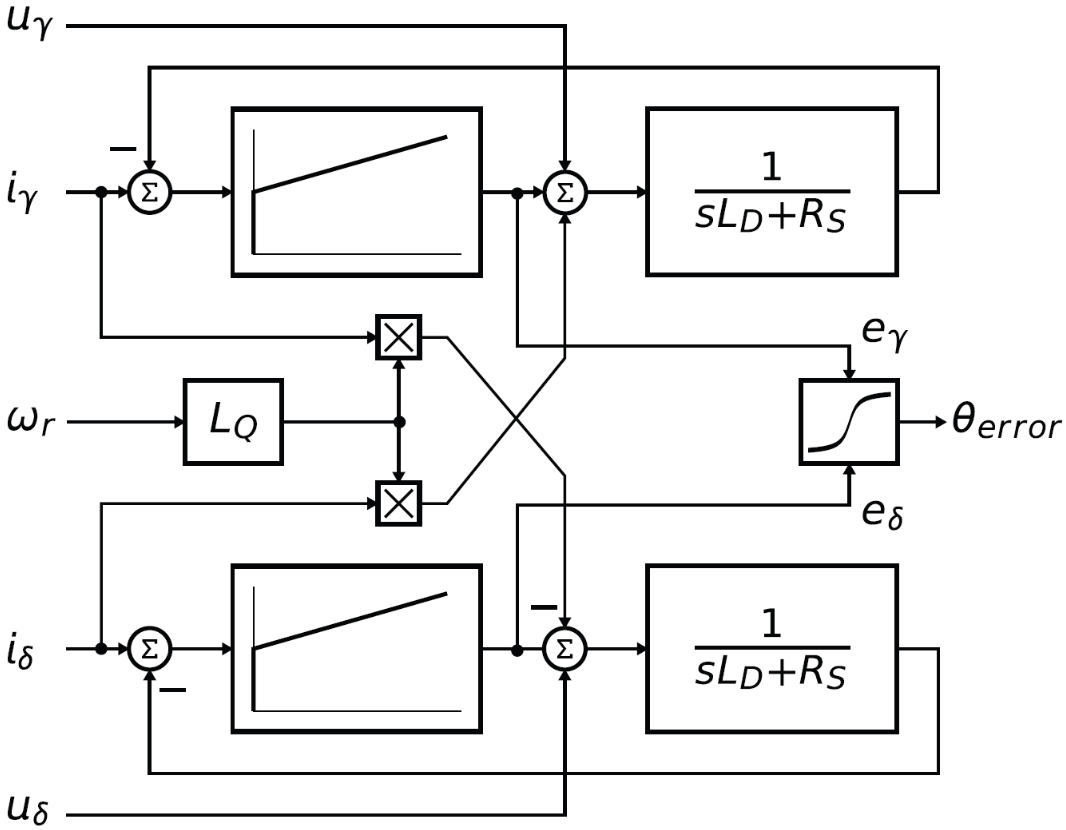


Figure 2-10. Block diagram of proposed Luenberger-type stator current observer acting as state filter for back-EMF

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

$$\theta_{error} = \operatorname{atan}\left(\frac{-e_\gamma}{e_\delta}\right)$$

Equation 35

The estimated position $\hat{\theta}_e$ can be obtained by driving the position of the estimated reference frame 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} zeroed, $\theta_{error} = 0$.

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

$$\hat{E}_{\alpha\beta}(s) = -E_{\alpha\beta}(s) \cdot \frac{F_C(s)}{sL_D + R_S + F_C(s)}$$

Equation 36

The appropriate dynamic behavior of the back-EMF observer is achieved by the 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.

The back-EMF observer is a Luenberger-type observer with a motor model, which is implemented using the backward Euler transformation as follows:

$$i(k) = \frac{T_s}{L_D + T_s R_S} \cdot u(k) + \frac{T_s}{L_D + T_s R_S} \cdot e(k) + \frac{L_Q T_s}{L_D + T_s R_S} \cdot \omega_e(k) \cdot i'(k) + \frac{L_D}{L_D + T_s R_S} \cdot i(k-1)$$

Equation 37

where:

- $i(k) = [i_\gamma, i_\delta]$ is the stator current vector in the actual step
- $i(k - 1) = [i_\gamma, i_\delta]$ is the stator current vector in the previous step
- $u(k) = [u_\gamma, u_\delta]$ is the stator voltage vector in the actual step
- $e(k) = [e_\gamma, e_\delta]$ is the stator back-EMF voltage vector in the actual step
- $i'(k) = [i_\gamma, -i_\delta]$ is the complementary stator current vector in the actual step
- $\omega_e(k)$ is the electrical angular speed in the actual step
- T_S is the sampling time [s]

This equation is transformed into the fractional arithmetic as follows:

$$i_{sc}(k) \cdot i_{max} = \frac{T_s}{L_D + T_s R_S} \cdot u_{sc}(k) \cdot u_{max} + \frac{T_s}{L_D + T_s R_S} \cdot e_{sc}(k) \cdot e_{max} + \frac{L_Q T_s}{L_D + T_s R_S} \cdot \omega_{esc}(k) \cdot \omega_{max} \cdot i'_{sc}(k) \cdot i_{max} + \frac{L_D}{L_D + T_s R_S} \cdot i_{sc}(k-1) \cdot i_{max}$$

Equation 38

where:

- $i_{sc}(k) = [i_\gamma, i_\delta]$ is the scaled stator current vector in the actual step
- $i_{sc}(k - 1) = [i_\gamma, i_\delta]$ is the scaled stator current vector in the previous step
- $u_{sc}(k) = [u_\gamma, u_\delta]$ is the scaled stator voltage vector in the actual step
- $e_{sc}(k) = [e_\gamma, e_\delta]$ is the scaled stator back-EMF voltage vector in the actual step
- $i'_{sc}(k) = [i_\gamma, -i_\delta]$ is the scaled complementary stator current vector in the actual step
- $\omega_{esc}(k)$ is the scaled electrical angular speed in the actual step
- i_{max} is the maximum current [A]
- e_{max} is the maximum back-EMF voltage [V]
- u_{max} is the maximum stator voltage [V]
- ω_{max} is the maximum electrical angular speed in [rad / s]

If the 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. However, this is only valid when the back-EMF term is not included in the observer model. The observer is a closed-loop current observer, therefore it acts as a state filter for the back-EMF term.

The estimate of the extended EMF term can be derived from Equation 36 on page 51 as follows:

$$-\frac{\hat{E}_{\gamma\delta}(s)}{E_{\gamma\delta}(s)} = \frac{sK_P + K_I}{s^2L_D + sR_S + sK_P + K_I}$$

Equation 39

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

$$s^2 + \frac{K_P + R_S}{L_D} \bullet s + \frac{K_I}{L_D} = s^2 + 2\xi\omega_0 s + \omega_0^2$$

Equation 40

where:

- ω_0 is the natural frequency of the closed-loop system (loop bandwith)
- ξ is the loop attenuation
- K_P is the proporional gain
- k_I is the integral gain

2.7.1 Available versions

This function is available in the following versions:

- Fractional output - the output is the fractional portion of the result; the result is within the range <-1 ; 1). The parameters use the accumulator types.
- Accumulator output with floating-point inputs - the output is the accumulator result; the result is within the range <-1 ; 1). The inputs are 32-bit single precision floating-point values.

The available versions of the AMCLIB_PMSMBemfObsrvDQ function are shown in the following table:

Table 2-13. Init versions

Function name	Parameters	Result type
AMCLIB_PMSMBemfObsrvDQInit_F16	AMCLIB_BEMF_OBSRV_DQ_T_A32 *	void
	Initialization does not have any input.	

Table 2-14. Function versions

Function name	Input/output type		Result type
AMCLIB_PMSMBemfObsrvDQ_F16	**Input**	GMCLIB_2COOR_DQ_T_F16 *	frac16_t
		GMCLIB_2COOR_DQ_T_F16 *	
		frac16_t	
	Parameters	AMCLIB_BEMF_OBSRV_DQ_T_A32 *	
	Back-EMF observer with a 16-bit fractional input D-Q current and voltage, and a 16-bit electrical speed. All are within the range <-1 ; 1).		

2.7.2 AMCLIB_BEMF_OBSRV_DQ_T_A32 type description

Variable name		Data type	Description
sEObsrv		GMCLIB_2COOR_DQ_T_F32	Estimated back-EMF voltage structure.
sIObsrv		GMCLIB_2COOR_DQ_T_F32	Estimated current structure.
sCtrl	f32ID_1	frac32_t	State variable in the alpha part of the observer, integral part at step k - 1. The variable is within the range <-1 ; 1).
	f32IQ_1	frac32_t	State variable in the beta part of the observer, integral part at step k - 1. The variable is within the range <-1 ; 1).
	a32PGain	acc32_t	The observer proportional gain is set up according to Equation 40 on page 52 as: $(2\xi\omega_0 L_D - R_S)\frac{i_{max}}{e_{max}}$ The parameter is within the range <0 ; 65536.0). Set by the user.
	a32IGain	acc32_t	The observer integral gain is set up according to Equation 40 on page 52 as: $\omega_0^2 L_D T_s \frac{i_{max}}{e_{max}}$ The parameter is within the range <0 ; 65536.0). Set by the user.
a32IGain		acc32_t	The current coefficient gain is set up according to Equation 38 on page 51 as: $\frac{L_D}{L_D + T_s R_S}$ The parameter is within the range <0 ; 65536.0). Set by the user.
a32UGain		acc32_t	The voltage coefficient gain is set up according to Equation 38 on page 51 as: $\frac{T_s}{L_D + T_s R_S} \cdot \frac{u_{max}}{i_{max}}$

Table continues on the next page...

Variable name	Data type	Description
		The parameter is within the range <0 ; 65536.0). Set by the user.
a32WIGain	acc32_t	The angular speed coefficient gain is set up according to Equation 38 on page 51 as: $$\frac{L_Q T_s}{L_D + T_s R_S} \cdot \omega_{max}$$ The parameter is within the range <0 ; 65536.0). Set by the user.
a32EGain	acc32_t	The back-EMF coefficient gain is set up according to Equation 38 on page 51 as: $$\frac{T_s}{L_D + T_s R_S} \cdot \frac{e_{max}}{i_{max}}$$ The parameter is within the range <0 ; 65536.0). Set by the user.
f16Error	frac16_t	Output - estimated phase error between a real D / Q frame system and an estimated D / Q reference system. The error is within the range <-1 ; 1).

2.7.3 Declaration

The available AMCLIB_PMSMBemfObsrvDQInit functions have the following declarations:

```
void AMCLIB_PMSMBemfObsrvDQInit_F16(AMCLIB_BEMF_OBSRV_DQ_T_A32 *psCtrl)
```

The available AMCLIB_PMSMBemfObsrvDQ functions have the following declarations:

```
frac16_t AMCLIB_PMSMBemfObsrvDQ_F16(const GMCLIB_2COOR_DQ_T_F16 *psIDQ, const
GMCLIB_2COOR_DQ_T_F16 *psUDQ, frac16_t f16Speed, AMCLIB_BEMF_OBSRV_DQ_T_A32 *psCtrl)
```

2.7.4 Function use

The use of the AMCLIB_PMSMBemfObsrvDQ function is shown in the following example:

```
#include "amclib.h"

static GMCLIB_2COOR_DQ_T_F16       sIdq, sUdq;
static AMCLIB_BEMF_OBSRV_DQ_T_A32  sBemfObsrv;
static frac16_t f16Speed, f16Error;
```

```
void Isr(void);

void main (void)
{
  sBemfObsrv.sCtrl.a32PGain= ACC32(1.697);
  sBemfObsrv.sCtrl.a32IGain= ACC32(0.134);
  sBemfObsrv.a32IGain = ACC32(0.986);
  sBemfObsrv.a32UGain = ACC32(0.170);
  sBemfObsrv.a32WIGain= ACC32(0.110);
  sBemfObsrv.a32EGain = ACC32(0.116);

  /* Initialization of the observer's structure */
  AMCLIB_PMSMBemfObsrvDQInit_F16(&sBemfObsrv);

  sIdq.f16D = FRAC16(0.05);
  sIdq.f16Q = FRAC16(0.1);
  sUdq.f16D = FRAC16(0.2);
  sUdq.f16Q = FRAC16(-0.1);
}

/* Periodical function or interrupt */
void Isr(void)
{
  /* BEMF Observer calculation */
  f16Error = AMCLIB_PMSMBemfObsrvDQ_F16(&sIdq, &sUdq, f16Speed, &sBemfObsrv);
}
```

2.8 AMCLIB_TrackObsrv

The AMCLIB_TrackObsrv function calculates a tracking observer for the determination of angular speed and position of the input error functional signal. The tracking-observer algorithm uses the phase-locked-loop mechanism. It is recommended to call this function at every sampling period. It requires a single input argument as a phase error. A phase-tracking observer with a standard PI controller used as the loop compensator is shown in Figure 2-11.

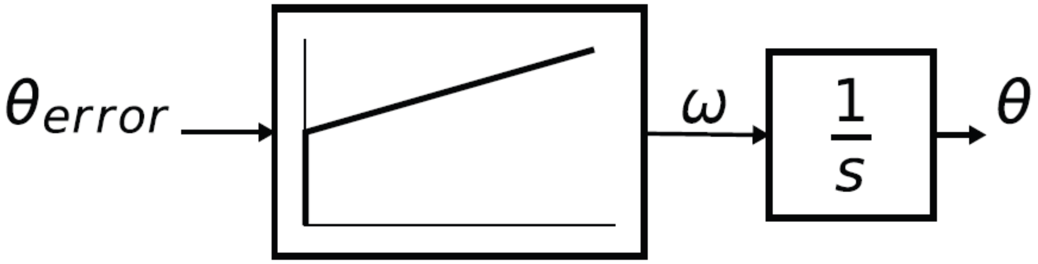


Figure 2-11. Block diagram of proposed PLL scheme for position estimation

The depicted tracking observer structure has the following transfer function:

$$\frac{\hat{\theta}(s)}{\theta(s)} = \frac{sK_P+K_I}{s^2+sK_P+K_I}$$

Equation 41

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 the block scheme in Figure 2-11 are as follows:

$$\omega(k) = K_P \bullet e(k) + T_s \bullet K_I \bullet e(k) + \omega(k-1)$$

Equation 42

$$\theta(k) = T_s \cdot \omega(k) + \theta(k-1)$$

Equation 43

where:

- K_P is the proportional gain
- K_I is the integral gain
- T_s is the sampling period [s]
- e(k) is the position error in step k
- ω(k) is the rotor speed [rad / s] in step k
- ω(k - 1) is the rotor speed [rad / s] in step k - 1
- θ(k) is the rotor angle [rad] in step k
- θ(k - 1) is the rotor angle [rad] in step k - 1

In the fractional arithmetic, Equation 41 on page 55 and Equation 42 on page 56 are as follows:

$$\omega_{sc}(k) \cdot \omega_{max} = K_P \cdot e_{sc}(k) + T_s \cdot K_I \cdot e_{sc}(k) + \omega_{sc}(k-1) \cdot \omega_{max}$$

Equation 44

$$\theta_{sc}(k) \cdot \theta_{max} = T_s \cdot \omega_{sc}(k) \cdot \omega_{max} + \theta_{sc}(k-1) \cdot \theta_{max}$$

Equation 45

where:

- $e_{sc}(k)$ is the scaled position error in step k
- $\omega_{sc}(k)$ is the scaled rotor speed [rad / s] in step k
- $\omega_{sc}(k-1)$ is the scaled rotor speed [rad / s] in step k - 1
- $\theta_{sc}(k)$ is the scaled rotor angle [rad] in step k
- $\theta_{sc}(k-1)$ is the scaled rotor angle [rad] in step k - 1
- ω_{max} is the maximum speed
- θ_{max} is the maximum rotor angle (typically)

2.8.1 Available versions

The function is available in the following versions:

- Fractional output - the output is the fractional portion of the result; the result is within the range <-1 ; 1).

The available versions of the AMCLIB_TrackObsrv function are shown in the following table:

Table 2-15. Init versions

Function name	Init angle	Parameters	Result type
AMCLIB_TrackObsrvInit_F16	frac16_t	AMCLIB_TRACK_OBSRV_T_F32 *	void
	The input is a 16-bit fractional value of the angle normalized to the range <-1 ; 1) that represents an angle (in radians) within the range <-π ; π).		

Table 2-16. Function versions

Function name	Input type	Parameters	Result type
AMCLIB_TrackObsrv_F16	frac16_t	AMCLIB_TRACK_OBSRV_T_F32 *	frac16_t
	Tracking observer with a 16-bit fractional position error input divided by π. The output from the obsever is a 16-bit fractional position normalized to the range <-1 ; 1) that represents an angle (in radians) within the range <-π ; π).		

2.8.2 AMCLIB_TRACK_OBSRV_T_F32

Variable name	Input type	Description
f32Theta	frac32_t	Estimated position as the output of the second numerical integrator. The parameter is within the range <-1 ; 1). Controlled by the algorithm.
f32Speed	frac32_t	Estimated speed as the output of the first numerical integrator. The parameter is within the range <-1 ; 1). Controlled by the algorithm.
f32I_1	frac32_t	State variable in the controller part of the observer; integral part at step k - 1. The parameter is within the range <-1 ; 1). Controlled by the algorithm.
f16IGain	frac16_t	The observer integral gain is set up according to Equation 44 on page 56 as: $T_s \cdot K_I \cdot \frac{1}{\omega_{max}} \cdot 2^{-Ish}$ The parameter is a 16-bit fractional type within the range <0 ; 1). Set by the user.
i16IGainSh	int16_t	The observer integral gain shift takes care of keeping the f16IGain variable within the fractional range <-1 ; 1). The shift is determined as: $\log_2(T_s \cdot K_I \cdot \frac{1}{\omega_{max}}) - \log_2 1 < Ish \le \log_2(T_s \cdot K_I \cdot \frac{1}{\omega_{max}}) - \log_2 0.5$ The parameter is a 16-bit integer type within the range <-15 ; 15>. Set by the user.

Table continues on the next page...

Variable name	Input type	Description
f16PGain	frac16_t	The observer proportional gain is set up according to Equation 44 on page 56 as: $K_P \cdot \frac{1}{\omega_{max}} \cdot 2^{-Psh}$ The parameter is a 16-bit fractional type within the range <0 ; 1). Set by the user.
i16PGainSh	int16_t	The observer proportional gain shift takes care of keeping the f16PGain variable within the fractional range <-1 ; 1). The shift is determined as: $\log_2(K_P \cdot \frac{1}{\omega_{max}}) - \log_2 1 < Psh \le \log_2(K_P \cdot \frac{1}{\omega_{max}}) - \log_2 0.5$ The parameter is a 16-bit integer type within the range <-15 ; 15>. Set by the user.
f16ThGain	frac16_t	The observer gain for the output position integrator is set up according to Equation 45 on page 56 as: $T_s \cdot \frac{\omega_{max}}{\theta_{max}} \cdot 2^{-Thsh}$ The parameter is a 16-bit fractional type within the range <0 ; 1). Set by the user.
i16ThGainSh	int16_t	The observer gain shift for the position integrator takes care of keeping the f16ThGain variable within the fractional range <-1 ; 1). The shift is determined as: $\log_2(T_s \cdot \frac{\omega_{max}}{\theta_{max}}) - \log_2 1 < THsh \le \log_2(T_s \cdot \frac{\omega_{max}}{\theta_{max}}) - \log_2 0.5$ The parameter is a 16-bit integer type within the range <-15 ; 15>. Set by the user.

2.8.3 Declaration

The available AMCLIB_TrackObsrvInit functions have the following declarations:

```
void AMCLIB_TrackObsrvInit_F16(frac16_t f16ThetaInit, AMCLIB_TRACK_OBSRV_T_F32 *psCtrl)
```

The available AMCLIB_TrackObsrv functions have the following declarations:

```
frac16_t AMCLIB_TrackObsrv_F16(frac16_t f16Error, AMCLIB_TRACK_OBSRV_T_F32 *psCtrl)
```

2.8.4 Function use

The use of the AMCLIB_TrackObsrv function is shown in the following example:

```
#include "amclib.h"

static AMCLIB_TRACK_OBSRV_T_F32  sTo;
static frac16_t      f16ThetaError;
static frac16_t      f16PositionEstim;

void Isr(void);

void main(void)
{
  sTo.f16IGain     = FRAC16(0.6434);
```

```
  sTo.i16IGainSh   = -9;
  sTo.f16PGain     = FRAC16(0.6801);
  sTo.i16PGainSh   = -2;
  sTo.f16ThGain    = FRAC16(0.6400);
  sTo.i16ThGainSh  = -4;

  AMCLIB_TrackObsrvInit_F16(FRAC16(0.0), &sTo);

  f16ThetaError    = FRAC16(0.5);
}

/* Periodical function or interrupt */
void Isr(void)
{
  /* Tracking observer calculation */
  f16PositionEstim = AMCLIB_TrackObsrv_F16(f16ThetaError, &sTo);
}
```

Appendix A
Library types

A.1 bool_t

The bool_t type is a logical 16-bit type. It is able to store the boolean variables with two states: TRUE (1) or FALSE (0). Its definition is as follows:

```
typedef unsigned short bool_t;
```

The following figure shows the way in which the data is stored by this type:

Table A-1. Data storage

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Value	Unused															Logical
TRUE	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
	0				0				0				1			
FALSE	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0				0				0				0			

To store a logical value as bool_t, use the FALSE or TRUE macros.

A.2 uint8_t

The uint8_t type is an unsigned 8-bit integer type. It is able to store the variables within the range <0 ; 255>. Its definition is as follows:

```
typedef unsigned char uint8_t;
```

The following figure shows the way in which the data is stored by this type:

Table A-2. Data storage

	7	6	5	4	3	2	1	0
Value	Integer							
255	1	1	1	1	1	1	1	1
	F				F			
11	0	0	0	0	1	0	1	1
	0				B			
124	0	1	1	1	1	1	0	0
	7				C			
159	1	0	0	1	1	1	1	1
	9				F			

A.3 uint16_t

The uint16_t type is an unsigned 16-bit integer type. It is able to store the variables within the range <0 ; 65535>. Its definition is as follows:

```
typedef unsigned short uint16_t;
```

The following figure shows the way in which the data is stored by this type:

Table A-3. Data storage

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Value	Integer															
65535	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	F				F				F				F			
5	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1
	0				0				0				5			
15518	0	0	1	1	1	1	0	0	1	0	0	1	1	1	1	0
	3				C				9				E			
40768	1	0	0	1	1	1	1	1	0	1	0	0	0	0	0	0
	9				F				4				0			

A.4 uint32_t

The uint32_t type is an unsigned 32-bit integer type. It is able to store the variables within the range <0 ; 4294967295>. Its definition is as follows:

```
typedef unsigned long uint32_t;
```

The following figure shows the way in which the data is stored by this type:

Table A-4. Data storage

	31	24	23	16	15	8	7	0
Value	Integer							
4294967295	F	F	F	F	F	F	F	F
2147483648	8	0	0	0	0	0	0	0
55977296	0	3	5	6	2	5	5	0
3451051828	C	D	B	2	D	F	3	4

A.5 int8_t

The int8_t type is a signed 8-bit integer type. It is able to store the variables within the range <-128 ; 127>. Its definition is as follows:

```
typedef char int8_t;
```

The following figure shows the way in which the data is stored by this type:

Table A-5. Data storage

	7	6	5	4	3	2	1	0
Value	Sign	Integer						
127	0	1	1	1	1	1	1	1
	7				F			
-128	1	0	0	0	0	0	0	0
	8				0			
60	0	0	1	1	1	1	0	0
	3				C			
-97	1	0	0	1	1	1	1	1
	9				F			

A.6 int16_t

The int16_t type is a signed 16-bit integer type. It is able to store the variables within the range <-32768 ; 32767>. Its definition is as follows:

```
typedef short int16_t;
```

The following figure shows the way in which the data is stored by this type:

Table A-6. Data storage

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Value	Sign	Integer														
32767	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	7				F				F				F			
-32768	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	8				0				0				0			
15518	0	0	1	1	1	1	0	0	1	0	0	1	1	1	1	0
	3				C				9				E			
-24768	1	0	0	1	1	1	1	1	0	1	0	0	0	0	0	0
	9				F				4				0			

A.7 int32_t

The int32_t type is a signed 32-bit integer type. It is able to store the variables within the range <-2147483648 ; 2147483647>. Its definition is as follows:

```
typedef long int32_t;
```

The following figure shows the way in which the data is stored by this type:

Table A-7. Data storage

	31	24	23	16	15	8	7	0
Value	S	Integer						
2147483647	7	F	F	F	F	F	F	F
-2147483648	8	0	0	0	0	0	0	0
55977296	0	3	5	6	2	5	5	0
-843915468	C	D	B	2	D	F	3	4

A.8 frac8_t

The frac8_t type is a signed 8-bit fractional type. It is able to store the variables within the range <-1 ; 1). Its definition is as follows:

```
typedef char frac8_t;
```

The following figure shows the way in which the data is stored by this type:

Table A-8. Data storage

	7	6	5	4	3	2	1	0
Value	Sign	Fractional						
0.99219	0	1	1	1	1	1	1	1
	7				F			
-1.0	1	0	0	0	0	0	0	0
	8				0			
0.46875	0	0	1	1	1	1	0	0
	3				C			
-0.75781	1	0	0	1	1	1	1	1
	9				F			

To store a real number as frac8_t, use the FRAC8 macro.

A.9 frac16_t

The frac16_t type is a signed 16-bit fractional type. It is able to store the variables within the range <-1 ; 1). Its definition is as follows:

```
typedef short frac16_t;
```

The following figure shows the way in which the data is stored by this type:

Table A-9. Data storage

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Value	Sign	Fractional														
0.99997	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	7				F				F				F			
-1.0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table continues on the next page...

Table A-9. Data storage (continued)

	8				0				0				0			
0.47357	0	0	1	1	1	1	0	0	1	0	0	1	1	1	1	0
	3				C				9				E			
-0.75586	1	0	0	1	1	1	1	1	0	1	0	0	0	0	0	0
	9				F				4				0			

To store a real number as frac16_t, use the FRAC16 macro.

A.10 frac32_t

The frac32_t type is a signed 32-bit fractional type. It is able to store the variables within the range <-1 ; 1). Its definition is as follows:

```
typedef long frac32_t;
```

The following figure shows the way in which the data is stored by this type:

Table A-10. Data storage

	31	24	23	16	15	8	7	0
Value	S	Fractional						
0.9999999995	7	F	F	F	F	F	F	F
-1.0	8	0	0	0	0	0	0	0
0.02606645970	0	3	5	6	2	5	5	0
-0.3929787632	C	D	B	2	D	F	3	4

To store a real number as frac32_t, use the FRAC32 macro.

A.11 acc16_t

The acc16_t type is a signed 16-bit fractional type. It is able to store the variables within the range <-256 ; 256). Its definition is as follows:

```
typedef short acc16_t;
```

The following figure shows the way in which the data is stored by this type:

Table A-11. Data storage

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Value	Sign	Integer								Fractional						
255.9921875	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	7				F				F				F			
-256.0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	8				0				0				0			
1.0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
	0				0				8				0			
-1.0	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0
	F				F				8				0			
13.7890625	0	0	0	0	0	1	1	0	1	1	1	0	0	1	0	1
	0				6				E				5			
-89.71875	1	1	0	1	0	0	1	1	0	0	1	0	0	1	0	0
	D				3				2				4			

To store a real number as acc16_t, use the ACC16 macro.

A.12 acc32_t

The acc32_t type is a signed 32-bit accumulator type. It is able to store the variables within the range <-65536 ; 65536). Its definition is as follows:

```
typedef long acc32_t;
```

The following figure shows the way in which the data is stored by this type:

Table A-12. Data storage

	31	24	23	16	15	8	7	0
Value	S	Integer			Fractional			
65535.999969	7	F	F	F	F	F	F	F
-65536.0	8	0	0	0	0	0	0	0
1.0	0	0	0	0	8	0	0	0
-1.0	F	F	F	F	8	0	0	0
23.789734	0	0	0	B	E	5	1	6
-1171.306793	F	D	B	6	5	8	B	C

To store a real number as acc32_t, use the ACC32 macro.

A.13 GMCLIB_3COOR_T_F16

The GMCLIB_3COOR_T_F16 structure type corresponds to the three-phase stationary coordinate system, based on the A, B, and C components. Each member is of the frac16_t data type. The structure definition is as follows:

```
typedef struct
{
    frac16_t f16A;
    frac16_t f16B;
    frac16_t f16C;
} GMCLIB_3COOR_T_F16;
```

The structure description is as follows:

Table A-13. GMCLIB_3COOR_T_F16 members description

Type	Name	Description
frac16_t	f16A	A component; 16-bit fractional type
frac16_t	f16B	B component; 16-bit fractional type
frac16_t	f16C	C component; 16-bit fractional type

A.14 GMCLIB_2COOR_ALBE_T_F16

The GMCLIB_2COOR_ALBE_T_F16 structure type corresponds to the two-phase stationary coordinate system, based on the Alpha and Beta orthogonal components. Each member is of the frac16_t data type. The structure definition is as follows:

```
typedef struct
{
    frac16_t f16Alpha;
    frac16_t f16Beta;
} GMCLIB_2COOR_ALBE_T_F16;
```

The structure description is as follows:

Table A-14. GMCLIB_2COOR_ALBE_T_F16 members description

Type	Name	Description
frac16_t	f16Apha	α-component; 16-bit fractional type
frac16_t	f16Beta	β-component; 16-bit fractional type

A.15 GMCLIB_2COOR_DQ_T_F16

The GMCLIB_2COOR_DQ_T_F16 structure type corresponds to the two-phase rotating coordinate system, based on the D and Q orthogonal components. Each member is of the frac16_t data type. The structure definition is as follows:

```
typedef struct
{
    frac16_t f16D;
    frac16_t f16Q;
} GMCLIB_2COOR_DQ_T_F16;
```

The structure description is as follows:

Table A-15. GMCLIB_2COOR_DQ_T_F16 members description

Type	Name	Description
frac16_t	f16D	D-component; 16-bit fractional type
frac16_t	f16Q	Q-component; 16-bit fractional type

A.16 GMCLIB_2COOR_DQ_T_F32

The GMCLIB_2COOR_DQ_T_F32 structure type corresponds to the two-phase rotating coordinate system, based on the D and Q orthogonal components. Each member is of the frac32_t data type. The structure definition is as follows:

```
typedef struct
{
    frac32_t f32D;
    frac32_t f32Q;
} GMCLIB_2COOR_DQ_T_F32;
```

The structure description is as follows:

Table A-16. GMCLIB_2COOR_DQ_T_F32 members description

Type	Name	Description
frac32_t	f32D	D-component; 32-bit fractional type
frac32_t	f32Q	Q-component; 32-bit fractional type

A.17 GMCLIB_2COOR_SINCOS_T_F16

The GMCLIB_2COOR_SINCOS_T_F16 structure type corresponds to the two-phase coordinate system, based on the Sin and Cos components of a certain angle. Each member is of the frac16_t data type. The structure definition is as follows:

```
typedef struct
{
    frac16_t f16Sin;
    frac16_t f16Cos;
} GMCLIB_2COOR_SINCOS_T_F16;
```

The structure description is as follows:

Table A-17. GMCLIB_2COOR_SINCOS_T_F16 members description

Type	Name	Description
frac16_t	f16Sin	Sin component; 16-bit fractional type
frac16_t	f16Cos	Cos component; 16-bit fractional type

A.18 FALSE

The FALSE macro serves to write a correct value standing for the logical FALSE value of the bool_t type. Its definition is as follows:

```
#define FALSE    ((bool_t)0)
```

```
#include "mlib.h"

static bool_t bVal;

void main(void)
{
  bVal = FALSE;               /* bVal = FALSE */
}
```

A.19 TRUE

The TRUE macro serves to write a correct value standing for the logical TRUE value of the bool_t type. Its definition is as follows:

```
#define TRUE     ((bool_t)1)
```

```
#include "mlib.h"

static bool_t bVal;
```

```
void main(void)
{
  bVal = TRUE;                /* bVal = TRUE */
}
```

A.20 FRAC8

The FRAC8 macro serves to convert a real number to the frac8_t type. Its definition is as follows:

```
#define FRAC8(x) ((frac8_t)((x) < 0.9921875 ? ((x) >= -1 ? (x)*0x80 : 0x80) : 0x7F))
```

The input is multiplied by 128 (=2^7). The output is limited to the range <0x80 ; 0x7F>, which corresponds to <-1.0 ; 1.0-2^{-7}>.

```
#include "mlib.h"

static frac8_t f8Val;

void main(void)
{
  f8Val = FRAC8(0.187);                /* f8Val = 0.187 */
}
```

A.21 FRAC16

The FRAC16 macro serves to convert a real number to the frac16_t type. Its definition is as follows:

```
#define FRAC16(x) ((frac16_t)((x) < 0.999969482421875 ? ((x) >= -1 ? (x)*0x8000 : 0x8000) :
0x7FFF))
```

The input is multiplied by 32768 (=2^{15}). The output is limited to the range <0x8000 ; 0x7FFF>, which corresponds to <-1.0 ; 1.0-2^{-15}>.

```
#include "mlib.h"

static frac16_t f16Val;

void main(void)
{
  f16Val = FRAC16(0.736);                /* f16Val = 0.736 */
}
```

A.22 FRAC32

The FRAC32 macro serves to convert a real number to the frac32_t type. Its definition is as follows:

```
#define FRAC32(x)  ((frac32_t)((x) < 1 ? ((x) >= -1 ? (x)*0x80000000 : 0x80000000) :
0x7FFFFFFF))
```

The input is multiplied by 2147483648 (=2^{31}). The output is limited to the range <0x80000000 ; 0x7FFFFFFF>, which corresponds to <-1.0 ; 1.0-2^{-31}>.

```
#include "mlib.h"

static frac32_t f32Val;

void main(void)
{
  f32Val = FRAC32(-0.1735667);              /* f32Val = -0.1735667 */
}
```

A.23 ACC16

The ACC16 macro serves to convert a real number to the acc16_t type. Its definition is as follows:

```
#define ACC16(x)  ((acc16_t)((x) < 255.9921875 ? ((x) >= -256 ? (x)*0x80 : 0x8000) : 0x7FFF))
```

The input is multiplied by 128 (=2^7). The output is limited to the range <0x8000 ; 0x7FFF> that corresponds to <-256.0 ; 255.9921875>.

```
#include "mlib.h"

static acc16_t a16Val;

void main(void)
{
  a16Val = ACC16(19.45627);               /* a16Val = 19.45627 */
}
```

A.24 ACC32

The ACC32 macro serves to convert a real number to the acc32_t type. Its definition is as follows:

```
#define ACC32(x) ((acc32_t)((x) < 65535.999969482421875 ? ((x) >= -65536 ? (x)*0x8000 :
0x80000000) : 0x7FFFFFFF))
```

The input is multiplied by 32768 (=2^{15}). The output is limited to the range <0x80000000 ; 0x7FFFFFFF>, which corresponds to <-65536.0 ; 65536.0-2^{-15}>.

```
#include "mlib.h"

static acc32_t a32Val;

void main(void)
{
  a32Val = ACC32(-13.654437);               /* a32Val = -13.654437 */
}
```

How to Reach Us:

Home Page:
nxp.com

Web Support:
nxp.com/support

Information in this document is provided solely to enable system and software implementers to use Freescale products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. Freescale reserves the right to make changes without further notice to any products herein.

Freescale makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in Freescale data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including “typicals,” must be validated for each customer application by customer's technical experts. Freescale does not convey any license under its patent rights nor the rights of others. Freescale sells products pursuant to standard terms and conditions of sale, which can be found at the following address: www.freescale.com/salestermsandconditions.

Freescale and the Freescale logo are trademarks of Freescale Semiconductor, Inc. All other product or service names are the property of their respective owners.

© 2019 NXP B.V.